UNIVERSITY OF MACEDONIA
DEPARTMENT OF APPLIED INFORMATICS
GRADUATE PROGRAM

The role and functionality of Choreography diagrams for process-driven applications

M.Sc. THESIS
of
Eleni Itsou

Thessaloniki, October 2023

The role and functionality of Choreography diagrams for process-driven applications

Eleni Itsou

Integrated Msc of Rural and Surveying Engineering, 2019

M.Sc. Thesis
submitted as a partial fulfillment of the requirements for
THE DEGREE OF MASTER OF SCIENCE IN APPLIED INFORMATICS

Supervisor: Dr Konstantinos Vergidis

Abstract

The focus of this thesis is the journey of choreography diagrams in the context of BPMN. Collaboration diagrams show how various participants interact within a business process. On the other hand, choreography diagrams focus on the interactions between participants and act as contracts that specify the messages exchanged and flow of interactions, while removing the focus from the internal processes of an individual participant. A real-world reservation process inspired by Airbnb is used to demonstrate these ideas. This helps to clarify the complex web interactions that occur between hosts and guests and improves understanding through collaboration and choreography diagrams. The research highlights the critical role that REST APIs play in application development before highlighting how important it is to include RESTful interactions in BPMN choreography diagrams. A practical layer is added to the research with building a Reservation REST API using the Model-View-Controller (MVC) design pattern. It elaborates on the four fundamental layers of the API and offers thorough insights into the project setup, documentation, and testing procedures. An in-depth analysis of three reservation scenarios shows how different API methods interact in a dynamic way. The report provides a thorough manual for developers and business analysts and emphasizes the value of automating API request workflows with Postman for effective issue identification and system functionality verification.

Keywords: Business Process Management, BPMN, Business Process Modeling, Choreography Diagrams, Collaboration Diagrams, REST APIs, REST Annotations, Workflow Management

Acknowledgments

I want to express my gratitude to everyone who has helped me along the way as I finish my thesis. I want to start by sincerely thanking Dr. Konstantinos Vergidis, my supervisor, for believing in me right from the start of our work together. His kind and encouraging approach was invaluable in helping me accomplish our final objective and acting as a guiding light during challenging times.

In addition, I would like to thank my good friends and colleagues for their unwavering support of in this long research journey. In addition to the knowledge, I acquired in my field of study, their influence and support had a major impact in my substantial personal growth.

Table of Contents

List of Figures

List of Tables

Locations of projects included in this master thesis:

Reservation API	https://github.com/itsoueleni/reservation/
Postman collection	https://blue-rocket-780039.postman.co/workspace/Reservation-API~5e74902e-08a2-41ff-8b46-b9e9f968385c/overview

Chapter 1 : Introduction

1.1 Background and motivation for the study

The study is motivated by the need to comprehensively explore the visualization and representation of participant interactions within the BPMN framework. The goal of this study is to better understand the different functions of choreographic diagrams, which highlight participant interactions and message exchanges, and collaboration diagrams, which explore each participant's internal operations (white-box collaboration diagrams). These ideas are demonstrated with a real-world example, modeled after Airbnb, featuring a reservation service that acts as a coordinator between hosts and guests.

The growing importance of Representational State Transfer (REST) Application Programming Interfaces (APIs) in application development is also a driving force behind the study. More specifically REST and how can be combined with BPMN choreography diagrams acted as motivation for this thesis. Consequently, the study aims to introduce the idea of REST annotations within a business process.

In conclusion, the context and inspiration for this research stem from the field of BPM, where participant interaction visualization, the complementary roles of collaboration and choreography diagrams, and the increasing significance of RESTful APIs in application development all come together. Business analysts, application developers, and BPM practitioners may all benefit from the practical insights that this research project is prepared to offer and expand upon.

1.2 Overview of the thesis structure

This thesis is organized to offer a methodical and thorough investigation of different aspects of Business Process Management (BPM) and the incorporation of REST Application Programming Interfaces (APIs) into the BPMN framework. It is broken up into multiple chapters, each of which adds to the overall objective of improving our comprehension of BPM and its real-world uses. An outline of the thesis structure is given below:

Chapter 2: BPM's Historical Development

This chapter examines the historical development of BPM. It traces the origins of business process management (BPM) back to Six Sigma techniques, Total Quality Management, Business Process Reengineering (BPR), and continuous improvement tactics. It looks at how Process-Driven Applications (PDAs) came about because of the introduction of IT systems, such as Workflow Management Systems and Enterprise Resource Planning (ERP) systems.

Chapter 3: BPMN Collaboration and Choreography

The chapter explores the fundamental ideas of BPMN. It breaks down the separate roles that choreography and collaboration diagrams play in displaying participant interactions. Collaboration diagrams clarify the relationships between numerous entities involved in a business process. Conversely, choreography diagrams focus more on the logical flow of interactions and message exchanges. These ideas are demonstrated in action using a real-world scenario that draws inspiration from Airbnb's reservation service.

Chapter 4: Utilizing RESTful APIs with BPMN Choreographies Integration

In this chapter, the importance of RESTful APIs for application development is highlighted. It discusses how to incorporate REST annotations into BPMN choreography diagrams and presents the idea of doing so inside a business process. Using a reservation application as an example, the practical implementation of RESTful interactions is shown, highlighting the significance of HTTP request/response exchanges and email notifications. This chapter demonstrates how RESTful interactions help close the gap between process modeling and API implementation, resulting in the creation of dependable, interoperable, and efficient systems.

Chapter 5: Using MVC to Create a Reservation REST API

This chapter focuses on the Model-View-Controller (MVC) design pattern and how it is used to build a Reservation REST API. The Repository layer, the Service layer (which handles business logic), and the Model layer (which depicts the entities and structure of the application) are the four fundamental layers of the API. In-depth instructions on configuring the REST API, connecting to the database with Spring Web and Spring Data MongoDB requirements, and generating API documentation using Spring Fox Swagger UI are all included in this chapter. Additionally, it provides an overview of the Reservation REST API's endpoints and features.

Chapter 6: Postman Scenarios and Automation

In Chapter 6, several reservation scenarios are covered, each of which illustrates how distinct API methods interact with one another. It highlights how important it is to use Postman, an automation tool, to make API request operations more efficient. Postman's features are demonstrated, including response validation, script integration, status code verification, and test reporting. The workflow is guaranteed to function as intended by the automatic execution of API collections, which generates a report with test results.

Chapter 7: The Role of Research, Its Limitations, and Upcoming Projects

The main contributions of this research are outlined in Chapter 7. It draws attention to the creative way that REST-specific annotations can enhance BPMN choreography diagrams, making the shift from choreography models to RESTful HTTP conversations easier. The chapter also recognizes the value of REST APIs in application development as well as the insights obtained from choreography diagrams and cooperation. It highlights how important Postman is as an API operation testing and automation tool. The chapter ends with a summary of the study's shortcomings and recommendations for further investigation.

Chapter 2 : Theoretical Background

2.1 Business Process Definition

In the current business landscape, organizations are facing fierce global competition, customers with fluctuating needs, and faster product lifecycles. To succeed in this challenging environment, organizations need to become faster and more flexible. They should adopt a horizontal or process-centric model with the aim of staying efficient and responsive to the changing marketing demands. Improvement goals within an organization are interrelated with business objectives. Cost reduction, shorter execution times, and lower error rates are common examples of improvement goals. Business process Management (BPM) is about managing a complete series of events, decisions and actions that provide value to the business and its customers. These series of events activities and decisions is referred to as a business process [1].
Every organization from a nonprofit to an enterprise must manage a series of business processes. Typical business processes that can be encountered in any organization are listed below:

- Order to pay: This process is carried out when a customer places an order to acquire a service or product, and it is completed when the product in question has been shipped and the payment is made. This type of business process can be encountered in everyday life from conventional food delivery services to e-commerce orders.
- Procure to pay: This process refers to an organization getting supplies. It is initiated by a person in charge who makes a quote for the purchase, and it gets approved by the administration. Other steps include selecting a vendor, issuing an order, receiving the products, and checking the invoice.
- Issue to resolution: This process begins when a customer raises a complaint while using a service. This process proceeds until the parties-customer and client agree that the problem has been successfully resolved.
- Application for approval: This process is initiated by the petition of a benefit and ends when the outcome of the petition is made as approved or rejected. One of these types of processes is student applications for a master’s degree program. The process begins when the student applies to the program and ends with the approval or denial of the application.

Many business process definitions have been mentioned in literature over the years. According to Hammer & Champy (1993), a business process involves a collection of activities with a set of inputs that create outputs aiming at providing value to the customer. Davenport, (1993), also stated that a business process is a chain of activities whose primary objective is producing a specific outcome for a given market or customer. In addition, Jacobson et al. (1994) defines a business process as 'the set of internal value-adding activities to serve a costumer.' A business process was later defined, incorporating the concept of business operation, as 'a collective set of tasks that are properly connected and sequenced' [2]. The primary objective of a business process is to execute a business operation aiming at delivering value to the organization. In addition, a business process must have discreet inputs and outputs as well as a purpose inside or between businesses [3].

As the former definitions indicate, the quality of business processes has an impact not only on the business efficiency in internal processes but also on customer satisfaction as better processes lead to a better quality of service. The ingredients of a business process might be activities or tasks as the basic elements that use the inputs or the resources (e.g., equipment or material) to achieve the intended outcomes that give value to the intended customers. These activities refer to different decision points and events (Figure 1**).** Actors (organizations or human actors) can also be perceived as a structural element of a business process or as an external entity that executes the business process [2] and they may be responsible to perform different activities.

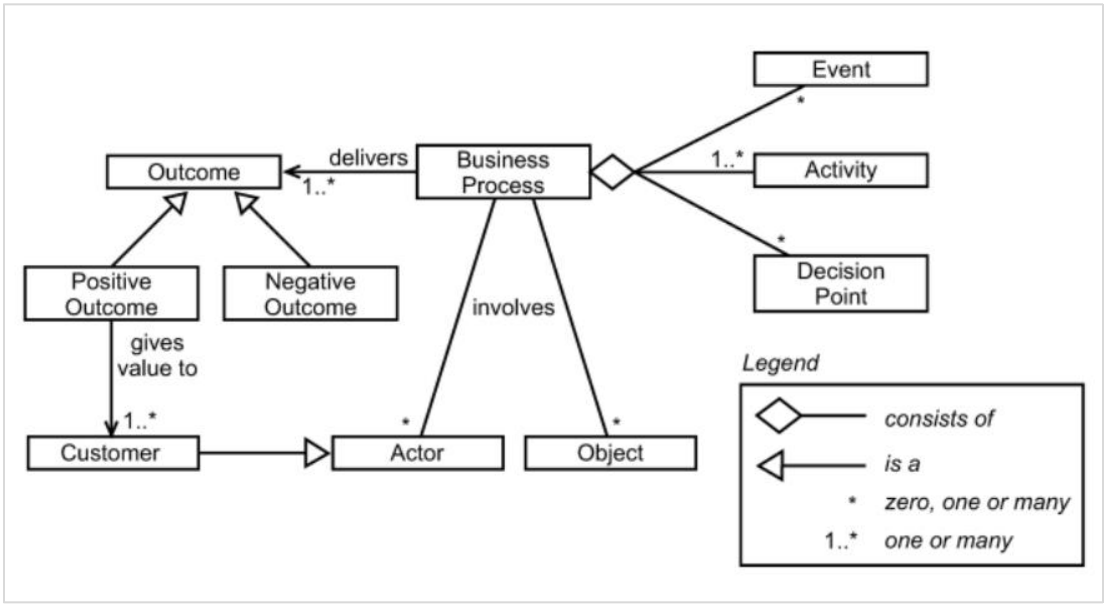


Figure 1: The ingredients of a business process [1]

2.2 The rise of Business Process Management

According to the European Association of BPM, Business process Management (BPM) is defined as a systematic approach aiming at capturing, documenting, executing, measuring, and monitoring automated and non-automated business processes within an organization, in order to satisfy a business's objective. It is a body of methods techniques and tools, dedicated to analyzing, designing, implementing, and continuously improving end-to-end or value-chain organizational processes [1]. By leveraging BPM, a business can enhance operational efficiency as well as customer satisfaction.

BPM has its roots in the birth of scientific management when Friedrich W. Taylor highlighted the importance of standardized processes and work procedures. This is when the role of managers was introduced, and laborers specialized and focused on a specific part of the production process. Before the BPM definition, many different management philosophies emerged and played a significant role in the process-thinking introduction. According to Hammer, BPM genesis was mostly characterized by two paths of development: process improvement and process development [4].

Earlier research in the field was centered around the improvement of established business processes. One of the first process improvement initiatives as a management practice was with Total Quality Management (TQM). According to the TQM philosophy, the cost of poor quality is higher in the long run than the cost of putting in place processes to produce high-quality products and services in the first place [5]. In addition to quality-focused processes, the Six Sigma strategy was developed by Motorola during the 1980s implementing TQM principles as well as a more analytical approach. The purpose of Six Sigma methodology is used in improving the predictable quality of developed products and services through the removal of normally distributed errors [6]. In business process improvement, the Six Sigma application can be used in reducing and eliminating defects in a business process [7].

At the start of the 1990's Business Process Redesign (BPR) emerged aiming at radically transforming business processes. Business Process Reengineering was aiming at initiating a comprehensive transformation, not limited to the business process itself, but also extending to all its associated elements, such as the organizational structure [8]. Businesses were challenged to rethink their functions in a process-centered, customer-focused way, taking advantage of information technology innovation [9]. The concept where an organization reshapes completely the way it functions was famous during the 1990s and later abandoned due to many factors such as rigid infrastructure issues which impeded radical changes as most businesses needed a more incremental approach. In addition, BPR lacked the aspect of quality improvement and did not have as disciplined an approach to metrics [10].

During the latter part of the 1990s, a more moderate approach that focused on continuous process improvement rather than process reconstruction was proposed, aiming at improving interdepartmental and inter-functional performance [11]. Business process orientation (BPO) is a management approach, that focuses on integrating different areas of expertise in process creation and delivering value to customers [12]. Enhancing BPO in an organization entails the introduction of process jobs, where employees are organized in cross-functional process teams and take full responsibility of delivering services while obtaining a full process view [13]. Michael Hammer also introduced the term process enterprise, mentioning that a whole enterprise could gain efficiency when is coordinated around standardized core processes that exceed the division's boundaries. Companies shifted their focus on process performance, by changing their measurement systems from unit to process goals [14].

The emergence of IT systems such as ERPs (Enterprise Resource Planning) and WfMSs (Workflow Management Systems) supported the existence of process-oriented operations. Although WfMSs initially focused on the distribution of work between human actors, after the rise of Service-oriented Architecture (SOA), were able to connect with other systems and they progressively embedded mechanisms for monitoring and analyzing a process execution. In conclusion, different methodologies and standards have appeared over recent history, to quantify and improve a business process. The disconnect between the IT and Business Management point of view led to the idea of the BPM unified approach. BPM is not only focused on the organization and planning of the process itself but also provides a more holistic approach, combining process management and process automation. BPM entails the entire business process lifecycle.

2.3 Business Process Management Lifecycle

Each business process initiative should start with the use of BPM to identify which processes need improvement, redesign or step removal that may have caused the largest number of errors or the biggest cost waste. If no BPM initiative has been applied in the past, process identification can be initiated by a high-level description of the organization’s major processes and their current state [15].

Object Management Group (OMG) has defined a scale to classify the maturity of a business process, comprised of 5 different levels. According to the OMG’s maturity model, a process can be ranked from an initial state where it is performed in an ad-hoc way, to an optimizing state where it can be continuously improved (Figure 2). Business processes can also be identified, delimited, and prioritized based on attributes such as their strategic importance, cross-functionality, and customer impact. Process architecture is the output of the process identification phase and represents a collection of core business processes within the organization and their interrelation.

Having understood, designed, and delimited the as-is processes which are the outcomes of process discovery, the next step is to evaluate their performance. KPIs should be determined aiming at examining process efficiency and effectiveness. Values measured can be time-related (e.g., intervals between activities), cost-related (e.g., money spent per month), or quality related (e.g., rate of defective products produced). Identifying and assessing issues and opportunities for process improvement is part of the process analysis phase [1]. Via process analysis, weaknesses are identified, and the impact of every possible change is evaluated. In this step an organization can decide which processes need to be redesigned.

In the redesign step, the as-is process is transformed into a to-be version based on informed decisions considered, and the available optimization options derived from the previous step analysis. There might be multiple redesign options that need to be analyzed and evaluated. This is why business process analysis and redesign are interdependent. Business process simulation can be used for model-based and data-driven analysis where different paths are evaluated, and potential bottlenecks are identified whereas suitable metrics are considered.

At the end of the redesign phase, the changes are about to be implemented and the model transforms into a running system [16]. In the process implementation step, organizational change management plays a significant role in synchronizing activities such as change planning, change introduction, and training of the participants to support the new processes. During the new process running, in the monitoring and control phase, process performance is evaluated. Execution data are constantly collected and assessed for conformance to objectives and performance goals. A new BPM lifecycle may be triggered by changed corporate goals and the business environment.

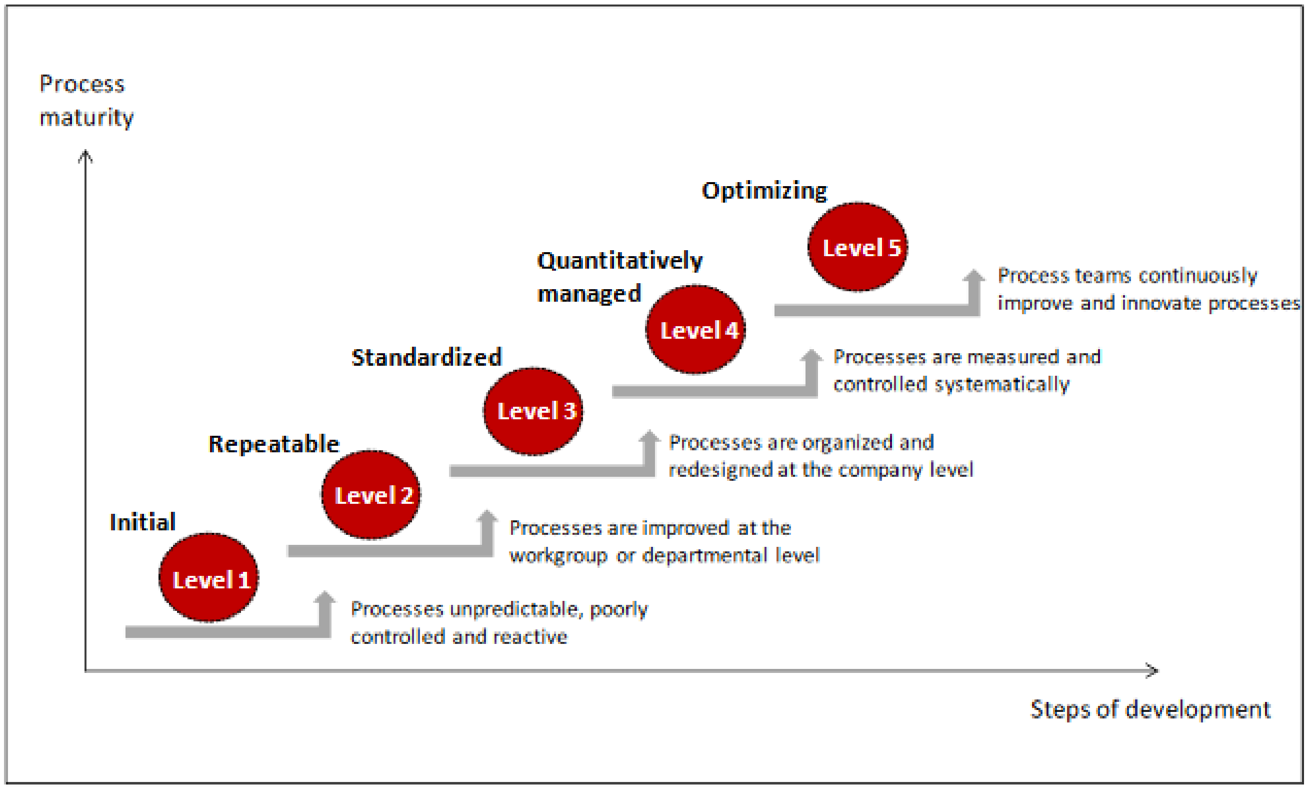

Process maturity
Optimizing
Level 5
Process teams continuously improve and innovate processes
Quantitatively managed
Level 4
Processes are measured and controlled systematically
Standardized
Level 3
Processes are organized and redesigned at the company level
Repeatable
Level 2
Processes are improved at the workgroup or departmental level
Initial
Level 1
Processes unpredictable, poorly controlled and reactive
Steps of development

Figure 2: Maturity model based on OMG [17].

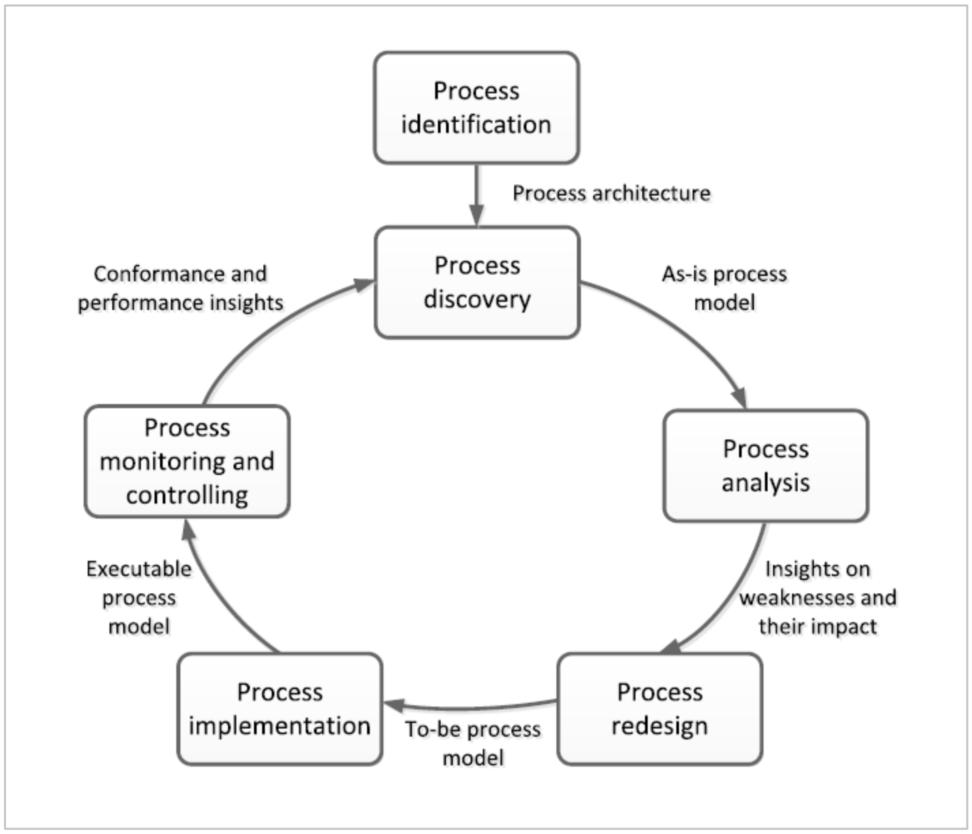


Figure 3: The BPM Lifecycle [1]

BPM lifecycle was refined in an empirical study (Figure 3) comprised of the following activities:

- The cycle is triggered when an existing process needs to be documented or improved or a new process needs to be introduced. Process discovery reveals the output generated by the process and its importance for the client.
- Items of process discovery are documented in the as-is or current state process model, and a systematic examination of its state can identify weak points.
- Process analysis reveals a weakness in the process and is the starting point of process design. Different process analysis approaches can be evaluated via process simulation.
- Process implementation involves both IT implementation and change management. The result of it is the to-be process.
- Process control involves the continuous monitoring of the process instances, which can lead to identification of new weaknesses that can trigger a new systematic process analysis.

The BPM lifecycle outlines a straightforward approach for achieving continuous improvement. Its implementation requires coordination of the responsible parties, the methodologies used and the supporting software tools. That is the responsibility of the process governance [18].

2.4 Process Driven Applications

Nowadays, due to global availability of knowledge, differentiation from competition through services and products is becoming less important. Internal processes on the other hand, create a sustained competitive advantage providing better quality services and products and lower costs. Furthermore, when products functionality is transparent through marketing, internal processes can be confidential and more difficult to copy, thus efficient process automation can provide strategic advantages [19]. Despite the growing relevance of data continuity and consistency, in the engineering area rigid monolithic software programs that are difficult to integrate are still prevalent.

Innovation can be provided via process driven applications (PDA) that offer business-oriented solutions and help identifying end-to-end business process workflows that include system and organizational constraints by reusing data and functionality from other applications and platforms [20]. More specifically, PDA is based on the BPMN 2.0 (Business Process Model Language and Notation) where the semantic addition permits to model and execute business processes but also to perform integration with different systems. As a result, PDA enables the orchestration of various processes as well as the integration of tools such as custom microservices via standardized interfaces such as REST API. In this way business processes can be more transparent and monitored [19].

Process Driven applications are user-centric and collaborative in nature, with a sense of engaging users in the processes. Ideal candidates for process-centric applications are processes inside a company that require optimization like cross-functional operations. PDA are also recommended in cases where processes are extended across many systems and changes are expected [19]. As discussed in the previous section, optimization can be achieved when a process is executable and as a result, constantly monitored [19].

Process-oriented applications are divided into the following layers.

- The (F-S) layer contains the user interfaces of the applications that are task oriented, and precisely customized in the role of the process participant. They can be implemented using web-services and communicate with the process engine via REST [21].
- The business processes are modeled inside the (PDA-L) layer.
- The service contract implementation layer (SCI-L) that focuses on service contract implementation is responsible for backend communication.

Communication can be stateless where the interaction is autonomous and no information is retained, or stateful where a component needs a response to proceed with the conversation. In the following example (Figure 4) a stateful case is shown where a message triggers a simulation microservice, and the process waits for the simulation outcome. Although stateful communication is optional, stateless addresses technical connectivity and data transfer via REST or SOAP services. To limit data transformations, it is recommended that a canonical data model is agreed [21]. As a result, data transformation is performed only during system integration. Data exchange with third party applications can be performed via REST API pre-defined JSON and XML structure.

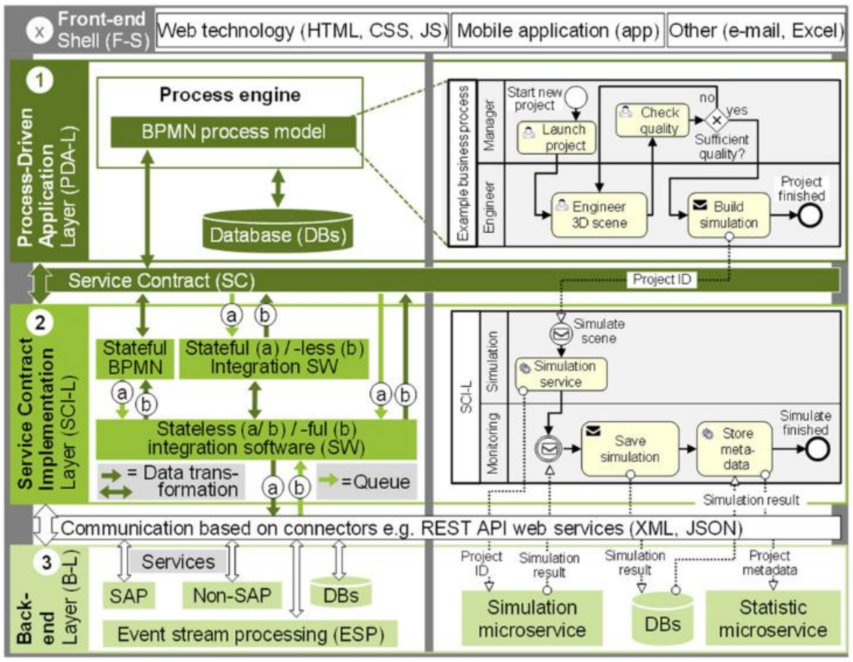


Figure 4: Process oriented application composition [20]

A process-driven application can be implemented in collaboration with the following roles:

- A business expert designs the process using a BPMN model, without implementing technical information.
- In collaboration with the business expert, process analyst can modify the existing model implementing information about the data structure and defining the service calls for data transferring inside the current system.
- Once the services and interfaces get associated with the BPMN activities within the business model it can be executable. Having consulted the previous roles a developer can make the final modifications in the model which now can be executed via a process engine.

Within the process engine human workflow management as well as service orchestration is being achieved [18]. More specifically, a process engine handles user task assignment to process participants via human workflow management whereas, communication with internal and external IT systems is part of service orchestration (Figure 5). In summary, the design and implementation of process-driven applications prioritize adaptability and self-sufficiency, resulting in traits like agility and scalability within complex IT environments. The implementation of a process driven application entails a step-by-step collaboration, that ensures alignment with the business requirements as well as the technical capabilities.

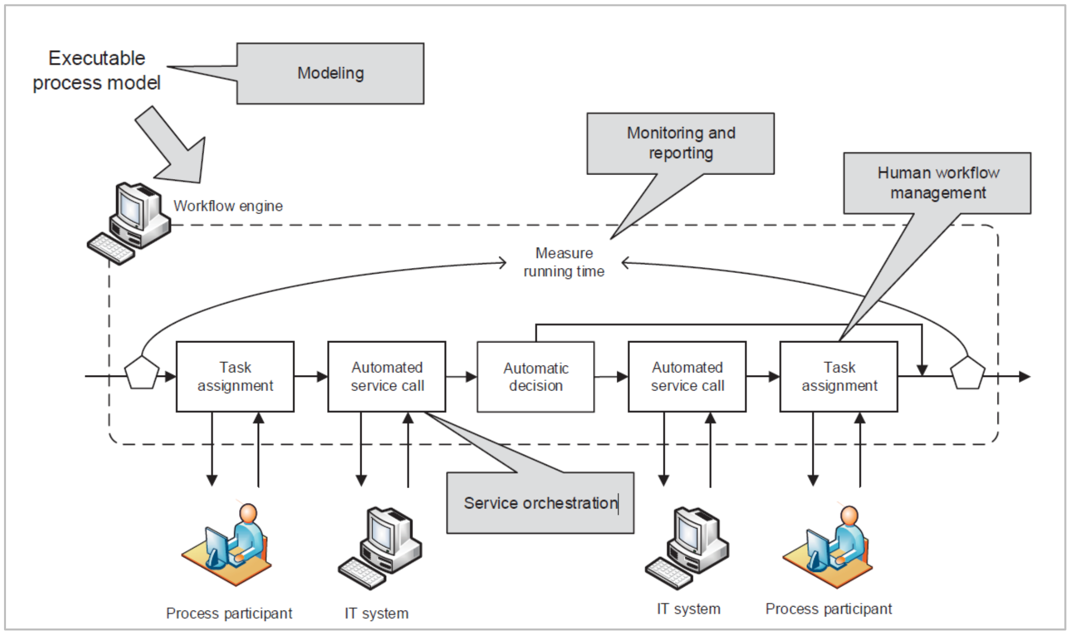


Figure 5: Process automation with a workflow engine [18]

2.5 Chapter Summary

Organizations in today's competitive business landscape face problems such as global competition, versatile client needs and shorter product lifecycles. For staying efficient and responsive, businesses adopt a process centric approach, focused on Business Process Management (BPM). BPM entails managing a series of events, decisions and activities that add value to the company and its customers.

BPM has historically progressed from Total Quality Management and Six Sigma Techniques to BPR and continuous improvement strategy. BPM gained in popularity with the development of IT systems such as ERPs and Workflow Management Systems, giving rise to Process Driven Applications. PDAs are collaborative user-centric applications that employ BPMN 2.0 to orchestrate processes. These applications enable organizations to be agile, transparent, and competitive by enabling continuously monitored and optimized business processes.

Collaboration between developers, business analysts and business experts are required for process driven application deployment. A process engine is also critical when it comes to human workflow as well as service orchestration. All in all, because BPM is iterative, organizations can constantly analyze, optimize, and adjust processes to align with business requirements and technology capabilities.

In conclusion, the transition from traditional procedures to more sophisticated techniques including BPMN and Process Driven Applications, helped firms in adapting to today's complex business landscape. Via the combined effort of different experts and with the use of process engines, BPM helped companies to provide additional value by optimizing their operations.

Chapter 3 : Business Process Modeling with BPMN

3.1 Introduction to Business Process Modeling Notation (BPMN)

The need for a common modeling language for business processes representation that could be sufficiently expressive and formal and easily understandable by users emerged over recent years. Converting a business process diagram into a machine-readable standard language is important for both sharing and execution purposes. BPMN (Business Process Model and Notation) is the primary standard for modelling workflows and business processes. On the one hand, a BPMN diagram can be editable in different domains and with the use of different tools. On the other hand, business process execution in distributed environments with the use of web services is widely used in many fields such as supply chain management and e-commerce [22]. For all the above reasons, BPMN is the primary standard for modelling workflows and business processes.

BPMN which is an open standard for business process modeling, was first introduced in 2004, aiming at reducing the fragmentation between the existing process modeling tools and notations [23], by offering graphical notation for representing a process in a form of a business process diagram [24]. It was initially developed by Business Process Management Initiative (BPMI) and later by Object Management Group (OMG) which is responsible for the BPMN standard maintenance [25]. The first BPMN versions until 2.0 lacked both well-defined semantics as well as a native serialization format. BPMN 2.0 which was introduced in 2011 had impeded a native XML serialization for BPMN diagrams that made it independent for languages like WS-BPEL and XPDL which are also XML based languages and were used in older BPMN versions (Figure 6).

By establishing a strategic BPMN model, all stakeholders can gain knowledge of the process logic. By using BPMN elements, process analysts can precisely record the business logic, while technical specifications required for process automation are hidden in background as XML code. As a result, BPMN process models are easily understandable as any technical information required for its execution is not exposed to the business users.

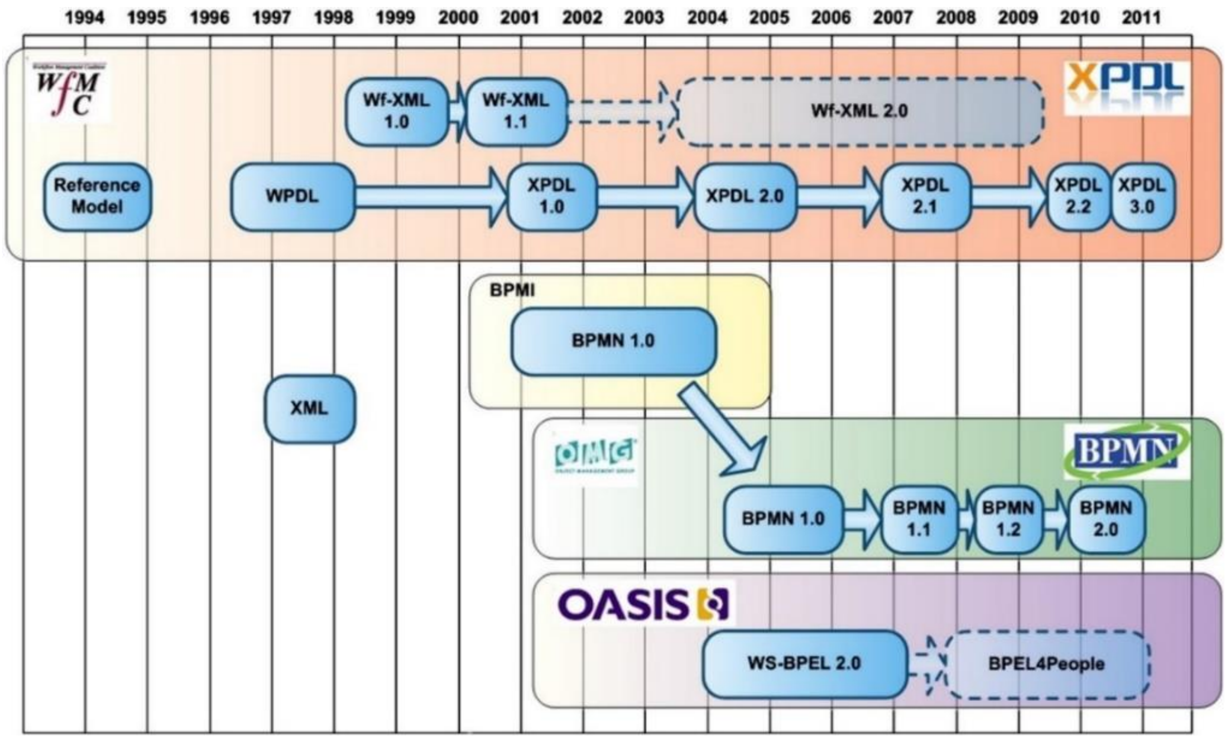


Figure 6: Business Process Standards timeline [24]

3.2 BPMN Elements

The elements of BPMN notation are organized into key categories that highlight different aspects of business activities, ranging from fundamental to more complex modeling. The main graphical elements of BPMN are the flow objects, connecting objects, swim lanes, data objects and artifacts (Figure 7). Flow objects are the major components of BPMN. They consist of activities or tasks, events, and gateways. Flow objects, including the sequence flows, are the main functional elements of BPMN. Tasks happen during a process and represent a unit of work inside the business process, without it being distinguished between human or automated work. Moreover, gateways indicate in what condition these tasks may happen, whereas events represent crucial incidents that may occur during the process.

Swim lanes, or pools and lanes associate a collection of tasks with a specific participant [24]. Sequence connecting flows define the connection and sequence between the flow objects within a pool, when message connecting flows refer to cross pool boundary connections. Artifacts that are linked with the tasks via the connecting object of association provide additional cosmetic information to the diagrams and do not affect the process flow. Lastly, data objects can also be employed as artifacts to indicate communication exchange between participants, as well as input and output parameters. In conclusion, flow objects as well as sequence flows that connect them, constitute the backbone of a well-structured BPMN diagram, whereas artifacts and data objects pools and lanes, provide additional implementation information without influencing the process flow. An overview of the flow objects is carried out in the following sections.

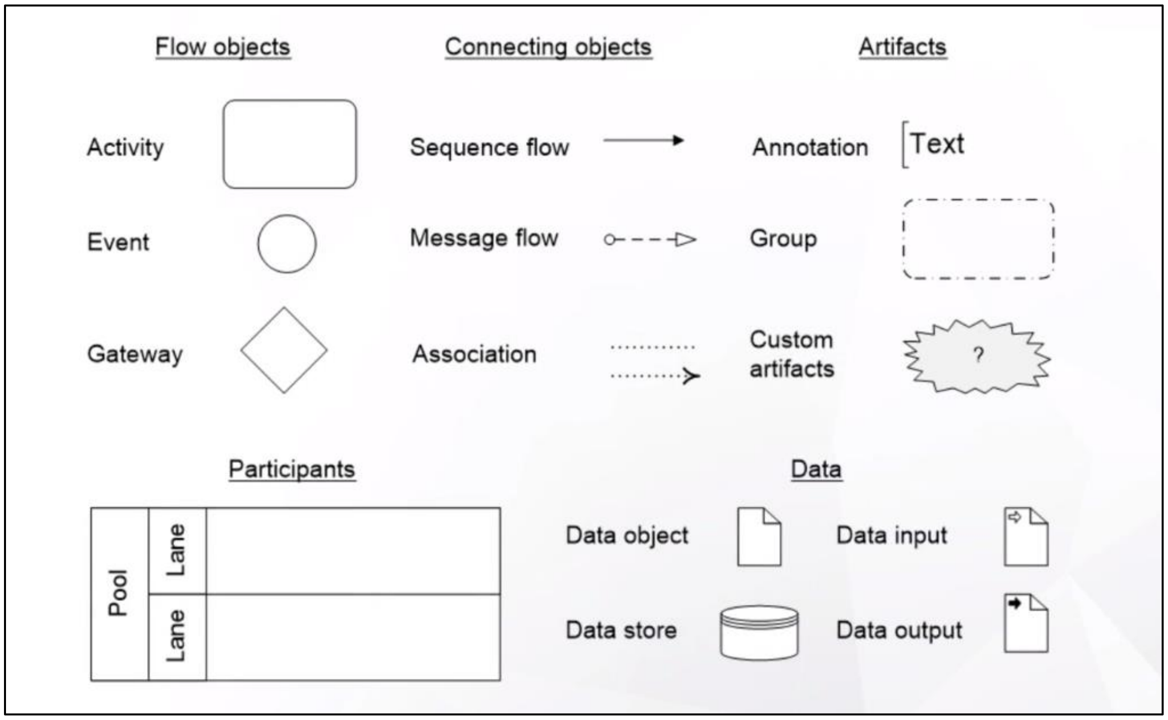


Figure 7: BPMN basic elements [26]

3.2.1 BPMN activities

BPMN activities are the heart of BPMN notation, since they represent an action that must occur [1]. Business modelers can use a range of tasks, each of those having their own label, describing the activity that must be completed. BPMN tasks enable both human and automated intervention, and they are represented as rectangles, with various symbols annotated in the upper left corner. By being labeled with task markers, activities can also entail additional business logic. Loop symbols, parallel and sequential multi-instance markers are task markers holding the cardinality concept in different ways (Figure 8): A Loop symbol indicates that an activity is repeated for a certain number of times. A parallel instance marker on the other hand, symbolizes that an activity is being done in parallel several times whereas, a sequential instance marker is depicting a repeating activity with a sequence (Figure 8).

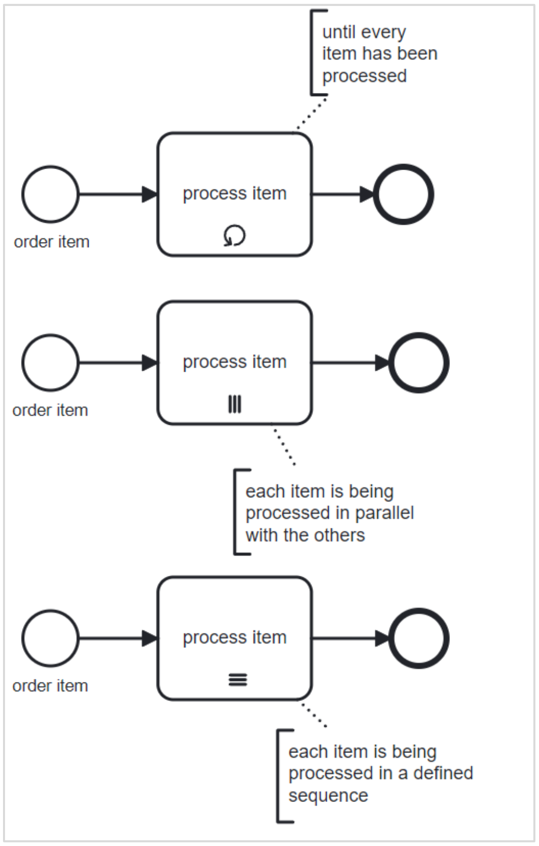

until every item has been processed
process item
order item
process item
order item
each item is being processed in parallel with the others
process item
order item
each item is being processed in a defined sequence

Figure 8:Task marker representation examples

The below process retrieved from National Registry of Administrative Public Services-Mitos [27] is displaying the process of a student enrollment in High school where the below task labels are identified:

- Service task: These activities are done via the assistance of a particular software or webservice entailing the execution of some code. The first step of the process below, which is the submission for enrollment, is carried out via an online application. The second step of the process concerns the spatial distribution of students and is conducted by a specific software belonging to the Ministry of education.

- Manual task: This type of activity is carried out by human actors without the use of an IT system. In the following process, the relevant supporting documents are submitted personally by the student's guardians, accomplished through physical contact at school.
- User task: Is the activity carried out by a human actor, with the assistance of an IT system. In the below example the principal of its school is obliged to verify the correctness of the submitted documents, compared to the digital documents retrieved from a digital archive.

The data objects represent the required documents that must be submitted.

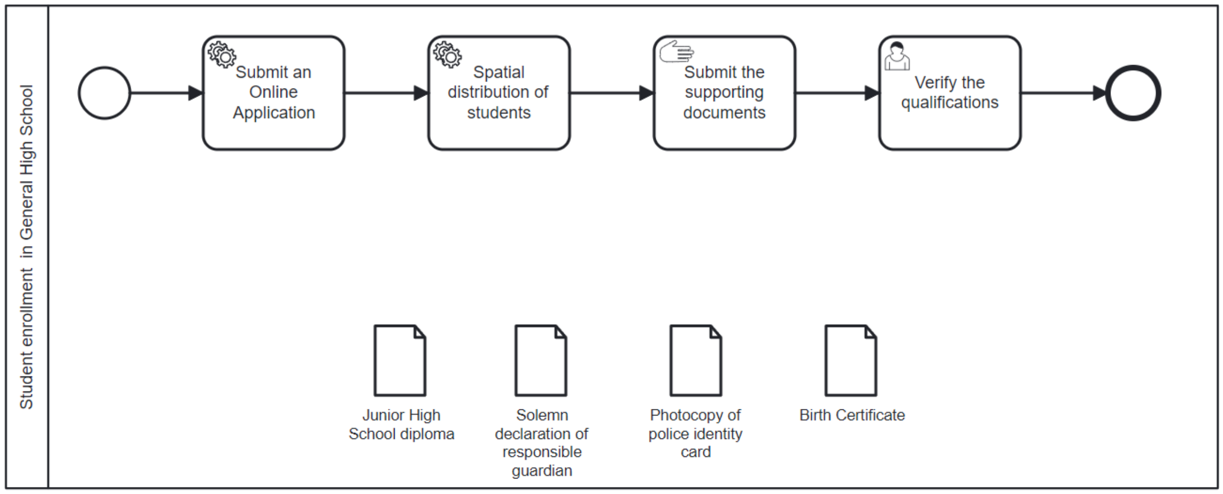

Student enrollment in General High School
Submit an Online Application
Spatial distribution of students
Submit the supporting documents
Verify the qualifications
Junior High School diploma
Solemn declaration of responsible guardian
Photocopy of police identity card
Birth Certificate

Figure 9: Example 1- Enrollment in Highschool BPMN diagram retrieved from mitos.gov.gr

More BPMN task subtypes are indicated in the following examples:

- Business rule task: This activity subtype is additionally linked to the DMN (Decision model and Notation) which can be used to model a decision flow and evaluate rules. In the following example (Figure 10), the most appropriate transportation method is calculated based on a decision diagram. Distance for determining the best mode of transportation is considered in the Transportation decision diagram (Figure 11).

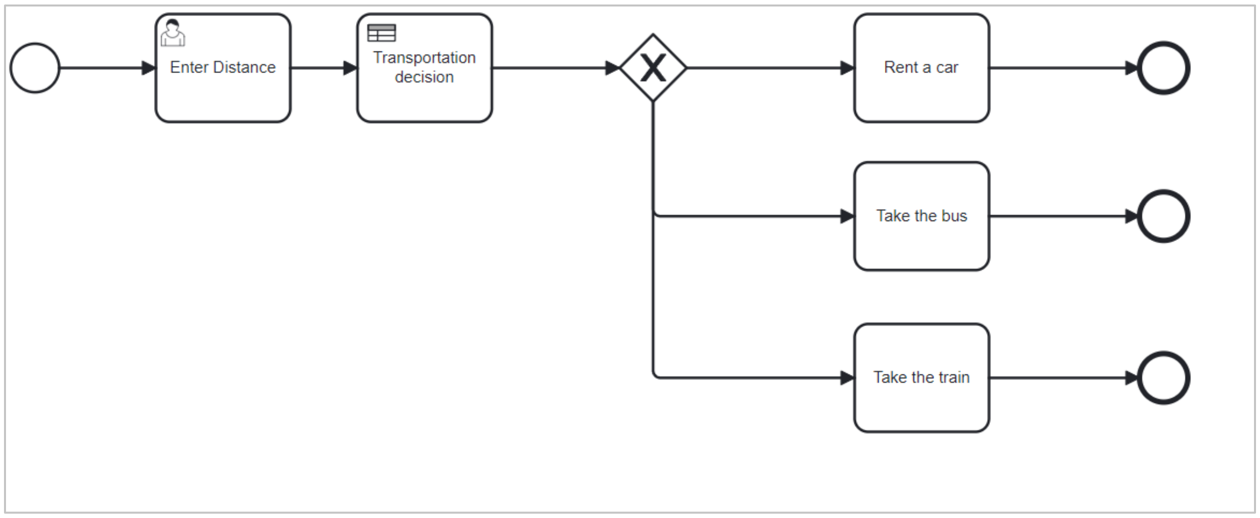
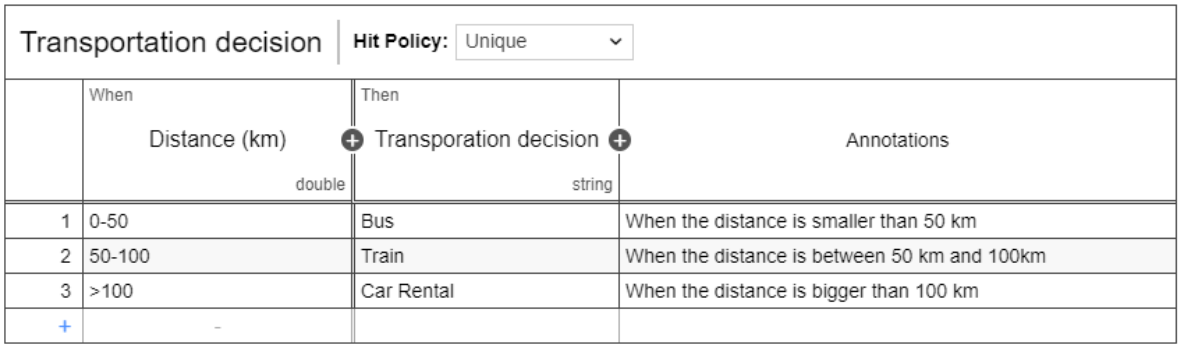


Figure 10: Travel method decision business process



Transportation decision | **Hit Policy:** Unique

	When Distance (km) double	Then Transporation decision string	Annotations
1	0-50	Bus	When the distance is smaller than 50 km
2	50-100	Train	When the distance is between 50 km and 100km
3	>100	Car Rental	When the distance is bigger than 100 km
+	-		

Figure 11: Transportation Decision, decision diagram

- Script task: This type of activity represents a task that may be automatically executed in a workflow engine. In the following example, an online order business process (Figure 12), a discount calculation is performed automatically with the use of a script.
- Send task: This type of activity symbolizes the receipt of a particular message notification, and it cannot be continued until the message is received. In online order business process, a message notification is sent by the user after order submission.
- Receive task: This activity task sends a message to a particular recipient. As seen below, a notification is sent by the e-shop to the user, confirming the order.

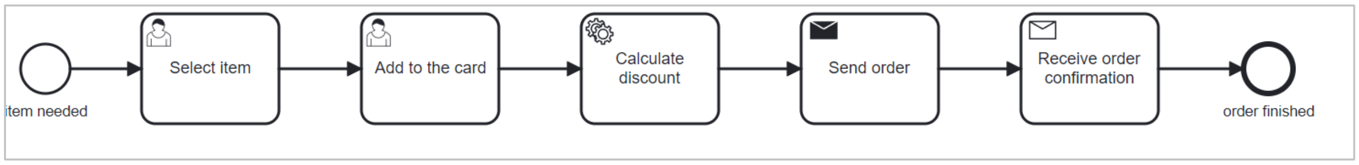


Figure 12: Business process of an online order

3.2.2 BPMN gateways

When a business process model is being executed, a new process instance is created. A process instance is interrelated with the concept of token. A token is a fictitious concept representing where we are inside the business process model at any given time during its execution [18]. The token 'lives' inside the business process until it gets consumed. However, depending on the business logic, a process instance may conclude differently, if the tokens generated follow a different path. In this regard, BPMN employs a unique element, namely the gateway element, that represents different routing points in a business process. Depending on the subtype of the gateways a cardinality of tokens is being generated.

Regardless of the gateway subtypes employed, gateway elements can be divided into two types: split gateways and join gateways. When a condition is met, a split gateway accepts a sequence flow and generates two or more outgoing sequence flows. As a result, when a token arrives at a split gateway, one or more sequence flows may be activated, resulting in the production of a token. A join gateway, on the other hand, receives two or more incoming flows and generates an outbound flow as the flow routes intersect (Figure 13).

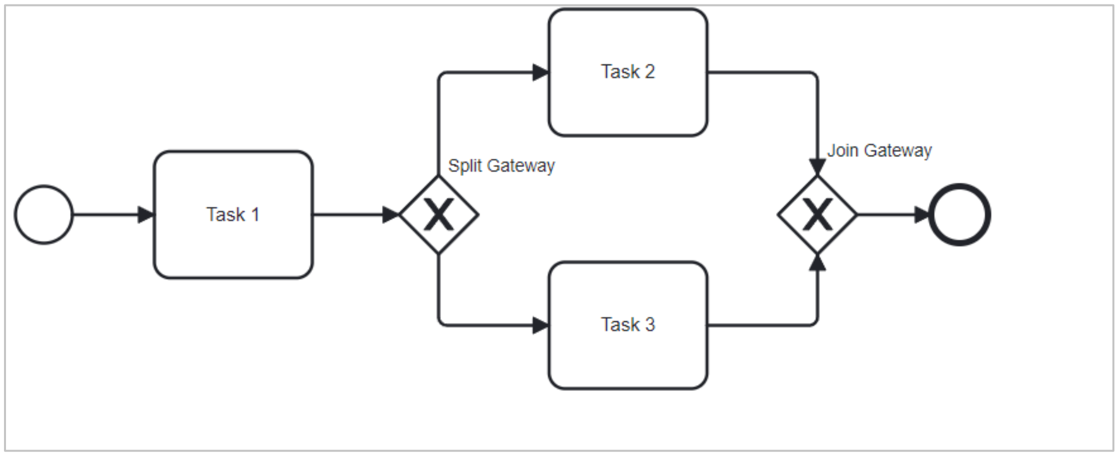


Figure 13: Split and Join XOR gateways.

Gateway subtypes:

- Data-based exclusive gateway, also known as XOR Gateway, is used when a condition needs to be evaluated so only one path can be chosen. In join XOR gateways no evaluation is being done and are used mostly for explicit modeling and readability purposes, ensuring that the token is going to follow just one path.
- Parallel gateways allow two or more different paths to be executed concurrently in a process instance. The split gateway generates a token for each sequence flow coming out of it. These tokens run in parallel, while the join gateway is used to synchronize all flows, by waiting for all the tokens to arrive, before moving forward with only one token. In the following example all the activities that need to be arranged are being executed concurrently, and the process is completed once all of them have arrived at the join parallel gateway.

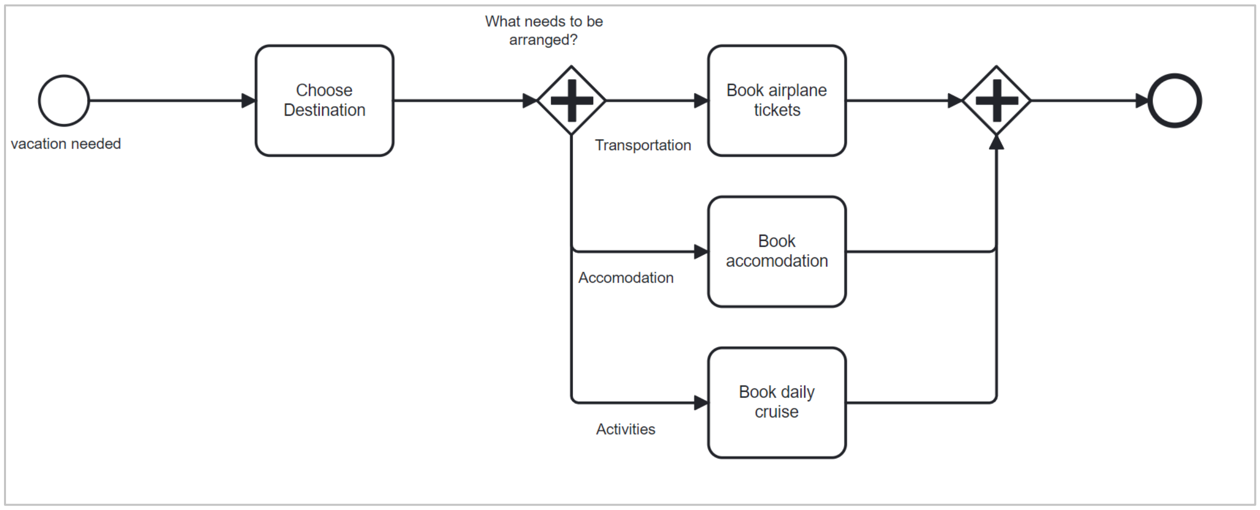


Figure 14: Parallel gateway example

- Data-based inclusive gateway also known as OR Gateway is a combination of XOR and parallel gateway. In other words, depending on the context of each scenario, it can act as a simple XOR gateway where only a path can be chosen or as parallel gateway where more paths can be followed, and more tokens are generated. In the following scenario, only one or more paths can be followed as the preferred activities at the beach can be more than one.

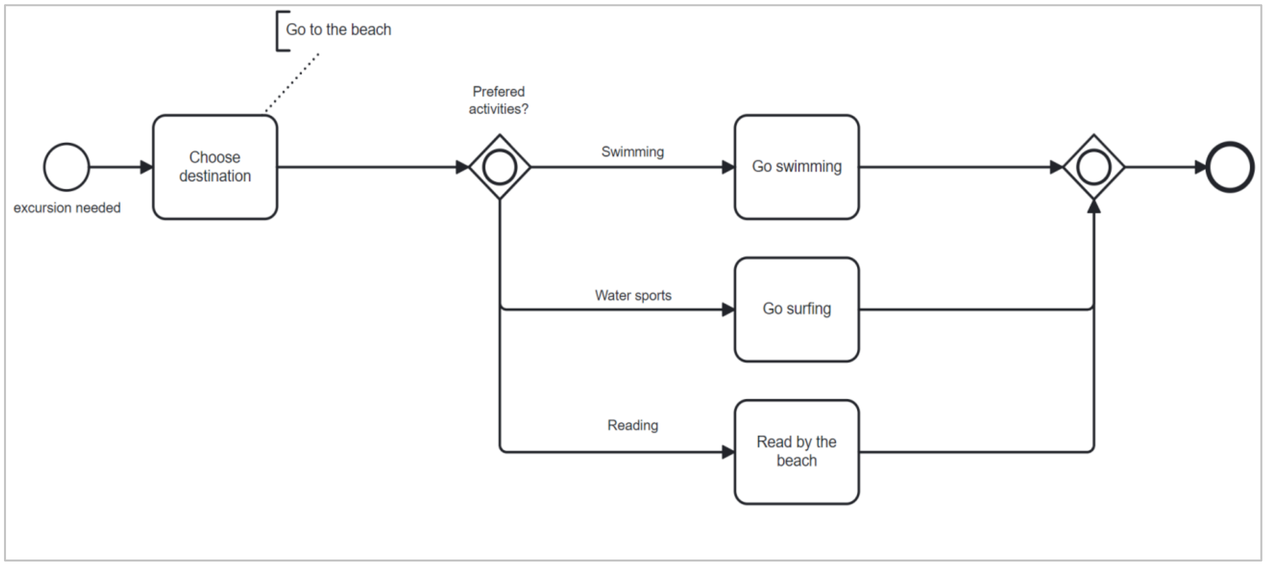


Figure 15: Inclusive gateway example

- Complex gateway is rarely used as a join node, indicating a complex behavior of synchronization. In the following process example, the token can only be consumed if all the criteria in accommodation booking are being met.
- Event-based gateway is mostly used as a split gateway. It leads to exclusive paths that indicate the outcome of possible events, that are not the result of process data evaluation. In the following process example where an item is ordered, the event-based gateway is displaying two different outgoing paths, resulting in a successful and an unsuccessful order.

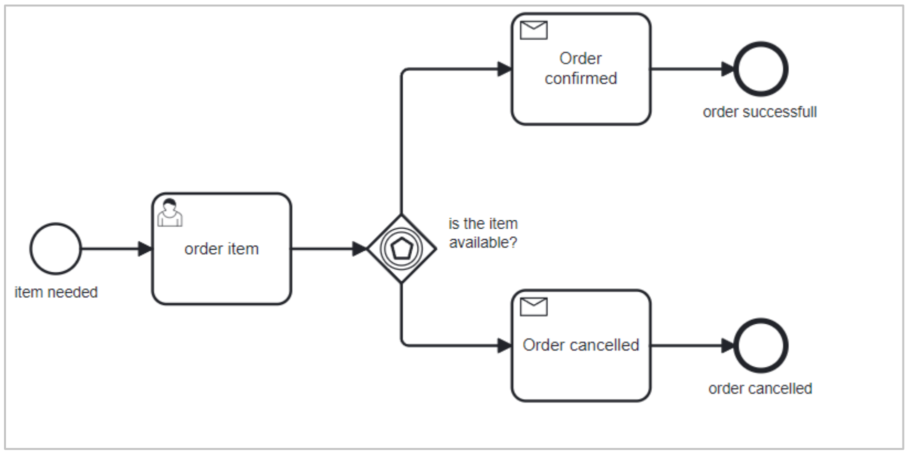


Figure 16: Event-based gateway example

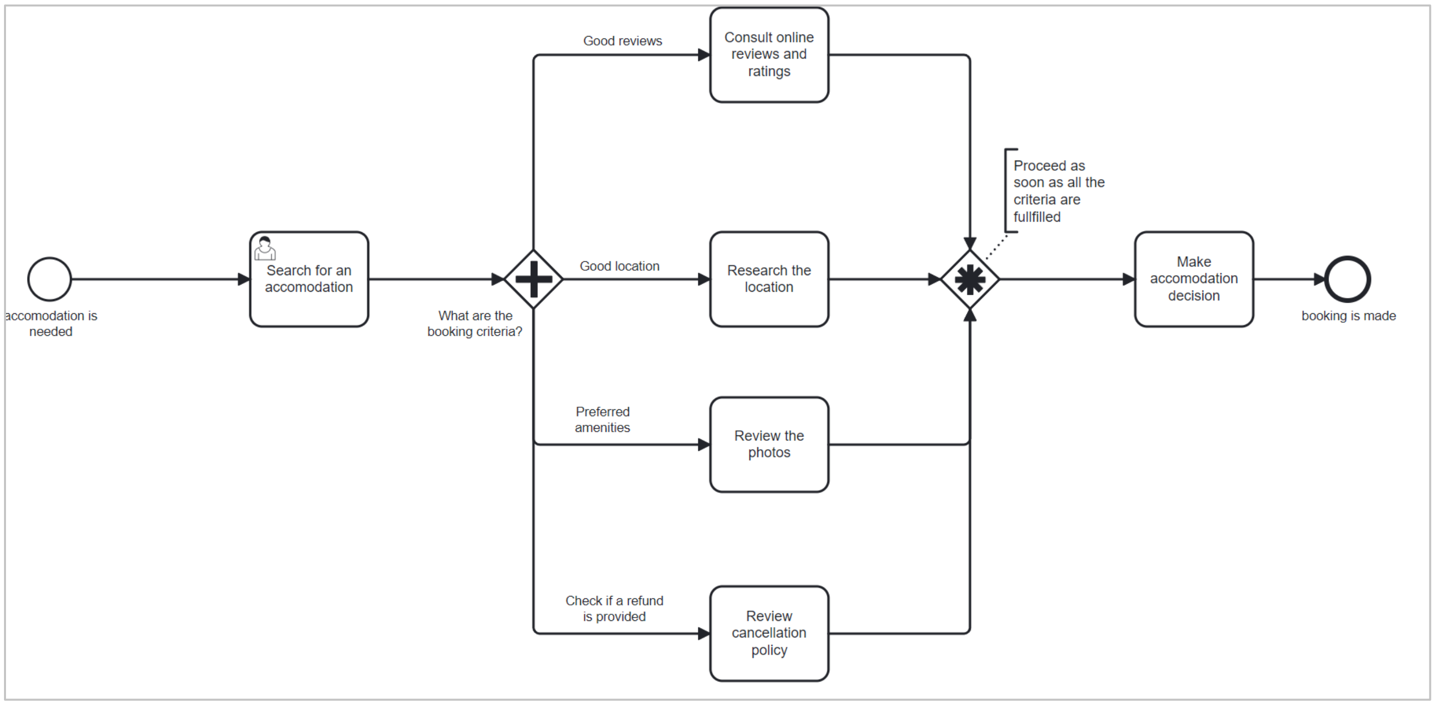

accomodation is needed
Search for an accomodation
What are the booking criteria?
Good reviews
Consult online reviews and ratings
Good location
Research the location
Preferred amenities
Review the photos
Check if a refund is provided
Review cancellation policy
Proceed as soon as all the criteria are fullfilled
Make accomodation decision
booking is made

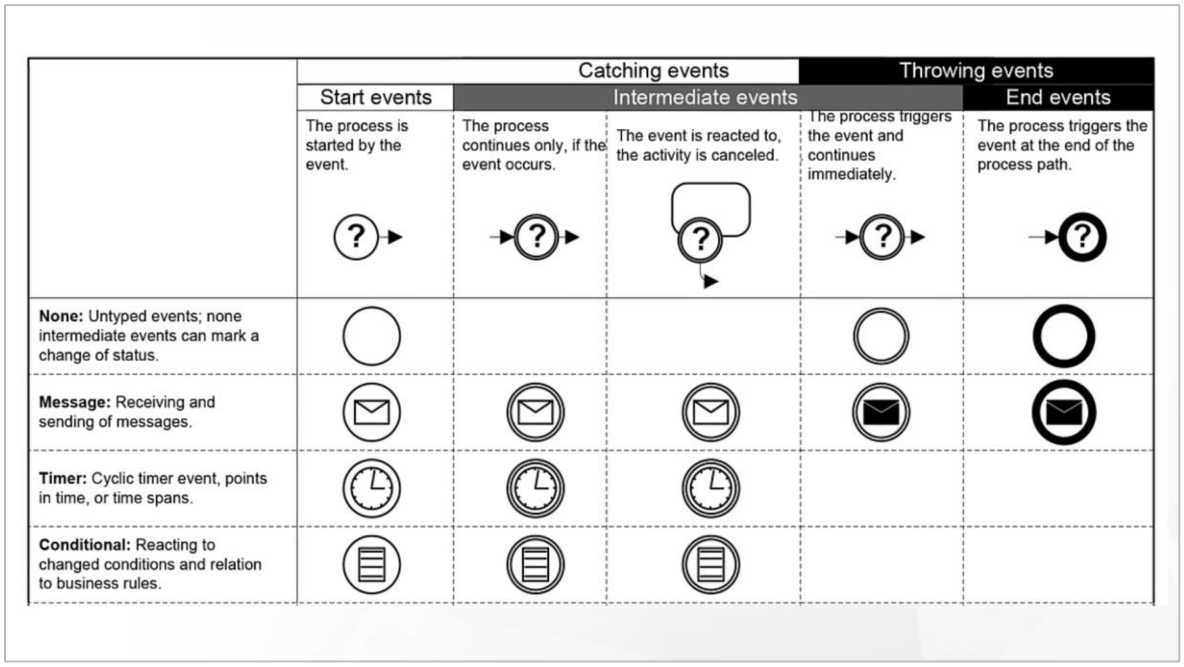
Figure 17: Complec gateway example

3.2.3 BPMN events

Another element of BPMN is events. It is considered a bridge between real word events and processes that react or trigger them. Hence, they play an important role in process modeling. They are classified into three essential groups based on the position on the process model, namely start, intermediate and end events.

- Start events identify the trigger and the beginning of a process execution.
- End events represent the completion of a process.
- Intermediate events represent the richness of a milestone within a process.

Events can be further categorized into catching and throwing events. A catching intermediate event is represented with a white envelope, stating that a particular message is required to be received. The token of the process instance cannot move forward until this message is received. On the other hand, a throwing intermediate event is represented with a bold and dark envelope and are initiated by the process rather than a wait for a response. Events can also have subtypes. Message, timer, and conditional events are the most common event subtypes (Figure 18).



The process is started by the event.

The process continues only, if the event occurs.

The event is reacted to, the activity is canceled.

The process triggers the event and continues immediately.

The process triggers the event at the end of the process path.

None: Untyped events; none intermediate events can mark a change of status.

Message: Receiving and sending of messages.

Timer: Cyclic timer event, points in time, or time spans.

Conditional: Reacting to changed conditions and relation to business rules.

Figure 18: BPMN events and subtypes [18]

Various events may occur, while a task is being performed prompting its suspension or exceptional execution [18]. Depending on whether an event interrupts the occurrence of the task, these events can be categorized into interrupting and non-interrupting boundary events. Furthermore, non-interrupting and interrupting boundary events, as well error event subtypes in BPMN introduce the concept of error handling in BPMN notation.

Throwing and catching intermediate events are most displayed between different pools of a BPMN diagram. As mentioned in the beginning of the chapter, its pool represents the greatest level of autonomy in a business process and sets the limits of actions for a specific participant. The exchange of messages is shown via message flows which are displayed as a dotted line.

In the example below (Figure 19), an airplane ticket purchase business process is being displayed. There are two pools depicting two different participants, the customer, and the airline. The customer initiates the process while the airline handles the payment verification, the seat allocation, and the flight cancellation. The customer selects the destination and the date of the flight which is depicted via a user task. Flights are being searched when a flight booking system API is being triggered which is displayed using a script task.

Then the customer chooses a flight based on the options retrieved. The next tasks include the entrance of the passenger information as well as the choice of the payment method, which is followed by the payment data insertion by the customer. Then the payment is script triggers the airline pool, where the airline is notified that a new order is being initiated. If the payment fails, then the process is terminated precociously with the use of a terminate event.

Otherwise, the airline verifies the payment, and the customer is being informed which is indicating with the use of an intermediate catching message event. Then the airline allocates the customer seat and sends the ticket to the customer which is shown with the use of a throwing and catching intermediate message event respectively. If the customer wants to cancel the flight, then the airline is being informed and cancellation management is initiated. When the flight is successfully cancelled, the customer is informed, and the process ends.

In the following example (Figure 20), the process starts with a message catching event indicating that an order has been received. The following BPMN element is an XOR gateway, evaluating the condition of the order items availability. If the items are available, then they are directly dispatched, and the process ends. However, when the other condition is being met, the item procurement task is being marked with two events. An error interrupting event as well as a non-interrupting escalating event, are both attached to the respective task.

The item procurement task indicates the existence of a subprocess. A subprocess defines a thorough sequence, although it occupies no place in the parent process rather than a task that is marked with a plus sign. The item procurement subprocess is displayed in a separate process inside the diagram. While the item procurement task is being executed, both events in the parent process are being triggered from the events occurring in the subprocess. More specifically, the interrupting event attached to the item procurement task terminates the subprocess that has been triggered from the item procurement task. Consequently, in the parent process the customer is being informed and the item is deleted from the catalog. Moreover, the escalation non-interrupting event in the parent process, is being triggered from subprocess when the delivery of a product is being delayed.

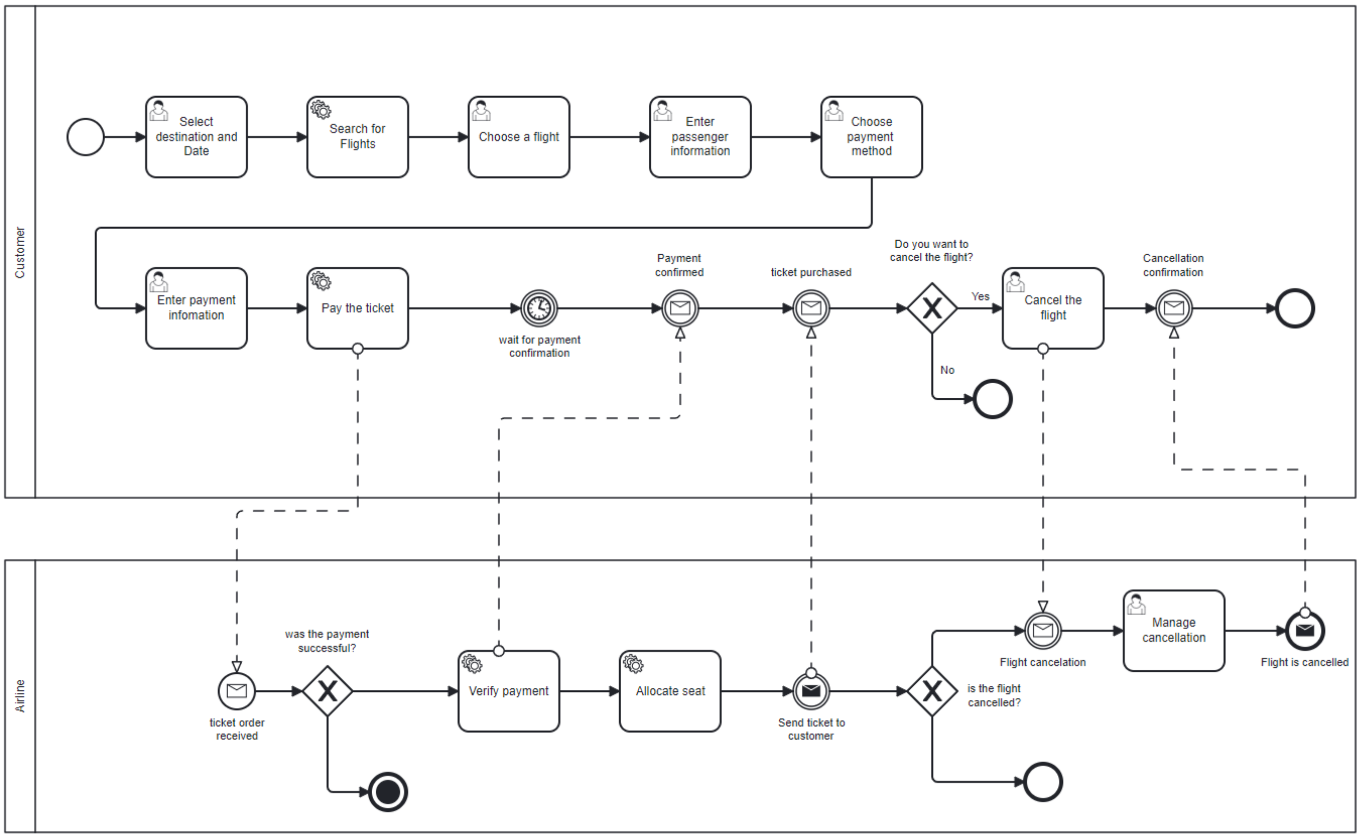


Figure 19: Example 2- Airplane Ticket purchase business process

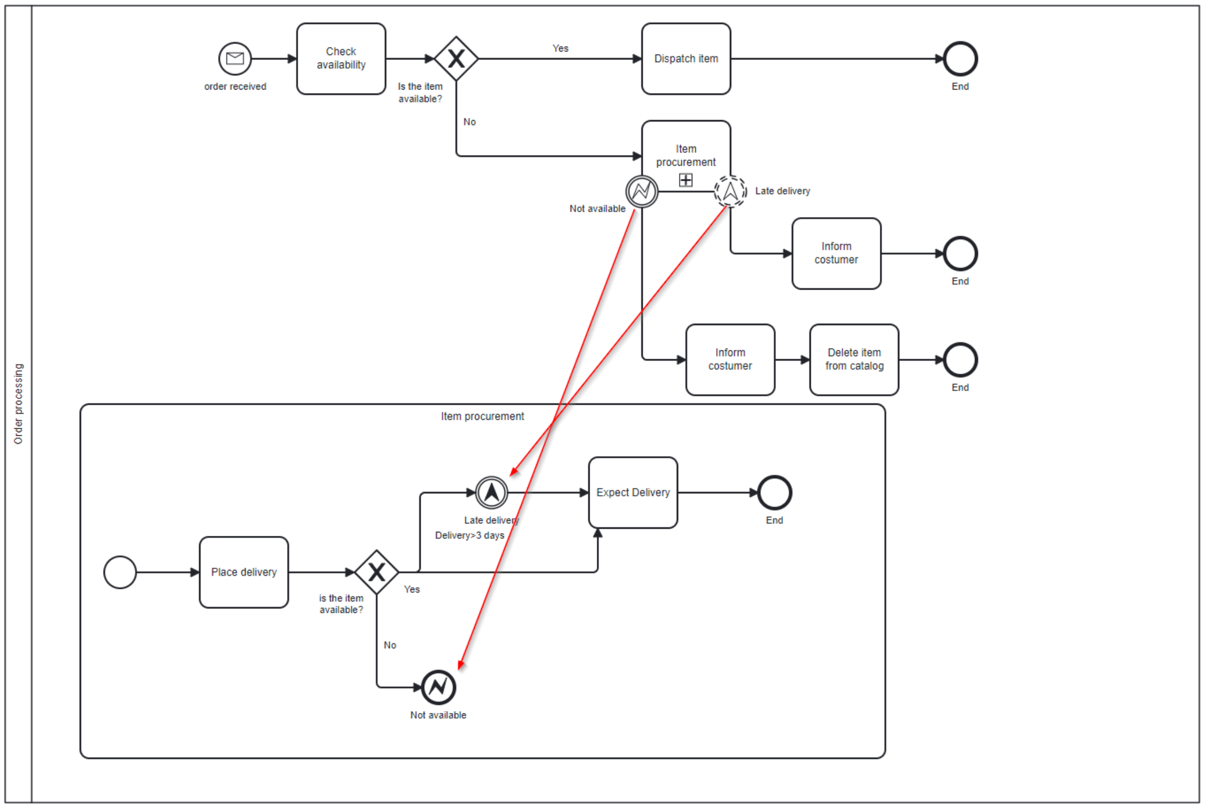


Figure 20: Example 3 -Order processing example

3.3 BPMN Collaboration Diagrams

A BPMN collaboration diagram indicates the visualization of the interaction between participants within a business process. More specifically, collaboration diagrams are composed of a collection of different participants represented as pools. They may represent a role or an entire business entity. Depending on the diagram context and the level of detail that is desired to be shown in the diagram, pools can be black-box or white-box that can also be separated into different lanes.

Pool interactions are represented as a message exchange modeled with message flows connecting two pools. The message flows are easily identifiable since they are depicted by dashed arrows. In addition, messages associated with the message flows can also be shown. Interaction between black box pools is shown with message flows that connect the boarders between different pools, whereas in white box processes the connection of the message flows is displayed between the corresponding process elements of the communicating pools.

The following collaboration diagram (Figure 21) is a high-level representation of a recruiting business process. In other words, the process is simplified as no cardinality of instances (many applicants and one job position) is displayed and the interview process has been omitted. Furthermore, different conductors that are not orchestrated by a process engine participate in the business process. Hence, there is no process instance, and the message must be transferred using a message flow.

To elucidate further, the process conditionally starts when a vacancy arises. This is when the Hiring department, which is represented as a separate pool, reports this vacancy to the Human Resources department, which is notified via an intermediate catching message event. Then the forenamed department is responsible for advertising the job position. Having notified the applicant who is also represented in a separate pool, apply an application to the human resources department. When the application is received, and checked by human resources, the vacancy is filled, and the hiring department is notified when the contract is signed.

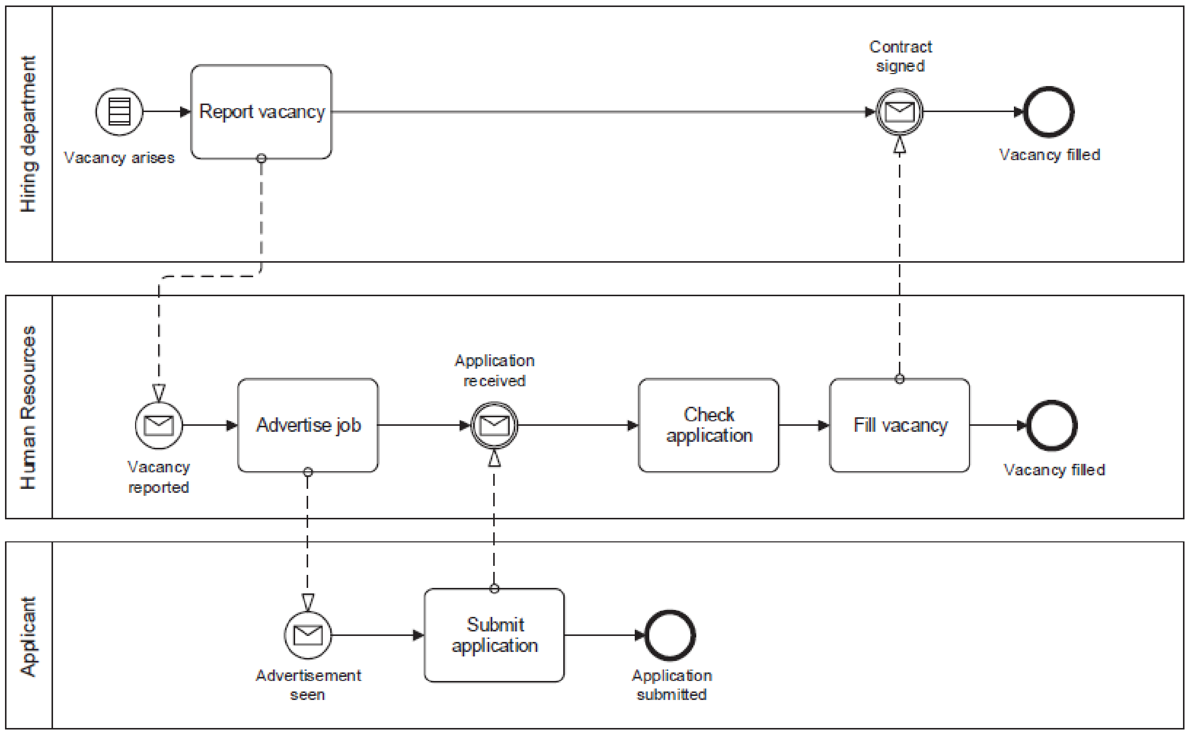


Figure 21: Recruitment process collaboration diagram [18]

3.4 BPMN Choreography Diagrams

BPMN's second version was expanded with the addition of choreography diagrams. Choreography diagrams show the interactions between two different participants of a business process rather than internal activities or activities performed by individual participants [28]. In comparison to collaboration diagrams which simply display that some form of communication occurs, chorography diagrams represent very precisely the time and logical order of the message flows [20]. They show which messages are received and sent and under what conditions.

The term choreography refers to the absence of a central agent controlling the actions in the business processes involved [29]. As a result, choreography diagrams can act as a contract between the different parties as they interact by receiving and sending messages. In this way, they abstract from each participant's own processes and individual work done by the participant, delivering a global perspective without any favoritism [30]. Choreography diagrams can be used to map out important points in collaborations, and lead to more comprehensive collaboration diagrams if necessary [18]. In comparison to the collaboration diagrams, they can bisect message flows between the communicating participants and as they provide more information for the message exchange (Figure 22).

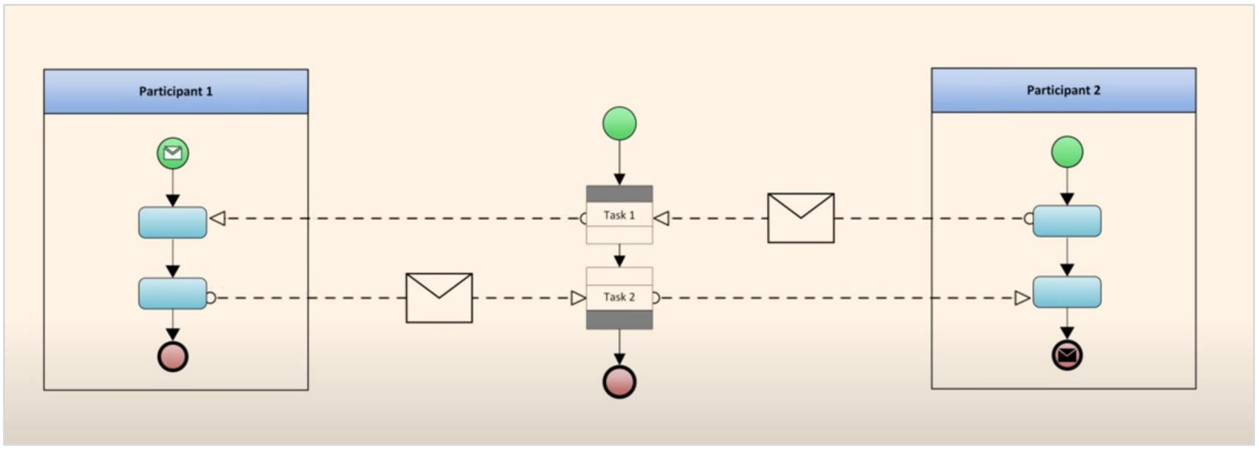


Figure 22: Comparison of collaboration and choreography diagrams [31]

A choreography process includes a series of activities as a sequence flow, that represent an interaction between two or more parties. A choreography task is the fundamental composing element of a choreography diagram. Every choreographic task represents a two-party message exchange. The initiator is the one that transmits the message. The second participant is the receiver who may optionally send a response to the initiator [30]. The participants are tied to the choreography task with a participant band, which is differentiated in color for the initiator and the receiver. Messages can also be connected to tasks using the envelope element [32] (Figure 23). In a choreography process, the sender of the second activity must be involved in the first activity, otherwise not every party can be aware of the message recipient.

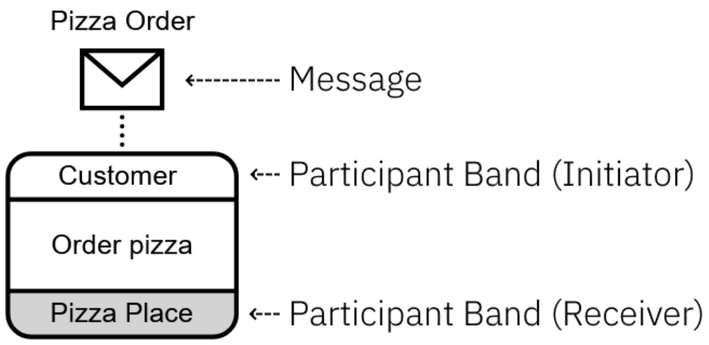


Figure 23: The definition of a choreographic task [32]

The benefit of choreography diagram is the illustration of a participant’s involvement in the process. As a result, it helps process analysts to get started. On the other hand, some steps that cannot involve communication between participants, but are important for the business process can be omitted.

The same recruiting process constructed with a collaboration diagram can also be displayed as a choreography diagram (Figure 24) including the following steps:

- Upon the vacancy arises, the first choreography task includes the Hiring department as the message initiator and the HR Department as the message receiver of the vacancy report. The vacancy notification is sent as a message via email to the recipient.
- Having received the vacancy report, the HR Department initiates the job advertisement process, which is represented as a collapsed subprocess. The possible ways of notification as displayed in the message of the choreography task, are the company’s site as well as other job finding websites. The recipients of the job advertisement are several applicants. The cardinality is represented with the parallel instance marker which symbolizes that the job advertisement is accessible in parallel to several applicants.
- Subsequently each Applicant follows the application’s submission process, which is also presented as a collapsed subprocess, and sends the application via an email message to the HR Department. The cardinality of application submission initiated by each applicant is also represented by parallel instance marker.
- Following this, the HR Department initiates the application checking process for the multiple submitted applications. The result of each application checking has two separate message recipients the HR Department as well as the applicant.
- After the applicant is selected which is displayed as an intermediate event, the next choreography task shows the vacancy filling process where the two communicating parties are the HR department the selected applicant.

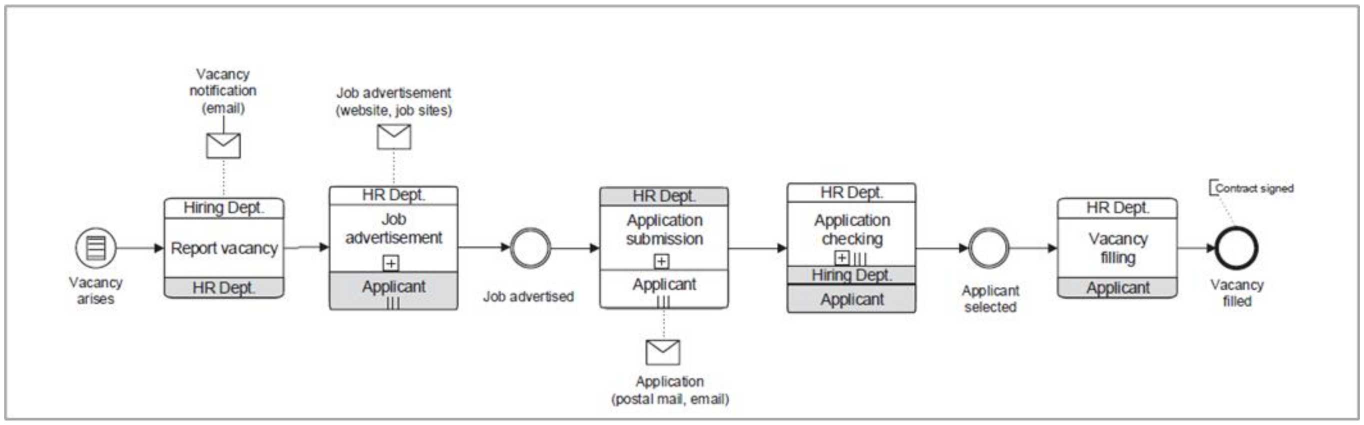


Figure 24: Recruitment process as a choreography diagram [18]

All in all, a choreography diagram displays the order of communication between various actors of a business process, and it is ideal for displaying conversation intense processes and can be a useful addition to operational level collaboration diagrams. In addition, a choreography diagram can integrate business process elements allowing a thorough examination of cardinalities. More specifically, the diagram displays that the job advertisement is viewed by multiple participants when a submission is also performed by different participants and the application checking is performed for all the submissions.

Furthermore, a choreography task can involve more than two participants as seen in the Application checking choreography task. Lastly, sequence of the sender's and receiver's participant bands in the same choreography task does not affect the process, given that the choreography task sequence is correct. This is the case of the Application submission choreography task, where the process is initiated by the applicants, but the conversation task is reversed showing the HR department which is the receiver participant at top of the choreography task.

3.5 Motivating Example - Airbnb booking process

The motivating example of the following approach is a booking accommodation process inspired by Airbnb. Airbnb is a platform that acts as an intermediary for hosts that can be property owners or renters and possible guests that seek short-term accommodations without owning any property. All activities of the booking process are managed by the Airbnb platform. Airbnb earns revenue by charging hosts a booking cost percentage and visitors a service fee (Figure 25).

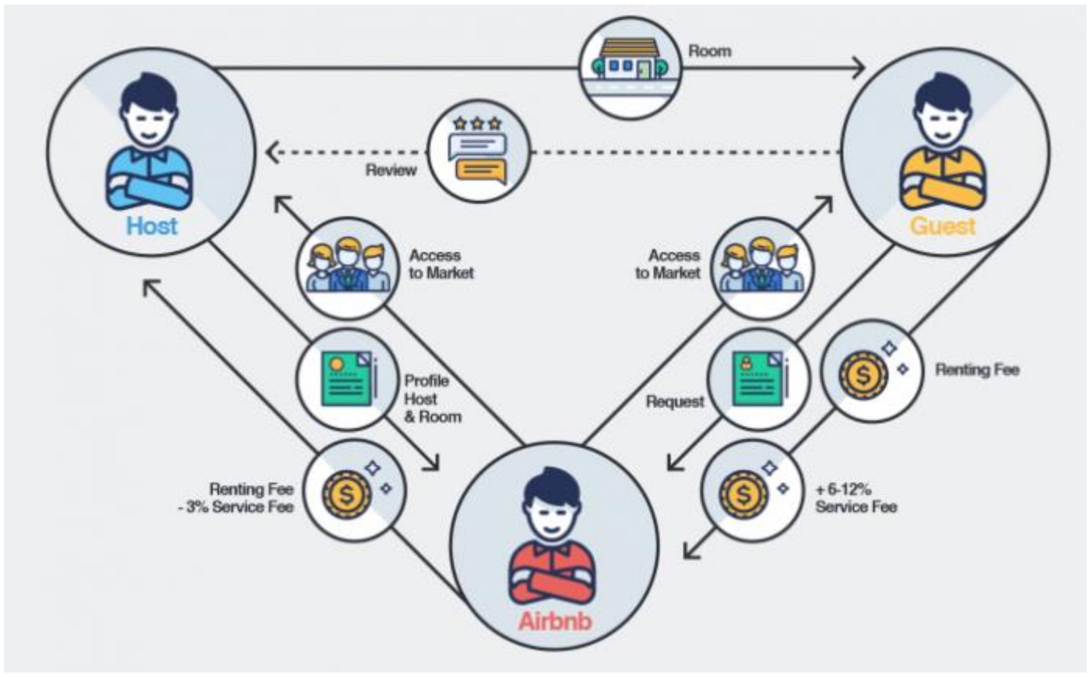
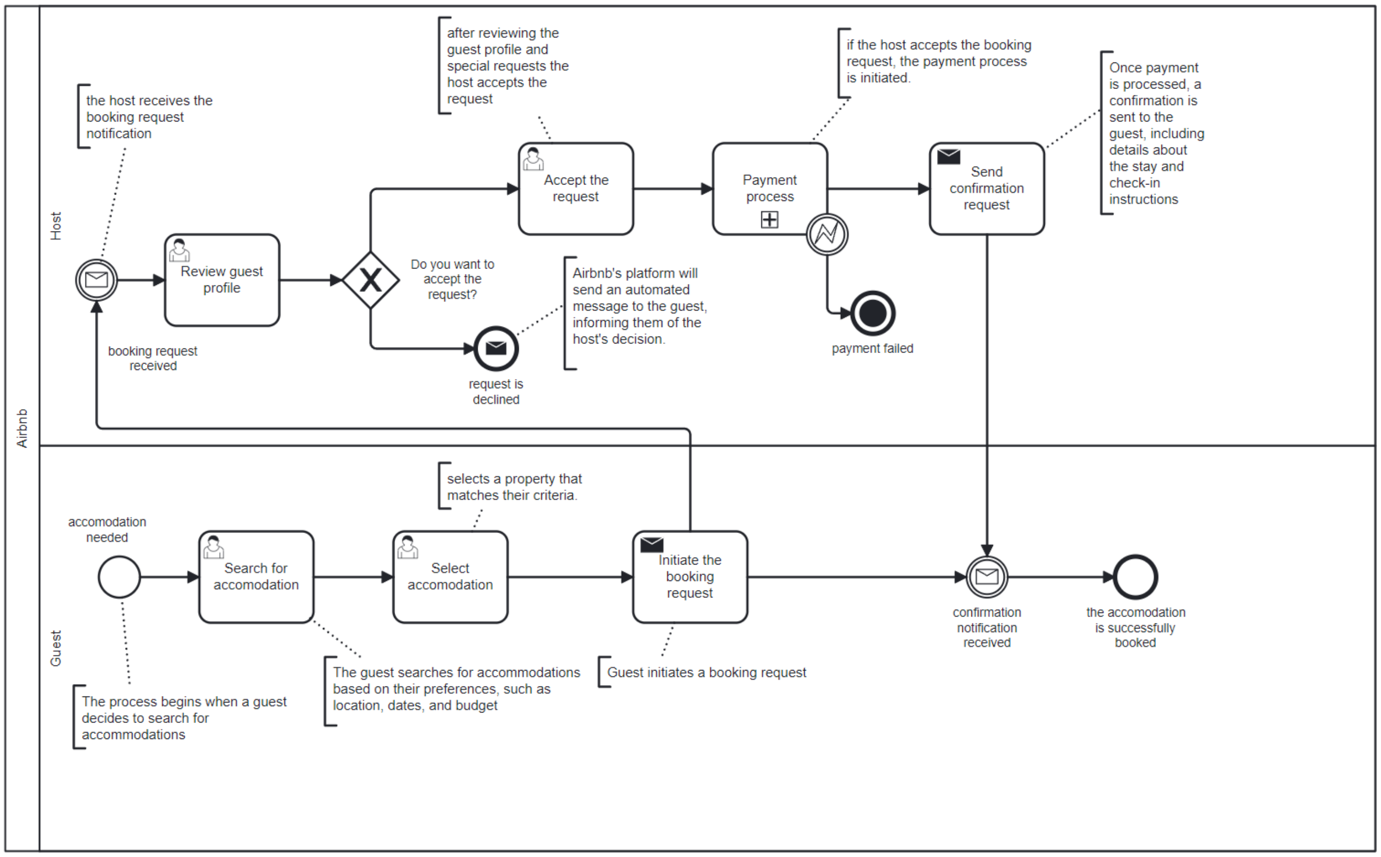


Figure 25: Airbnb simplified business model [33]

For describing the Airbnb booking process a BPMN diagram is used, containing a pool representing the Airbnb platform and two separate lanes that represent the platform users, the guest, and the host. More specifically the Airbnb booking process can be described in a BPMN diagram with the use of the following steps:

- When a guest searches for accommodation on Airbnb the booking process begins which is displayed as a start event.
- ‘Search accommodation’ is the guest activity where a place that meets their criteria, such as location, dates, number of guests and budget is being searched.

- As a result of the previous search activity, ‘Select accommodation’ is the activity where the guest makes the final selection of an accommodation.
- After the accommodation is selected the guest initiates the booking request, where more details such as special requests to the guest are specified.
- Following this, via the Airbnb platform the host receives the booking requests and decides whether the request should be accepted or declined.
- When the condition is being evaluated and the host accepts the booking request the payment process is being initiated. The payment process task is linked interrupting boundary event, and the process ends unexpectedly if the payment fails. If the booking request is declined by the host, a notification of declination is sent to the guest by the Airbnb platform and the process ends.
- When the payment process is completed successfully, a confirmation message is sent to the guest, and when the guest is notified, the process ends.



Airbnb
Host
Guest
the host receives the booking request notification
after reviewing the guest profile and special requests the host accepts the request
if the host accepts the booking request, the payment process is initiated.
Once payment is processed, a confirmation is sent to the guest, including details about the stay and check-in instructions
Accept the request
Payment process
Send confirmation request
Review guest profile
Do you want to accept the request?
Airbnb's platform will send an automated message to the guest, informing them of the host's decision.
booking request received
request is declined
payment failed
selects a property that matches their criteria.
accomodation needed
Search for accomodation
Select accomodation
Initiate the booking request
confirmation notification received
the accomodation is successfully booked
The process begins when a guest decides to search for accommodations
The guest searches for accommodations based on their preferences, such as location, dates, and budget
Guest initiates a booking request

Figure 26: Airbnb high level strategic booking process.

As a strategic process model the above example of the Airbnb booking process that contains pools and lanes serves its purpose of being easily comprehensible by stakeholders as the two participants are Airbnb users. However, the guests and hosts need to be orchestrated by a conductor that has control over both the users. For the above reason, Airbnb platform can act as an orchestrator for coordinating the reservation process. In summary, a collaboration diagram can be used where the reservation application is presented as a separate entity that acts as a virtual conductor, managing interactions and communication between guests and hosts.

3.5.1 Reservation app booking process collaboration diagram

In the below collaboration BPMN diagram a reservation application business process inspired by the Airbnb booking process is displayed. The three pools represent the roles and their activities within the application process. In this case, the reservation application acts as a coordinator between the two separate users. The business process collaboration diagram includes the following steps:

- The guest initiates the reservation process: The process starts when a guest user initiates a reservation request via a service task. This request is forwarded to the reservation application for further processing.
- The reservation app is checking the accommodation availability: The update reservation status task within the reservation application pool that acts as a coordinator, is triggered by the reservation request. This is where the availability of the requested accommodation reservation is being checked within the database. Two critical conditions are evaluated:
    - If the accommodation is not available in the database an error message email is sent to the guest and the reservation request is cancelled.
    - If accommodation is available, the process advances to a parallel gateway.
- The parallel gateway splits the flow into two separate branches indicating that both the host and the guest of the accommodation are notified simultaneously via a throwing message event. This coordination in the notification results in more efficient communication among participants.

- After the host is notified via email about the new reservation request, an exclusive gateway splits the flow into two separate branches. Subsequently, the host can choose between to alternative paths: Confirming or rejecting the request.
    - In case the request is confirmed by the host, the application notifies the guest and then the application places the reservation in the system.
    - In case the request is rejected by the host, the user is notified and consequently the application cancels the reservation request.
- Within the guest pool an event-based gateway shows the alternative paths of the process ending depending on the precedent events. Subsequently, based on the host's decision to confirm or reject the request the process may result in a reservation confirmation or rejection.

In conclusion, collaboration diagrams like the one shown below are useful tools for describing complicated interactions in a business process. They promote interaction clarity as they display the sequence of messages simplifying the comprehension of the activity flow. Moreover, they aid in the simplification of complicated processes that involve many participants and can improve interaction management efficiency. Finally, they can encourage business process transparency, ensuring that all participants can have a comprehensive picture of each process activity which can boost trust and accountability.

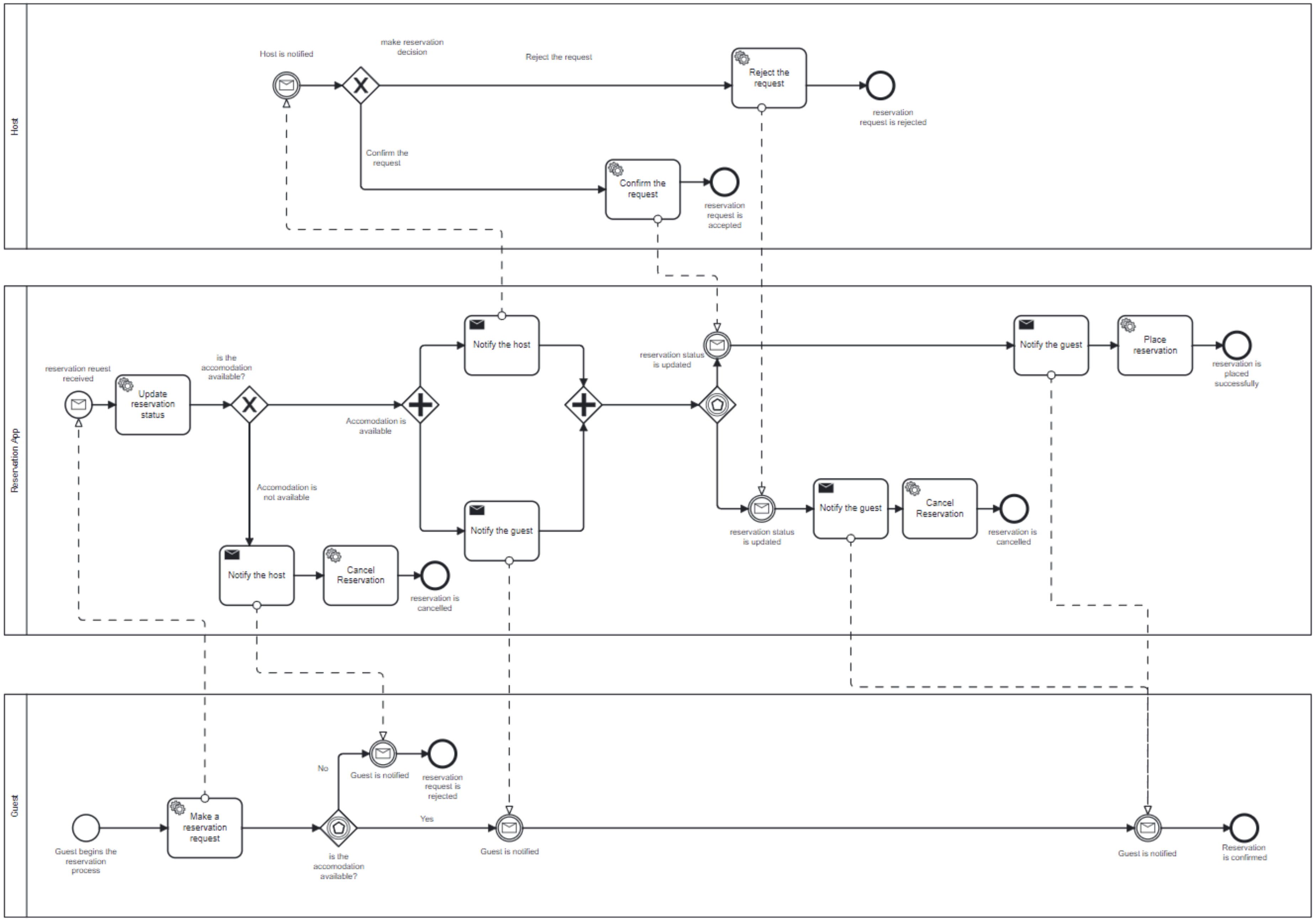

Host
Host is notified
make reservation decision
Reject the request
Reject the request
reservation request is rejected
Confirm the request
Confirm the request
reservation request is accepted
Reservation App
reservation reuest received
Update reservation status
is the accomodation available?
Accomodation is available
Accomodation is not available
Notify the host
Notify the guest
Notify the host
Cancel Reservation
reservation is cancelled
reservation status is updated
Notify the guest
Place reservation
reservation is placed successfully
reservation status is updated
Notify the guest
Cancel Reservation
reservation is cancelled
Guest
Guest begins the reservation process
Make a reservation request
is the accomodation available?
No
Guest is notified
reservation request is rejected
Yes
Guest is notified
Guest is notified
Reservation is confirmed

Figure 27: Collaboration diagram of a reservation app example

3.5.2 Reservation app booking process choreography diagram

The Reservation booking process can also be displayed as a choreography diagram which is mainly focused on the different participant interaction. Starting with a collaboration process model where each participant is separated as a separate pool, the choreography diagram will abstract from the work done in the individual pools, portraying only the message exchanges. As a result, a respective choreography diagram entails the following tasks:

- The first choreography task is displaying the interaction between the guest user (initiator) and the reservation application (receiver). More specifically, the guest triggers the reservation application by sending as a message the accommodation request.
- Following this, an exclusive gateway is introduced to evaluate the availability condition which is performed internally within the reservation application database. Depending on the condition evaluation result, the following choreography tasks are introduced:
    - In case accommodation is available, a parallel gateway splits the flow into two separate branches, resulting in two different choreography tasks running concurrently. More specifically in both cases, the reservation application sends an email message to the host and guest recipients participating in the process.
    - In case the accommodation is not available, the exclusive gateway leads to a choreography task in which the reservation task sends an email to the guest with the context of the accommodation non availability.
- Subsequently, another exclusive gateway splits the based on the host's decision to accept or to reject the reservation request. Both choreography tasks involve the host (initiator) and the reservation application (receiver) where the host is delivering a confirmation or rejection request.
- Finally, an event-based gateway is introduced which leads to two separate chorography tasks based on the request approval. More specifically:
    - In case the host confirms the reservation, the reservation application interacts with the guests by sending a confirmation email message.
    - In case the host confirms the reservation, the reservation application interacts with the guests by sending a rejection email message.

Overall, the choreography diagram emphasizes message exchanges and not internal processing. This is why specific internal actions within the Reservation application such as updating the reservation status, which is an internal database related operation, are abstracted from the choreography diagram. For the above reason the choreography diagram implements the collaboration diagram as it excludes the representation of additional activities and decision logic that the collaboration diagram includes.

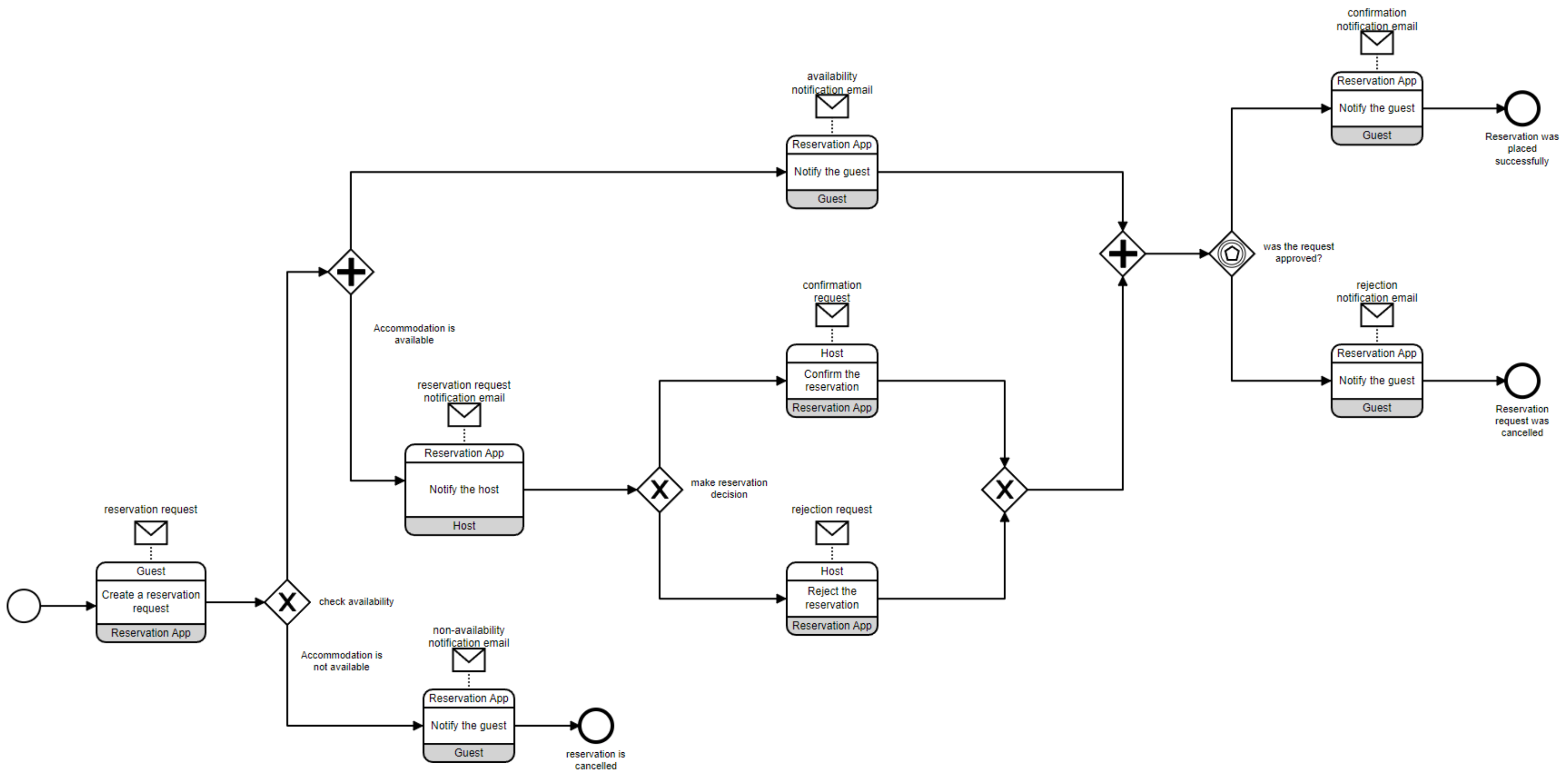


Figure 28: Reservation app choreography diagram

3.6 Chapter Summary

The chapter explores the need for standard modeling language for Business processes which led to the creation of BPMN and its broad usage. The open standard created in 2004 and improved in version 2.0 in 2011, offers a graphical depiction of business processes. Flow objects, connecting objects, data objects and artifacts are the essential categories in which essential components are grouped. These components are illustrating a range of business processes from simple tasks to more complex business process scenarios.

In fact, the foundation of process modelling is BPMN activities. Activities include a variety of tasks which involve user interactions and decision-making that can be both manual and automated. Conversely, gateways serve as processes decision making points. BPMN events are also essential for displaying incidents that trigger or respond to tasks within a business process. Events can be catching (waiting for a trigger) or throwing (started by the process). Finally, message flows can improve the collaborative aspect of BPMN diagrams by facilitating communication exchanges among different participants.

In BPMN context, collaboration and choreography diagrams are essential tools for displaying interactions among participants. First, collaboration diagrams concentrate on how different participants of a business process connect with each other and more specifically white-box collaboration diagrams give insights in each participant's internal activities. Meanwhile, choreography diagrams display participant interactions without focusing on its participant internal process. Therefore, they primarily focus on the message exchange between participants and the logical flow of these interactions. They also act as contracts, defining the messages to be transmitted and received and under what circumstances. All in all, choreography diagrams and can are useful in illustrating conversation intensive processes and can act as supplementary to collaboration diagrams as they abstract from the individual processes.

These concepts are examined via the Airbnb business process motivating example where an Airbnb inspired booking process is displayed via BPMN diagrams. To be precise, a reservation application can serve as a coordinator between the visitors and hosts is presented in a collaboration diagram. All interactions are displayed in detail starting from visitor requests for a reservation, moving on to availability checking, host and guest alerts and concluding for a reservation confirmation or rejection. This diagram considerably improves the comprehension of the booking process by outlining the precise flow of communications sent and received by all parties. In contrast to the collaboration reservation process diagram, the choreography diagram concentrates on message exchange between participants which is displayed via the choreography tasks rather than delving into each participant internal processing.

In summary, both collaboration and choreography diagrams have important functions for depicting business processes. Collaboration diagrams provide in-depth insights and provide a rich visualization of how each entity inside a business process interacts, communicates and shares information inside a business process. On the other hand, choreography diagrams abstract from internal mechanisms and complexities and focus entirely on the precise sequence and timing of message exchange between the process participants. By doing so, they highlight essential interactions, however they don’t get into detail about internal mechanisms.

Chapter 4 : REST and BPMN

4.1 Application Programming Interface evolution

An API (Application Programming Interface) is a collection of features and rules that are built inside an application, enabling computer-to-computer communication rather than relying on a user interface screen [34]. It can act as an interface between the application that offers it and items such as third-party software and hardware. APIs that enable communication and interaction via the web are also called web services. REST is rapidly becoming one the most preferable architectural style of choice for developing web services, resulting in the widespread development of RESTful APIs. Representational State Transfer (REST) is one of the most widely used architectural approaches for Web interaction [35].

The REST API History shows a dramatic shift in the way developers use APIs to communicate between the servers. Prior to REST, the dominant protocol was SOAP, which was noted for its complexity and the requirement for handcraft XML-based documents for executing remote process calls (RPC). REST, however, was created in 2000 providing a set of limitations that made APIs more accessible and easier to integrate. REST, like SOAP, increased in popularity and became the de facto means of developing web-based APIs over time.

In 2015, Facebook announces GraphQL, an API query language that introduces a new approach of constructing web-based APIs in comparison to REST. When it comes to web-based APIs, we can see from a Google trends diagram (Figure 29) that REST APIs are the most popular type of APIs used currently, however GraphQL is gaining ground. On the other hand, the SOAP protocol is quickly losing favor.

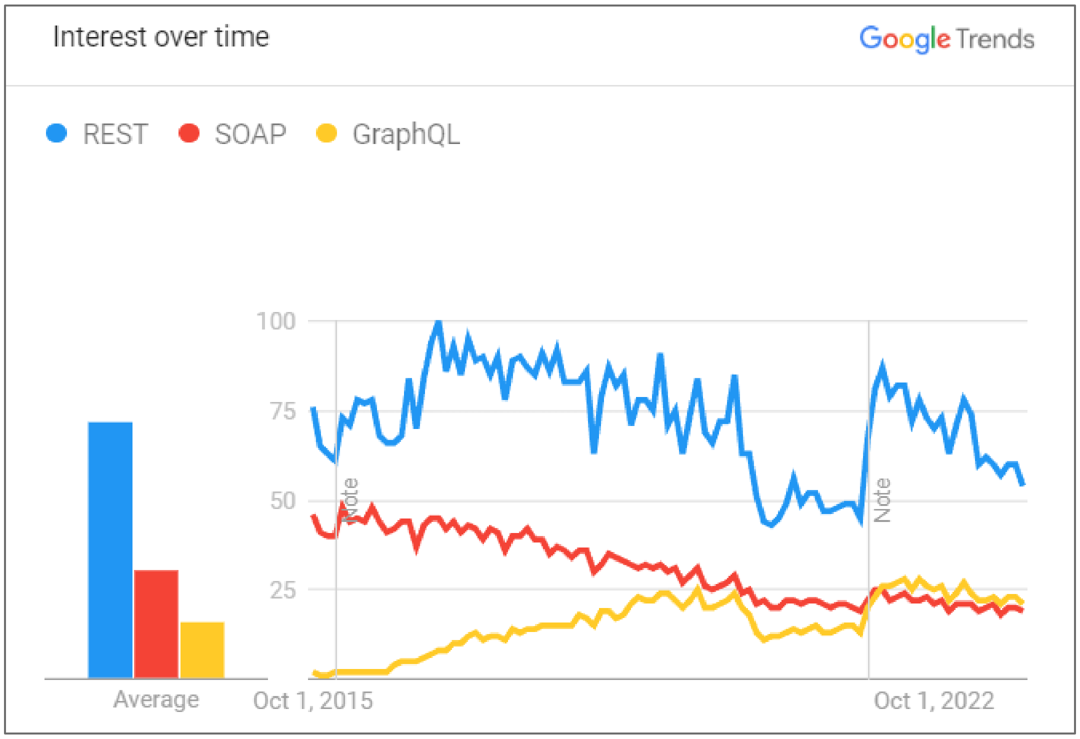
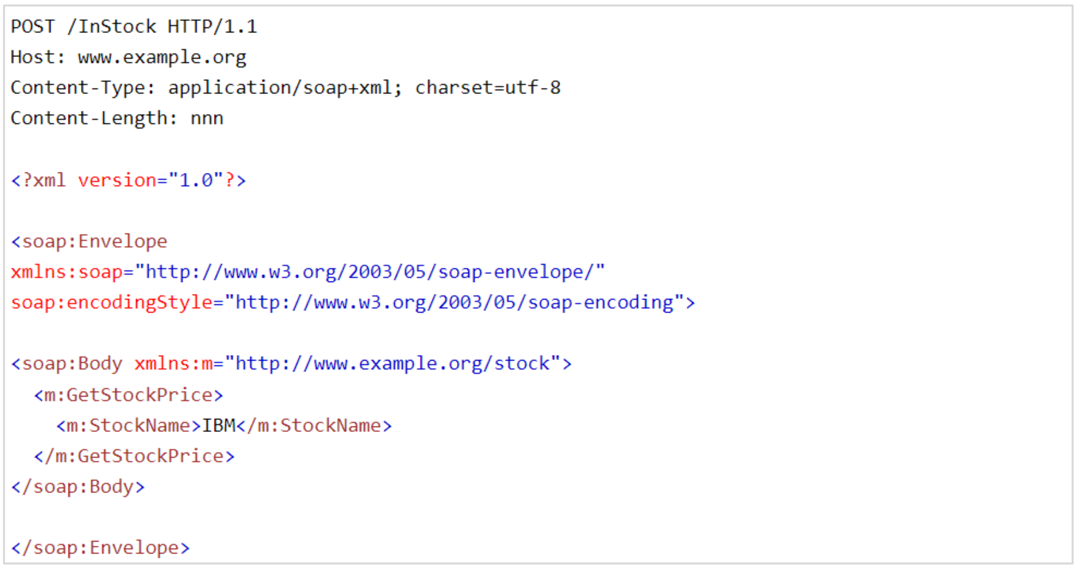


Figure 29: APIs popularity in google trends.

SOAP and REST are two different techniques for developing APIs. SOAP is a protocol whereas REST is an architecture style with a set of protocols. REST enables more flexible development using mostly JSON format in message exchange. In contrast, SOAP sends data using XML format. SOAP are XML-based messages with a certain structure, that include an envelope, header, and a body. More specifically, the body entails the entire message while defining its start and end. If present the header includes the metadata such as security information and other credentials when the body includes the main content of the message (Figure 30).



```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

Figure 30: Example of a SOAP request [36].

All in all, SOAP gives more specific information about what API performs, but it may not be appropriate for newer applications. However, SOAP is a robust trustworthy style of webservices in business-to-business communication and government systems where security plays an important role, and when REST is not required.

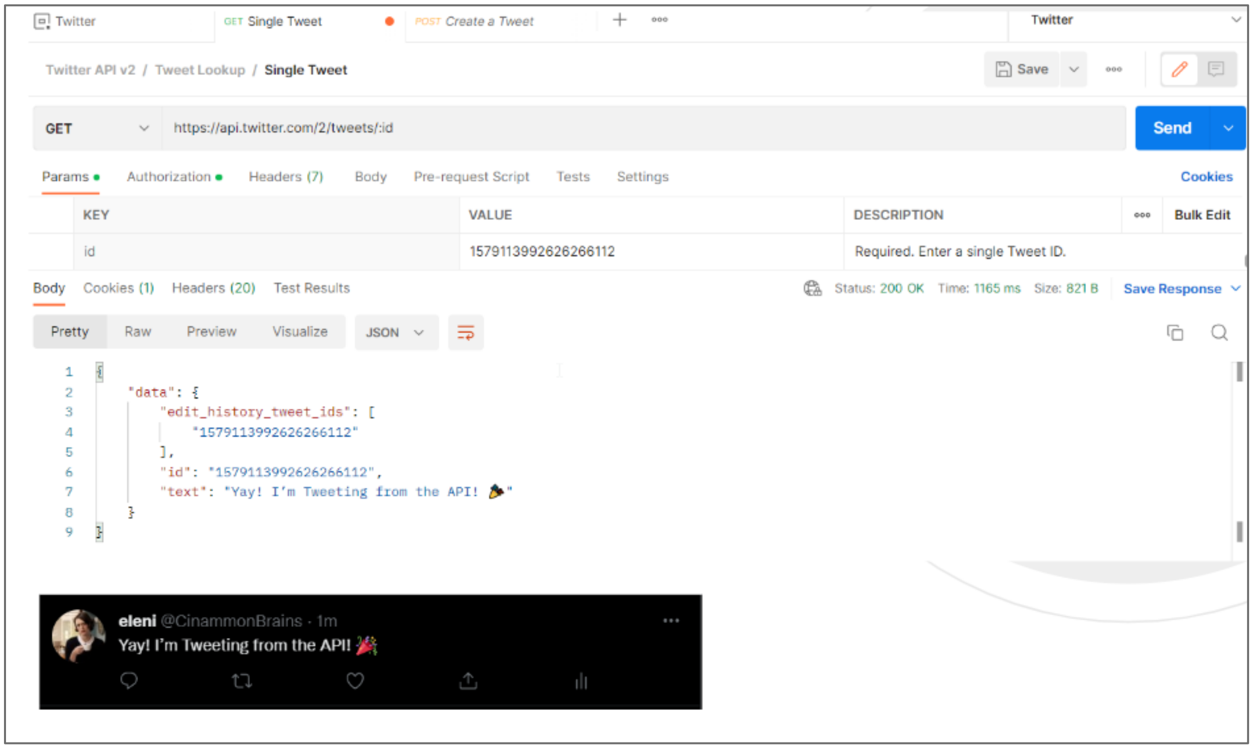
However, REST outperforms SOAP in various ways. Scalability is a crucial feature, as REST allows separation between the client and the server permitting their independent expansion without any difficulties. Furthermore, REST APIs are more flexible and portable, easing server migration and permitting database modifications. Moreover, REST is lightweight and quick by taking advantage of HTTP standard and by supporting various formats such as XML, HTML and JSON. REST are suitable for IoT devices and mobile applications and other projects that require efficiency and speed due to their lightweight nature.

RESTful APIs are used by numerous elite companies, in a variety of industries:

- Amazon S3 offered by Amazon web services (AWS) is an API that allows developers to obtain access to Amazon S3 functionality, with features like retrieving any amount of data anywhere on the web. It incorporates API with AI technology, enabling adaptive interactions and improving data security.
- Twitter (recently renamed to X) provides an API for developers, allowing smooth integration with applications, easing registration process, and displaying tweets based on

location enabling marketing. For example, it can be used to retrieve data from a user's twitter accounts, for example, to display their latest tweets on a web page (Figure 31).

- Google APIs are a set of application programming interfaces created by google for developers, enabling them to interact with google services and products. These APIs allow developers to connect with google services, access user data, and take activities on user’s behalf by using google services.



KEY	VALUE	DESCRIPTION
id	1579113992626266112	Required. Enter a single Tweet ID.

```
{
    "data": {
        "edit_history_tweet_ids": [
            "1579113992626266112"
        ],
        "id": "1579113992626266112",
        "text": "Yay! I'm Tweeting from the API! 🎉"
    }
}
```

eleni @CinammonBrains · 1m
Yay! I’m Tweeting from the API! 🎉

Figure 31: Twitter API example

4.2 Representational State Transfer (REST) Architectural Style

When it comes to building newer applications, REST is critical for enabling communication of various systems via the internet. Any item of information or capability accessed by the API is seen as a resource and can be represented in JSON. The resources resting on a server are globally and uniquely identified via a URL. RESTful architecture is based on a client-server model enabling request and response message exchange. A client-API contract may enable the creation and deletion of the API resources, or the update and the retrieval of their representation. Each communication is stateless. Therefore, the server does not remember the previous interactions with the client. As a result, the state of the conversation is managed as well as maintained by the client [37].

REST can use HTTP protocol as a means of interactions between a client and a server. REST architecture's standardized interface provides the communication primitives utilized in a conversation [37]. In the case of HTTP-based APIs, each communication is started by the client and takes the form of a request to the server, followed by a response message. The interactions are achieved by using the standard HTTP verbs (GET, POST, PUT, DELETE) on resources. Subsequently, there are four HTTP verbs GET, POST, PUT and DELETE respectively. POST will create a new resource with an identifier specified by the server, whereas GET returns the status of the resource. PUT will update a resource with the client's state and identifier while DELETE removes it [28].

An API flow can be initiated by a 'Client app' that can be any type of system with or without a user interface, requesting information for a unique user. The REST API recognizes the data source of the requesting resource, reads its content, and returns a response in JSON or XML format.

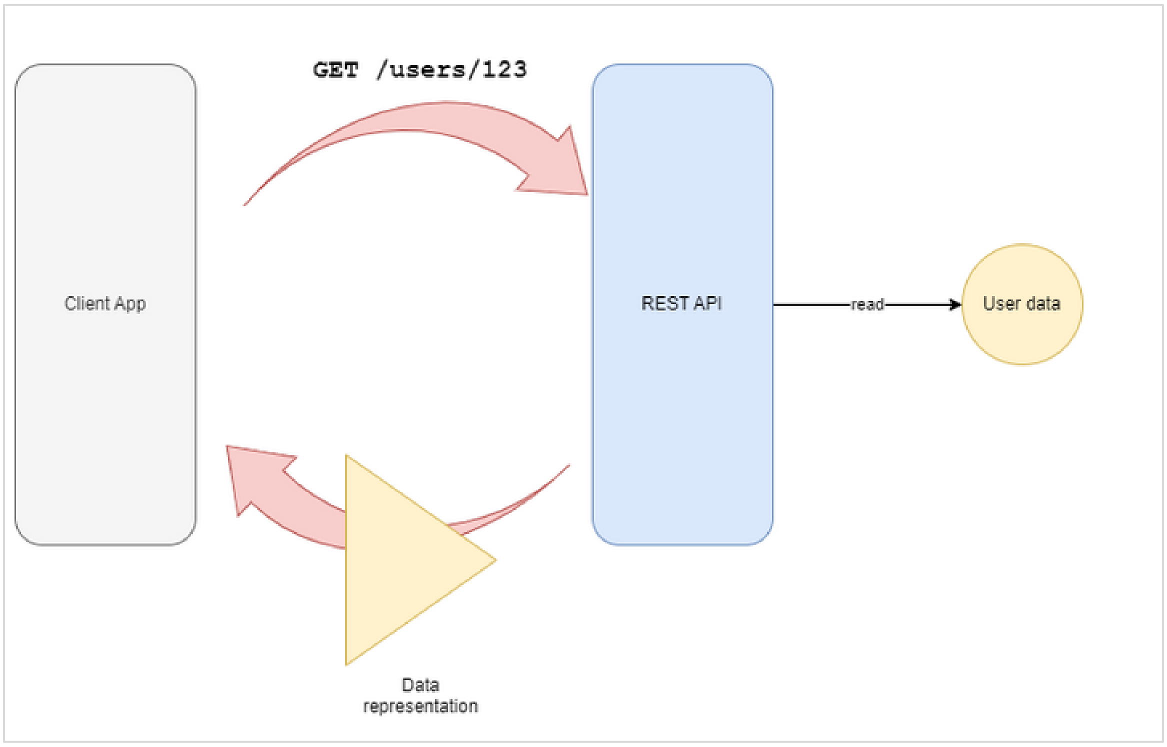


Figure 32: RESTful API example [38]

Overall, according to [39], the main principals of RESTful API can be summarized as follows:

- Statelessness is the first core principle of RESTful APIs. The client is responsible for consuming the returned data while the server is responsible for processing the resource data. The server stores no client-specific state, which simplifies deployment and increases scalability.
- Client and Server are separate entities, which means the client is responsible for using the data retrieved when the server is responsible for processing the resource data. As a result, they can evolve and scale independently.
- REST APIs ought to employ a uniform interface, which means that they should follow a consistent set of standards when interacting with resources. This includes performing activities called with the acronym CRUD for creating a resource, reading a resource, updating, and deleting a resource respectively while using HTTP methods. Furthermore, APIs should employ HTTP headers and status codes, to communicate the metadata of the requests.
- Each resource should have its own unique URL. The API should offer consistent and intuitive structures for exploring resources.
- HATEOAS (Hypermedia as the Engine of Application State) signifies that an API response should provide links to similar resources. These connections provide discoverability and allow clients to dynamically browse through the APIs.
- Data for each resource can be provided with a variety of data formats. A consumer for instance should be able to request product information in XML or JSON format.

By conforming to the above standards, a RESTful API can achieve traits like scalability, flexibility, and interoperability. The use of REST APIs provides loose coupling, making the system easier to evolve and maintain overtime. However, REST APIs may be vulnerable to security issues such as lack of proper authentication and failure to encrypt payload data. Despite these problems, careful design and execution is critical for ensuring that RESTful APIs are used effectively and securely.

4.3 RESTful Conversations Modelling with Extended BPMN Choreography Diagrams

Choreography tasks are used in representing specific participant interactions inside a business process. In addition, communication between different parties can be done via REST services. In this context, and by incorporating REST annotations, choreography tasks can be utilized in modeling HTTP request/response interaction, and email notifications between different roles engaged in the process. Messages in a choreography task can be employed with annotations, containing REST metadata. This metadata can contain information such as HTTP methods and REST service endpoints, and any required headers or parameters.

Moreover, hyperlinks can be provided to the request and response to guide participants in following the RESTful choreography. Subsequently, a choreography diagram can be implemented with REST annotations, where the requested message can be specialized to HTTP requests and HTTP responses (

Figure 33). For example, inside a choreography task an HTTP request message can entail, the HTTP method and the API endpoint and any request data (e.g., JSON payload). Following this information, the HTTP response would comprise information such as status code, response header and body, including hyperlinks for continuing the business flow. Furthermore, an email notification may include an email as an initiating message which may conclude in resuming the process.

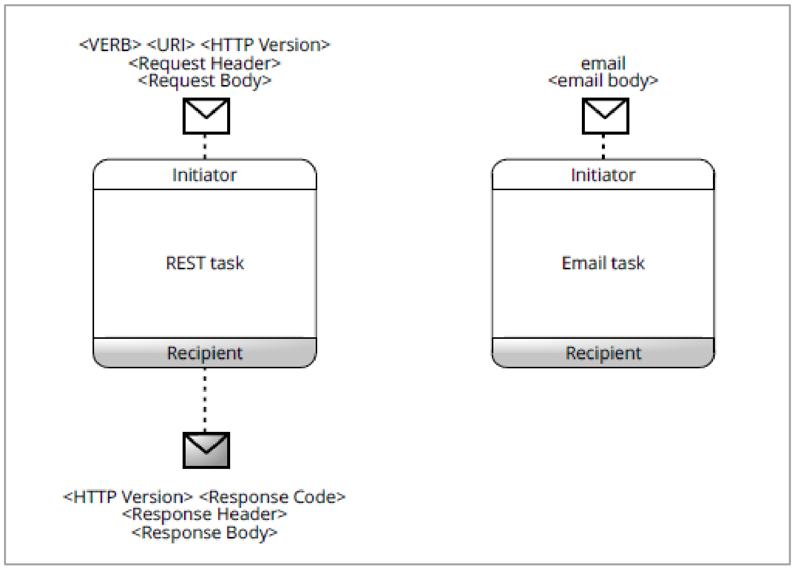


Figure 33: REST annotations in a choreography diagram

By incorporating these elements into the choreography diagram, a visual representation of how different participants interact with different REST services can be presented. This method not only aids in the documentation of the HTTP request and response flow, but it also provides a clear picture of the process as it comes to RESTful interactions. According to [40] modeling with REST annotations is preferable to developing new graphical features or metamodels, as it enables RESTful choreographies to be integrated into the existing modeling ecosystem. However, if necessary, a choreography metamodel can be built by parsing the REST messages to extract specific information.

Links are also important in REST interactions, as different patterns can access and alter shared resources. Link patterns have been observed and categorized based on the number of links conveyed in a choreography task. These patterns are useful in understanding how they can influence choreography task interaction dynamics [28].

- No link pattern: In the no link pattern, a choreography task may include an HTTP message or an email but no links. This is often a notification message, giving brief information on the resource such as each deletion. This pattern may be followed by a conclusion or activity involving other participants.
- Single link pattern: A choreography task with a single link pattern involves only a single link. This link serves as a handler for subsequent exchanges, allowing the recipient to gain more information. For instance, an email can contain a link leading to a subsequent action.
- Multi-link pattern: A choreography task that includes many links, is usually followed by an exclusive or an event-based gateway. By having conditional sequence flows, this pattern can be utilized, when alternative resources can result in different results.

In the below choreography example, the Single link pattern reflects the creation of the customer's order. A 201 created status code is returned along with the link of the order details. Subsequently, in the next choreography task, the customer receives an email notification where the multi-link pattern is followed, and the response contains multiple links. When a customer wishes to make a return request the links can be utilized in proceeding with this action. Exchange or refund can be carried out depending on the customer's link selection and accompanied by a confirmation email.

This example highlights the importance of links as it comes to interactions in a choreography diagram.

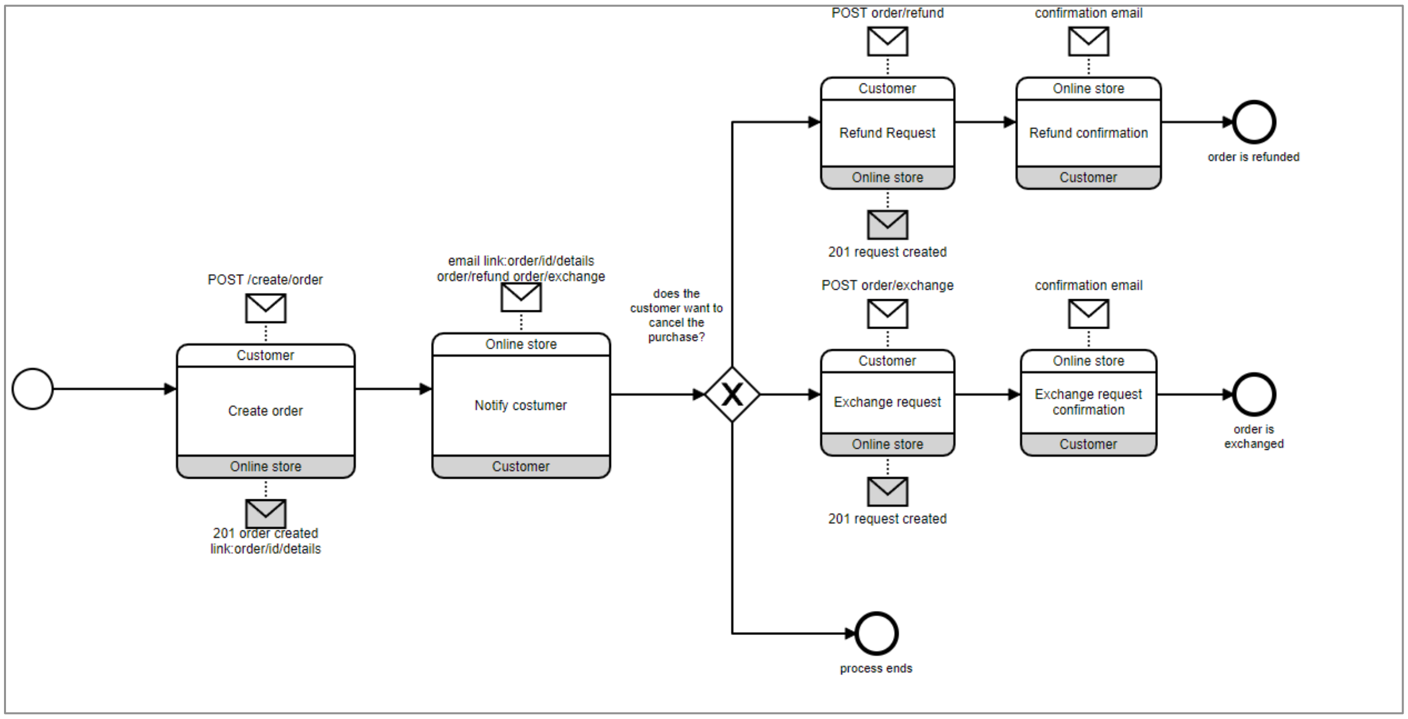


Figure 34 : Choreography diagram example

In general, RESTful choreography diagrams are aiming at displaying all valid interactions that will lead to many different outcomes. These will permit the prediction of all possible interactions based on its shared resources.

4.4 Reservation app booking process choreography diagram extended with REST annotations.

The first choreography task is initiated by the guest with a RESTful POST request representing the reservation accommodation request. The choreography task follows in the single link pattern as the guest receives a hyperlink containing the location of the reservation details. The reservation request task is followed by an exclusive gateway, for evaluating the availability condition inside the reservation database via a RESTful GET request. The following choreography task is introduced depending on the condition evaluation.

- In case accommodation is available, a parallel gateway splits the flow into two separate branches resulting in two different choreography tasks happening concurrently. In both cases, an email notification request is sent to the host and guest involved in the business process. The 'Notify the host' the multi-link pattern as the email entails the hyperlinks that need to be followed for confirming or rejecting the request.
- In case the accommodation is not available, the exclusive gateway initiates a choreography task, in which a reservation notification email is sent to the guest informing of the accommodation's non availability. The process is terminated, and the reservation application is deleting the reservation request with an internal RESTful DELETE request operation.

Another exclusive gateway is introduced depending on the host acceptance or rejection of the reservation request. The host (initiator) may confirm or reject the reservation request via two separately defined RESTful POST requests to the reservation application (recipient). More specifically, in case the host wants to accept the request, the host performs the POST accept request operation, whereas in case of denial the host performs the POST reject operation.

Finally, an event-based gateway is established, which results in two distinct choreography tasks based on the request approval.

- If the host confirms the reservation the reservation application communicates with the guest, by sending a confirmation email. The process ends with the reservation application changing the reservation status to complete with an internal RESTful POST request operation.
- If the host rejects the reservation the reservation application is sending a refusal email message to the guest. The process ends with the reservation application deleting the reservation request with an internal RESTful DELETE request operation.

All in all, the choreography diagram points out the RESTful interactions inside the reservation process with emphasis on external participant interactions from internal operations inside the reservation application.

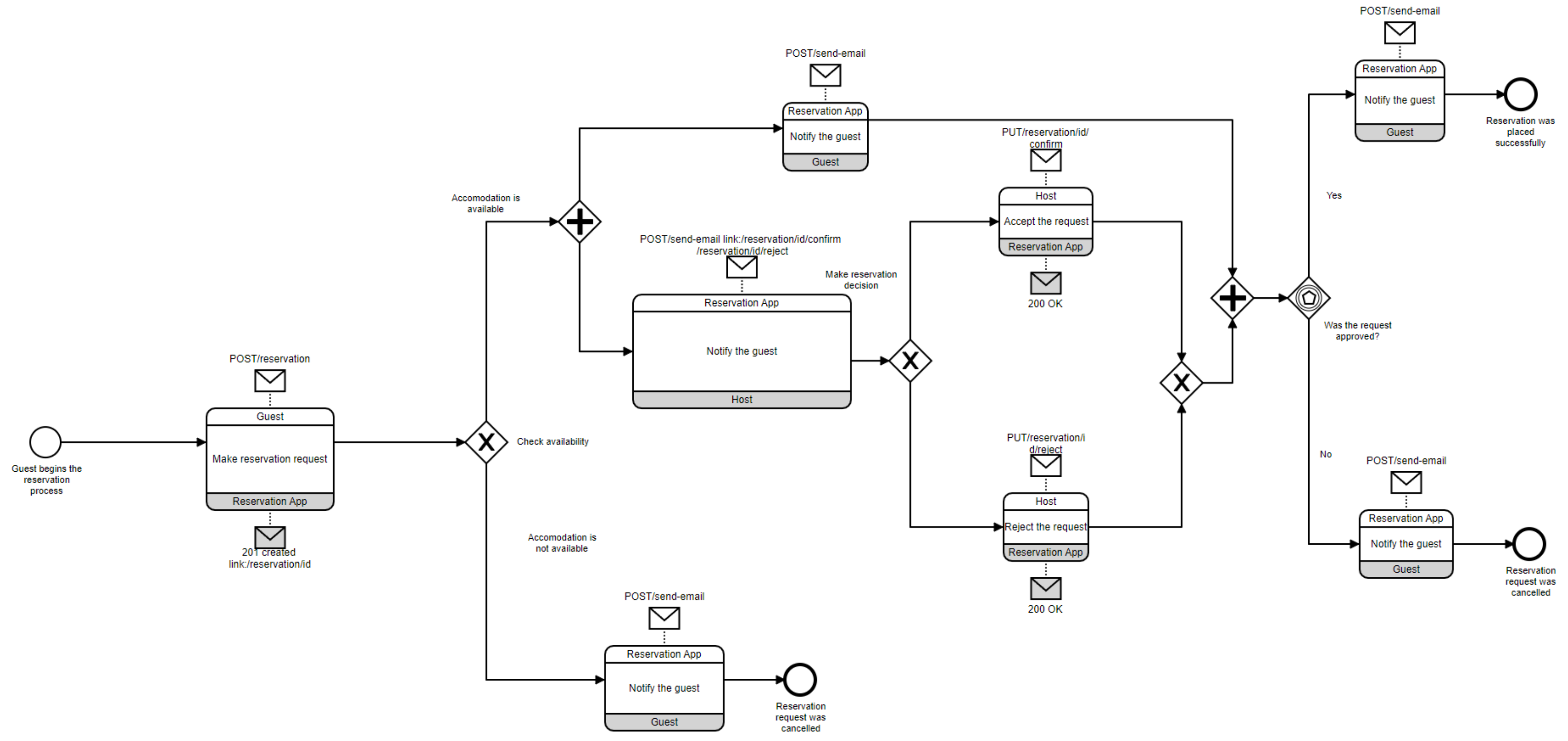


Figure 35: Choreography diagram of reservation app with REST annotations

4.5 Chapter Summary

This chapter explores how to incorporate REST APIs into choreography diagrams. First, it examines how API technologies have changed over time, and focuses on the switch from SOAP to REST and the birth of GraphQL. The importance of REST APIs in modern application development is emphasized in the first part of this chapter, particularly in terms of scalability, flexibility and portability.

Next, the integration of REST APIs and BPMN choreographies which introduces the idea of REST annotations within a business process is analyzed. The reservation application business process is showcased for modeling how HTTP request/response exchange and email notification can be modeled. In order to emphasize the importance of links in these interactions, the chapter further divides RESTful interactions into types like no link, single link, and multi-link. Overall, this chapter offers a thorough grasp of how to incorporate RESTful APIs into BPMN choreography diagrams, facilitating seamless communication between various business actors.

All in all, this chapter emphasizes that businesses can efficiently describe and visualize interactions by incorporating RESTful interactions into BPMN choreography diagrams. It serves the purpose of closing the gap between RESTful APIs implementation and process modeling and make it possible to construct reliable, interoperable and effective applications, which can be a helpful guide for developers and business analysts.

Chapter 5 : From Modeling to Implementation level

5.1 MVC Architecture and Dependency Management in RESTful API development

For building the Reservation API the Model-View-Controller (MVC) pattern used in Spring-boot-applications has been utilized.

More specifically the Reservation REST API is implemented in four different layers:

- Model Layer: The model represents the application structure and the entities involved in the application.
- Controller Layer: Which is the initial layer responsible for receiving and sending HTTP requests. As it is the top layer, it calls the service layer after receiving input or a request from the client. After processing the service layer's output, the client receives the response.
- Service Layer: It is the middle layer where all business logic is being executed.
- Repository Layer: Is the layer where database communication is performed.

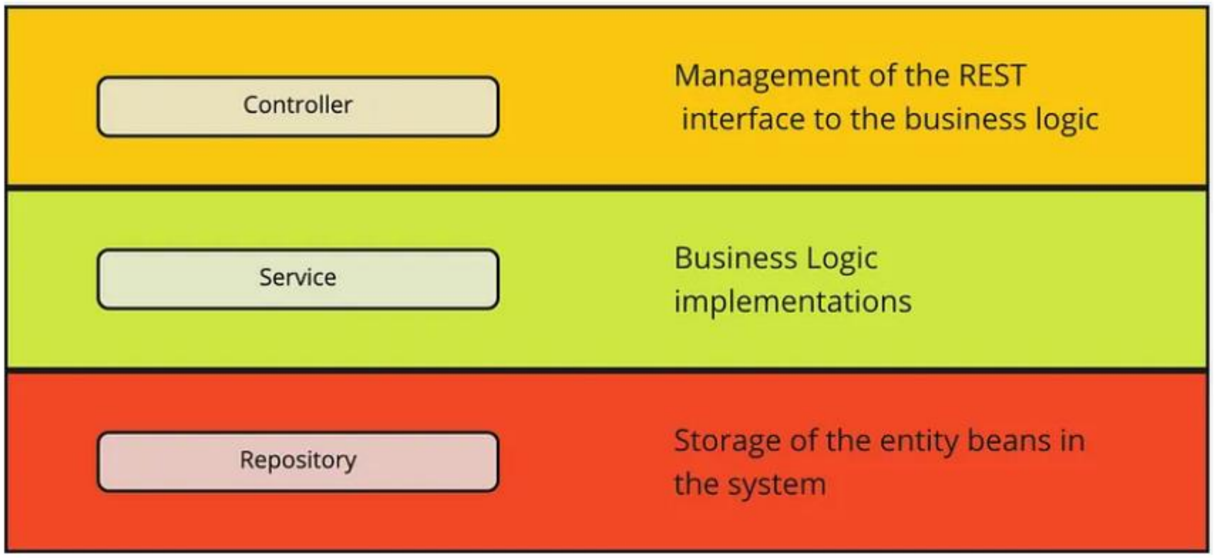

Controller
Management of the REST interface to the business logic
Service
Business Logic implementations
Repository
Storage of the entity beans in the system

Figure 36: The Controller-Service-Repository layers [41]

The tool for managing dependencies used with Spring Boot is Maven. Although it may be used for other programming languages, Apache Maven is a popular build automation tool used for java applications. It facilitates project development, as it gives structure to the project through a set of conventions. More specifically, it provides the Project Object Model (POM) file, which is the main control file in Maven and defines the project [42].

The pom file is described as an XML file where project structure, dependencies and configurations of the project are specified. In addition, the dependencies declared in the pom.xml file, are downloaded in remote dependencies automatically, which makes it easier to ensure that all necessary libraries and dependencies are available [42]. As a result, when executing a task, Maven searches its directory in the POM file.

Maven also provides a standard way of building a new project, as it provides a project directory layout which simplifies the project's set up and maintenance. The standard layout makes it easier for Maven to identify and manage project elements. Maven project structure can also be customized however, the default pattern is suggested, as it helps maintaining consistency. All in all, Maven simplifies the process of building a java-based project, as the task of downloading Jar files and other dependencies is not done manually.

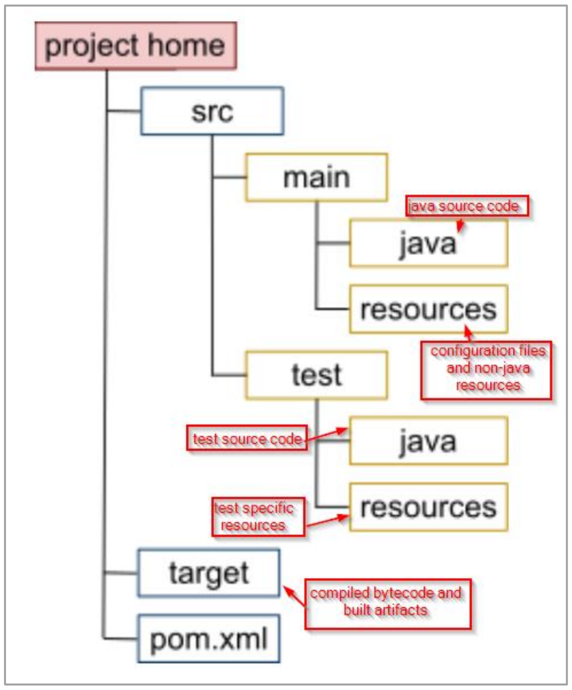


Figure 37: Maven directory structure [43]

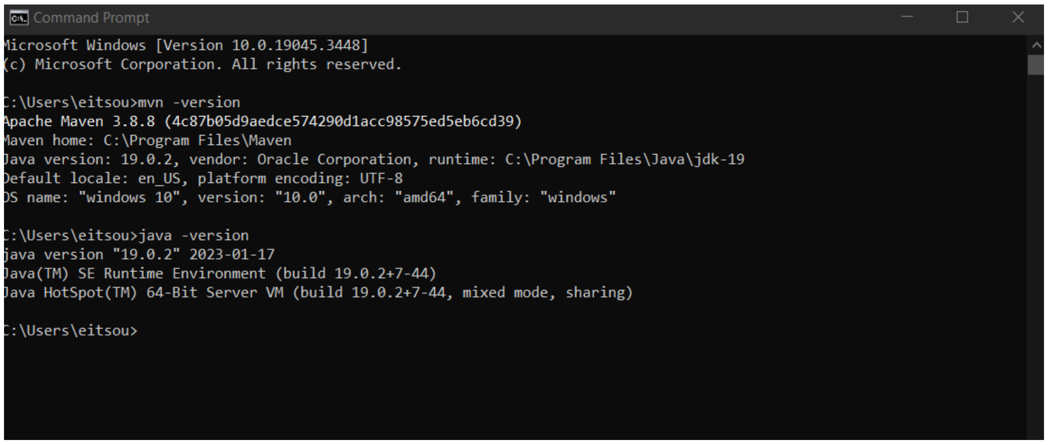
The standard structure of a Maven project is presented in the following table:

Directory	Description
src/main/java	This directory includes your project's primary Java source code.
src/main/resources	This directory contains configuration files, property files, and other non-Java resources needed by your program.
src/main/webapp (optional)	This directory contains web-related resources for web applications, such as HTML files, JSPs, CSS, JavaScript, and web.xml
src/test/java	This directory includes the source code for your test cases, which are commonly written in Java and run using testing frameworks such as JUnit.
src/test/resources	This directory, like src/main/resources, contains test-specific resources.
target	During the build process, Maven generates compiled bytecode, JARs, WARs, and other build artifacts in this directory. It is the output directory for build results. Maven generates compiled bytecode, JARs, WARs, and other build artefacts in this directory during the build process.
pom.xml	The Project Object Model (POM) file, which is the central configuration file for your Maven project. It includes project metadata, dependencies, and other project-specific settings.

Table 5.1-1: Maven project structure

5.2 Building a Reservation RESTful API with Spring Boot, Mongo DB and Swagger: A comprehensive implementation guide

Before building the project, Maven and Java were installed locally with 3.8.8 Maven and Java 19.0.2 versions respectively.



```
icrosoft Windows [Version 10.0.19045.3448]
(c) Microsoft Corporation. All rights reserved.

:\Users\eitsou>mvn -version
pache Maven 3.8.8 (4c87b05d9aedce574290d1acc98575ed5eb6cd39)
aven home: C:\Program Files\Maven
ava version: 19.0.2, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-19
efault locale: en_US, platform encoding: UTF-8
S name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

:\Users\eitsou>java -version
java version "19.0.2" 2023-01-17
ava(TM) SE Runtime Environment (build 19.0.2+7-44)
ava HotSpot(TM) 64-Bit Server VM (build 19.0.2+7-44, mixed mode, sharing)

:\Users\eitsou>
```

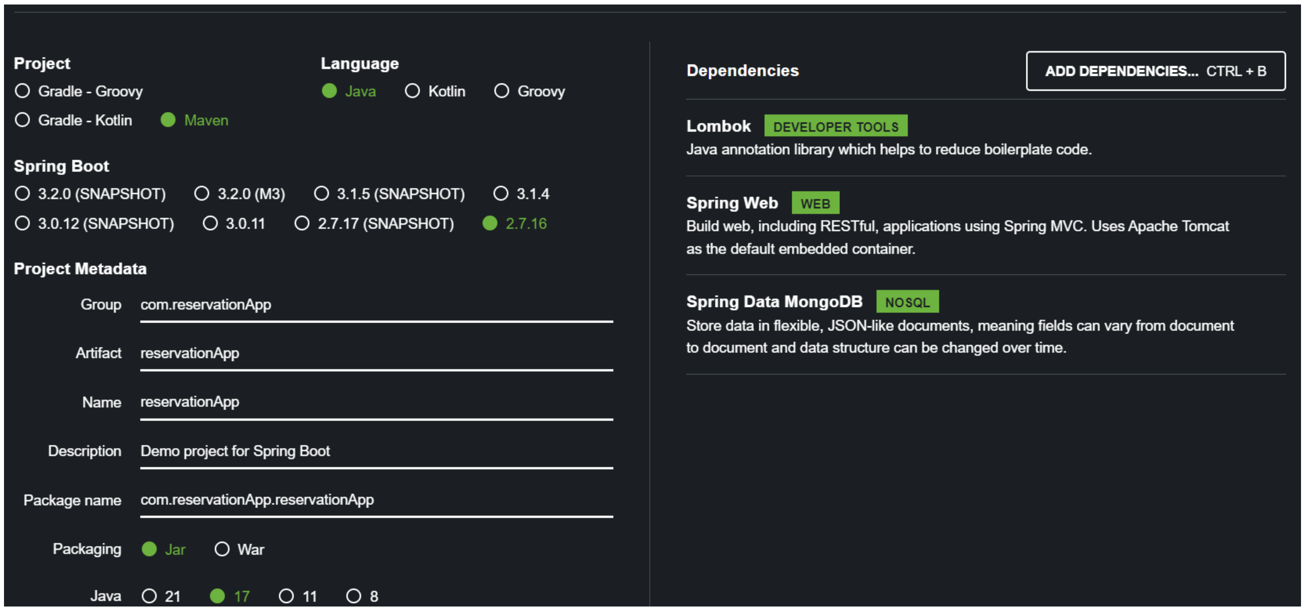
Figure 38: Java and Maven version installed.

The skeleton of the application was generated in spring boot initializer website. The project type selected is Maven and the preferred language is Java. In addition, project metadata are generated like the group and the artifact name. Moreover, the preferred packaging type and the java version are selected. Spring Web is the dependency that provides the spring boot framework and enables building Restful applications and provides default configuration for MVC (Model-View-Controller) design pattern.

Spring Web entails spring-boot-starter-web dependency and a mixture of other components needed to start a web application. Spring web starter also entails a default JSON marshaller responsible for automatically converting data objects to JSON format. In addition, Apache Tomcat is a default embedded server which is responsible for building an executable JAR Application. Furthermore, spring-boot-starter-test testing dependency is also included in the spring web application dependency.

More dependencies were also added in the Spring boot initializer:

- Lombok java annotation library for reducing boilerplate code.
- Spring Data Mongo DB dependency for communicating with a Mongo DB database.



Project
○ Gradle - Groovy ○ Gradle - Kotlin [✓ Maven]

Language
[✓ Java] ○ Kotlin ○ Groovy

Spring Boot
○ 3.2.0 (SNAPSHOT) ○ 3.2.0 (M3) ○ 3.1.5 (SNAPSHOT) ○ 3.1.4
○ 3.0.12 (SNAPSHOT) ○ 3.0.11 ○ 2.7.17 (SNAPSHOT) [✓ 2.7.16]

Project Metadata

Group	com.reservationApp
Artifact	reservationApp
Name	reservationApp
Description	Demo project for Spring Boot
Package name	com.reservationApp.reservationApp
Packaging	[✓ Jar] ○ War
Java	○ 21 [✓ 17] ○ 11 ○ 8

Dependencies

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data MongoDB NOSQL
Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

Figure 39: Springboot Initializer and the dependencies added.

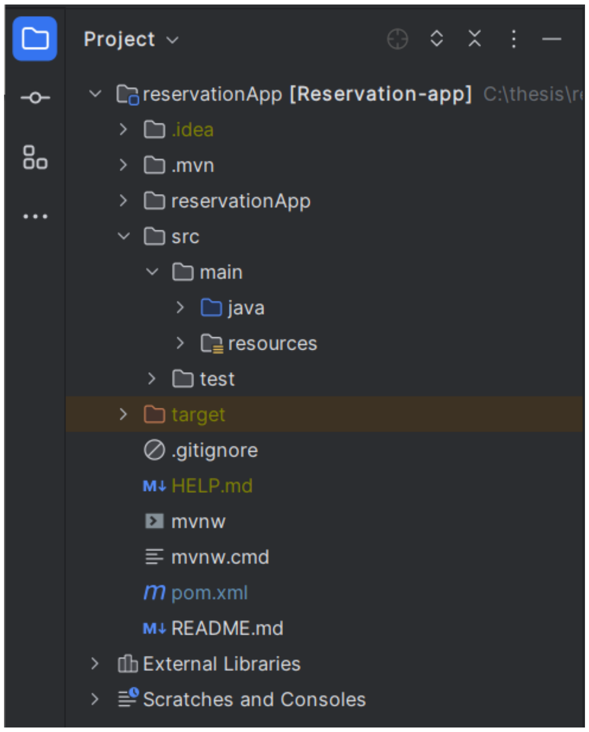
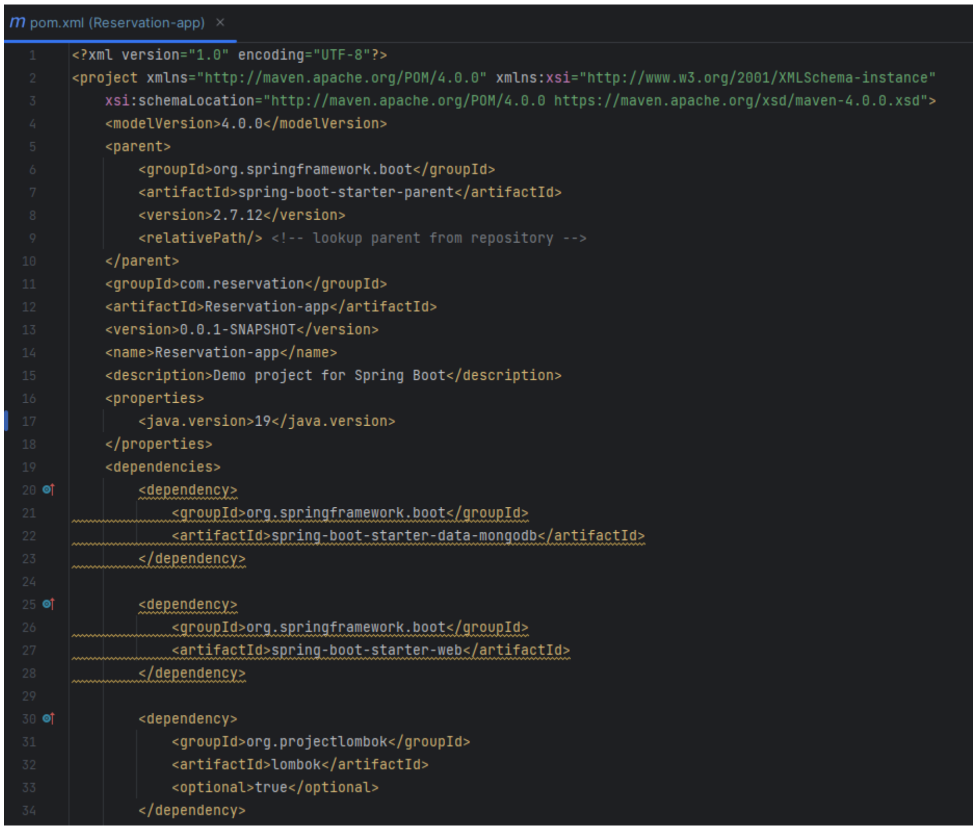


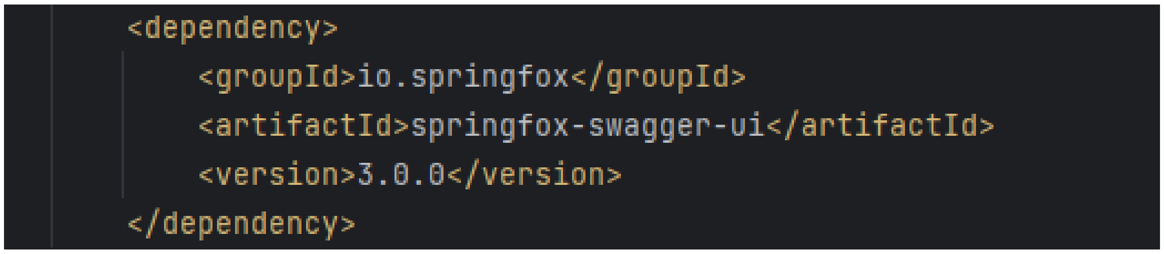
Figure 40: Project structure generated via Spring Boot initializer.



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.12</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.reservation</groupId>
    <artifactId>Reservation-app</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Reservation-app</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>19</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-mongodb</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
```

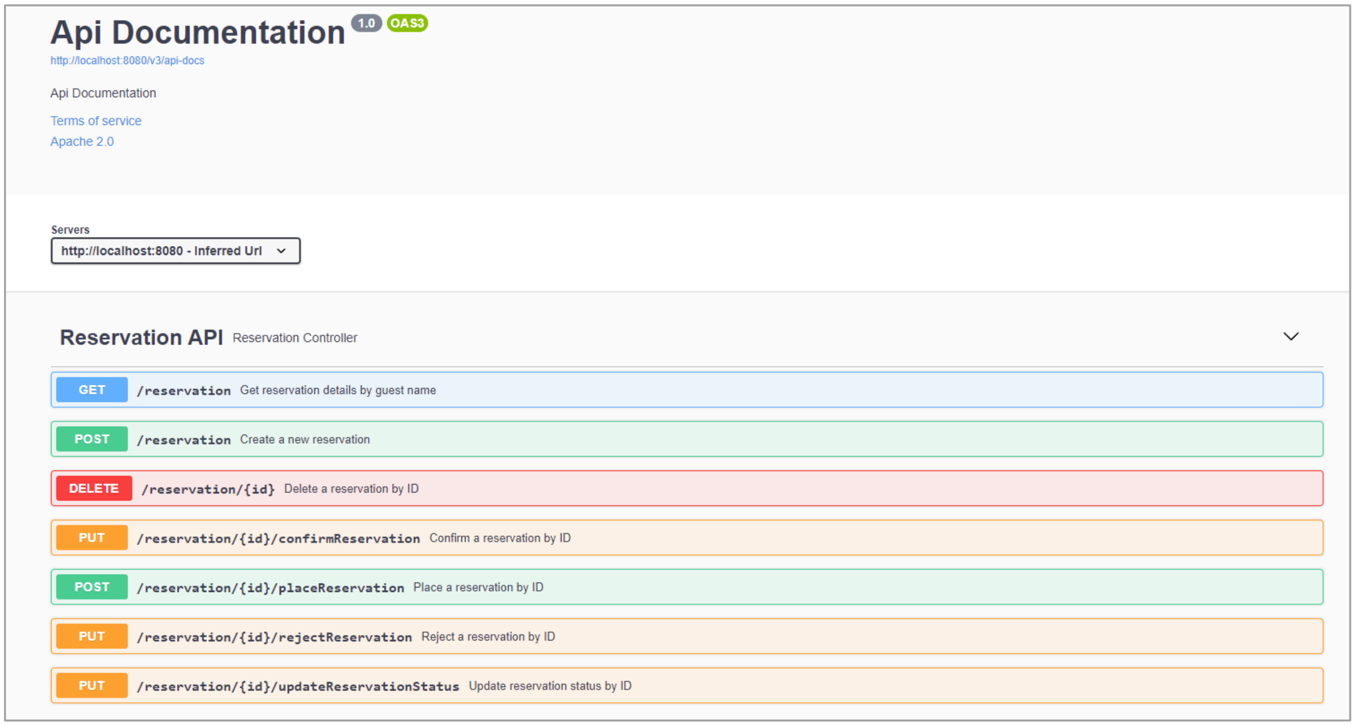
Figure 41: pom.xml and Spring boot dependencies

The file is downloaded and imported in the IntelliJ IDE having the structure of a Springboot Maven project. In addition, it contains the above pom.xml file where all the dependencies can be found (Figure 41). In the pom.xml file springfox-swagger-ui dependency was also added (Figure 42). This indicates that we can use Springfox Swagger UI for generating API documentation. When included in the spring boot application, documentation including information about API endpoints is immediately generated and accessible via Swagger UI (Figure 43).



```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>3.0.0</version>
</dependency>
```

Figure 42: springfox dependency in the pom.xml file



Api Documentation 1.0 OAS3

http://localhost:8080/v3/api-docs

Api Documentation

Terms of service

Apache 2.0

Servers

http://localhost:8080 - Inferred Url

Reservation API Reservation Controller

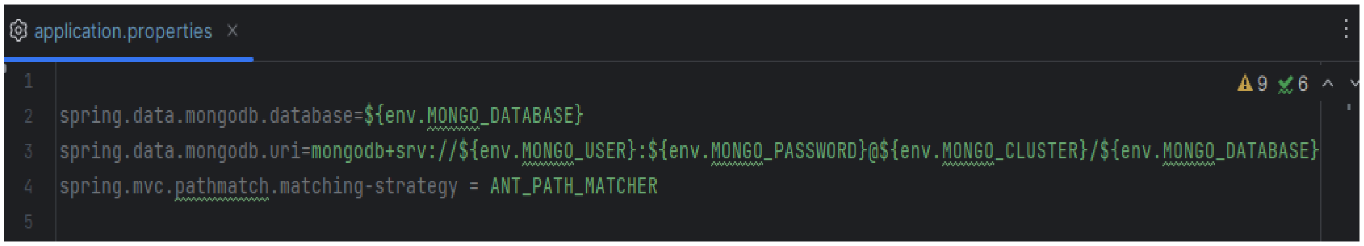
```
GET     /reservation                                  Get reservation details by guest name
POST    /reservation                                  Create a new reservation
DELETE  /reservation/{id}                             Delete a reservation by ID
PUT     /reservation/{id}/confirmReservation          Confirm a reservation by ID
POST    /reservation/{id}/placeReservation            Place a reservation by ID
PUT     /reservation/{id}/rejectReservation           Reject a reservation by ID
PUT     /reservation/{id}/updateReservationStatus     Update reservation status by ID
```

Figure 43: API documentation of the Reservation API in the Swagger UI

Swagger Annotations	Definition
@Api	Provides metadata about the API, including its name or tags. This can be used by API documentation tools like Swagger.
@ApiOperation	Describes an API operation. In this case, it describes the purpose of the save method, which is to create a new reservation.
@ApiResponse	Defines specific responses with details such as HTTP status code and message.

Table 5.2-1: Swagger annotation explanation

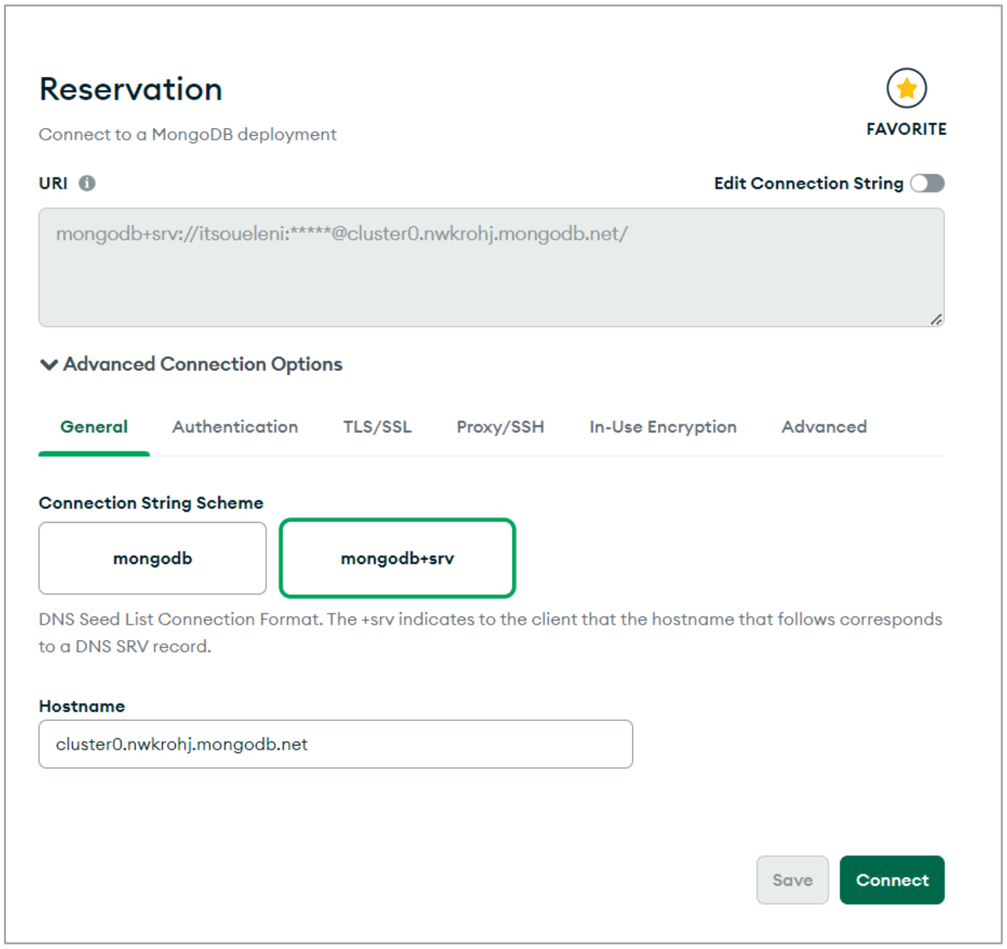
MongoDB is the database utilized in the Reservation API implementation. MongoDB Atlas, a cloud-based database service supplied upon creating an account, was used to create a new database. The configuration of a cluster that will act as a database repository is being done using Mongo DB Atlas. The database may be accessed using MongoDB compass, which is the UI interface where the database credentials are entered. These credentials are also required for connecting to the database via the java application.



application.properties

```

spring.data.mongodb.database=${env.MONGO_DATABASE}
spring.data.mongodb.uri=mongodb+srv://${env.MONGO_USER}:${env.MONGO_PASSWORD}@${env.MONGO_CLUSTER}/${env.MONGO_DATABASE}
spring.mvc.pathmatch.matching-strategy = ANT_PATH_MATCHER

```

Figure 44: MongoDB database connection credentials in JAVA application properties



Reservation

Connect to a MongoDB deployment

URI

mongodb+srv://itsoueleni:*****@cluster0.nwkrohj.mongodb.net/

Edit Connection String

Advanced Connection Options

General | Authentication | TLS/SSL | Proxy/SSH | In-Use Encryption | Advanced

Connection String Scheme

mongodb [✓ mongodb+srv]

DNS Seed List Connection Format. The +srv indicates to the client that the hostname that follows corresponds to a DNS SRV record.

Hostname

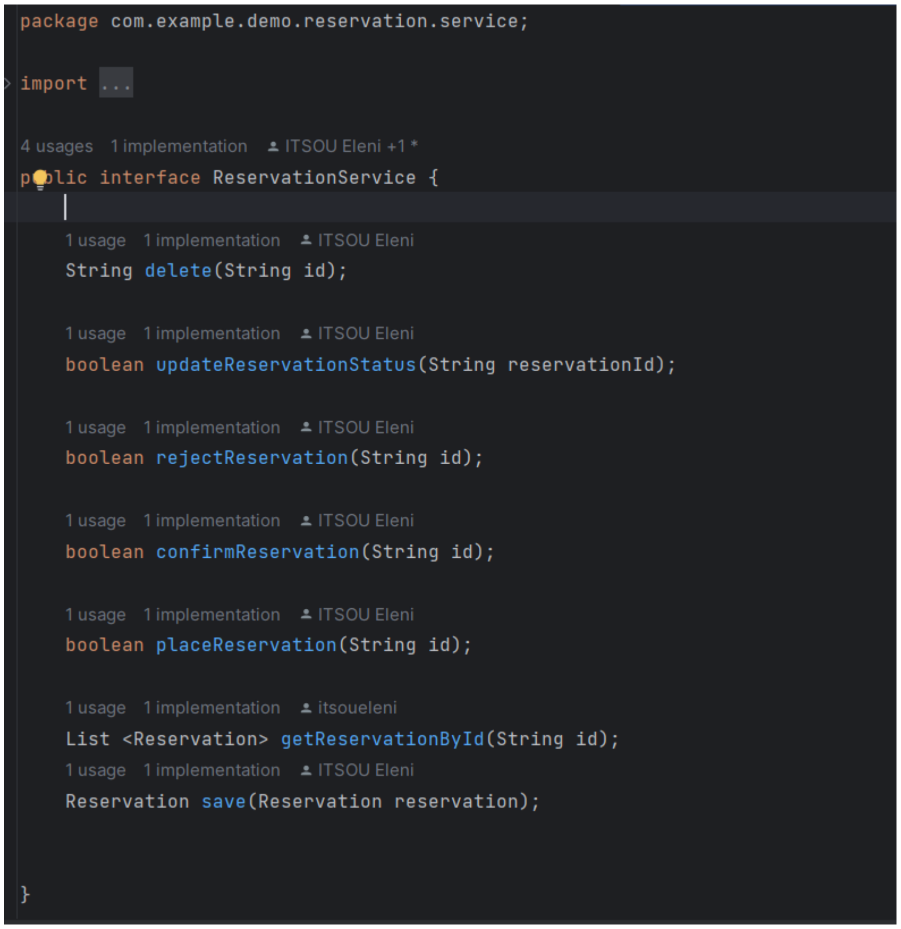
cluster0.nwkrohj.mongodb.net

Figure 45: Database connection via MongoDB Compass

5.2.1 Create a new Reservation Operation

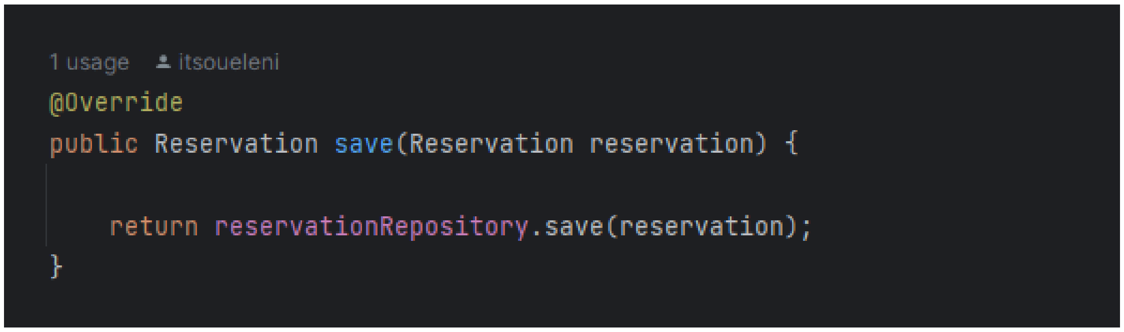
For creating a new reservation, the endpoint /reservation is specified, where clients can send a POST request using JSON data that represents a reservation. The creation method in ServiceImpl class in the service package oversees saving a new reservation in the Mongo DB database with the use of the Reservation repository. More specifically, by using the save method, the reservation object is saved inside the reservation repository. @Override annotation, specifies that a method is being overwritten from the Service interface (Figure 46), (

Figure *47*)



```
package com.example.demo.reservation.service;

import ...

4 usages   1 implementation   ITSOU Eleni +1 *
public interface ReservationService {

    1 usage   1 implementation   ITSOU Eleni
    String delete(String id);

    1 usage   1 implementation   ITSOU Eleni
    boolean updateReservationStatus(String reservationId);

    1 usage   1 implementation   ITSOU Eleni
    boolean rejectReservation(String id);

    1 usage   1 implementation   ITSOU Eleni
    boolean confirmReservation(String id);

    1 usage   1 implementation   ITSOU Eleni
    boolean placeReservation(String id);

    1 usage   1 implementation   itsoueleni
    List <Reservation> getReservationById(String id);
    1 usage   1 implementation   ITSOU Eleni
    Reservation save(Reservation reservation);


}
```

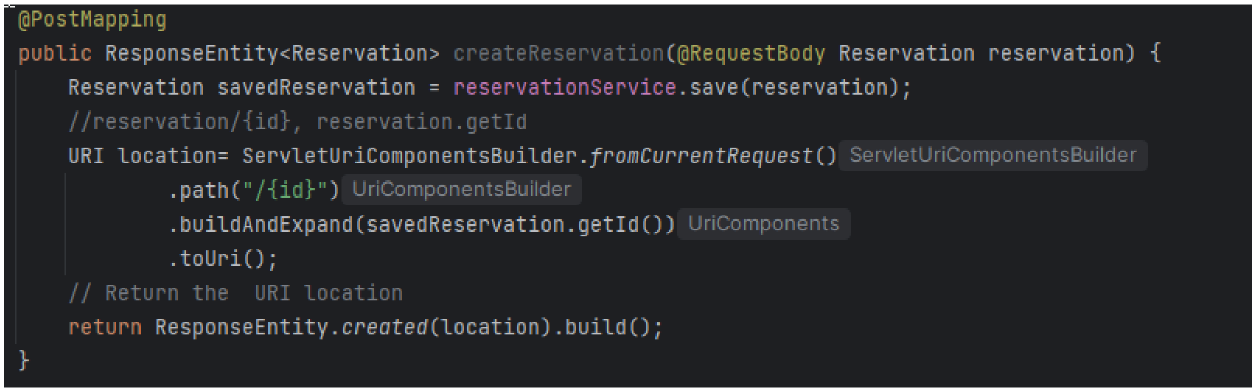
Figure 46: Methods defined in the service interface.



```
1 usage   itsoueleni
@Override
public Reservation save(Reservation reservation) {

    return reservationRepository.save(reservation);
}
```

Figure 47: Create method implemented in the service layer.

The Controller layer handles the reservation creation using the Reservation service. As a result, an instance of the ReservationService is injected into the controller using @Autowired dependency injection, allowing the save method to persist the receiving reservation. ServletUriComponentsBuilder.fromCurrentRequest() is used to generate a URI for the reservation id generated. Finally, the Response entity is returned, having a 201-response status (created), with the newly created URI included in the response headers.

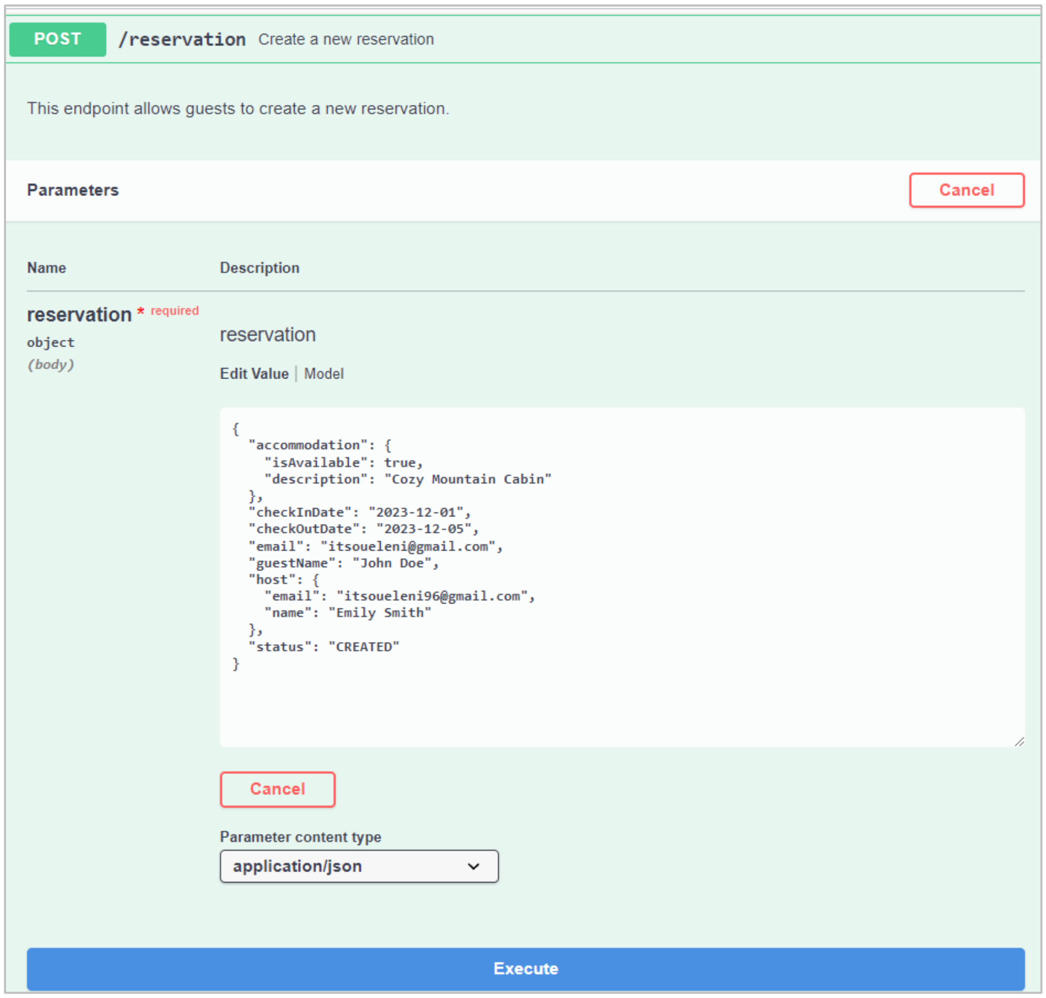


```java
@PostMapping
public ResponseEntity<Reservation> createReservation(@RequestBody Reservation reservation) {
    Reservation savedReservation = reservationService.save(reservation);
    //reservation/{id}, reservation.getId
    URI location= ServletUriComponentsBuilder.fromCurrentRequest() ServletUriComponentsBuilder
            .path("/{id}") UriComponentsBuilder
            .buildAndExpand(savedReservation.getId()) UriComponents
            .toUri();
    // Return the  URI location
    return ResponseEntity.created(location).build();
}
```

Figure 48: POST method as defined in the Reservation Controller class.

We can test the POST /reservation endpoint by using the Swagger UI and send a request. This is done by clicking the 'Trying it out' button. At this point, we can provide input data in the request body and then click the 'Execute' button Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code '201' is returned as well as the expected response body. In the response headers the URI of the reservation id is retrieved.

After the response is retrieved, we can also check that the reservation request is created successfully in the MongoDB database.

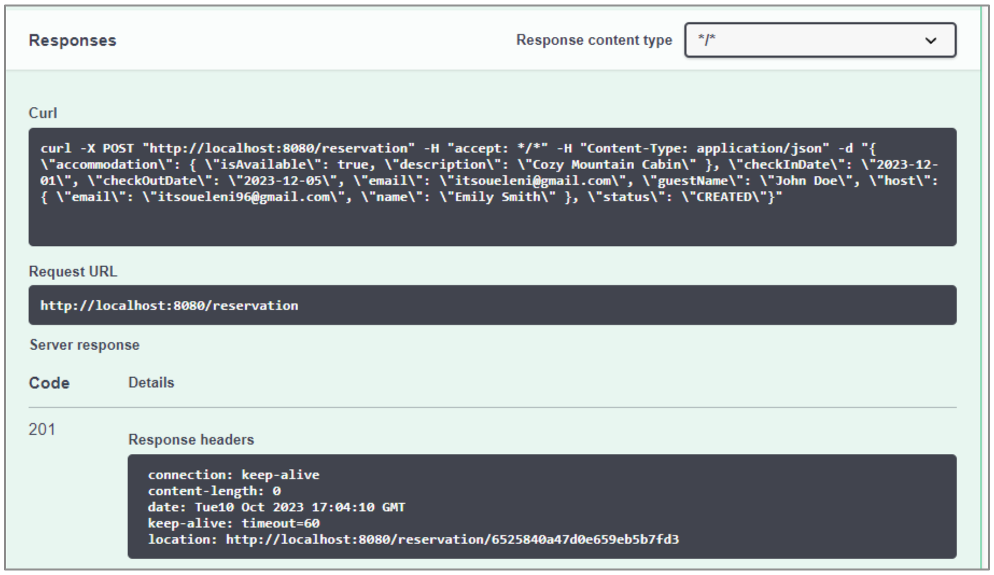


This endpoint allows guests to create a new reservation.

```
{
  "accommodation": {
    "isAvailable": true,
    "description": "Cozy Mountain Cabin"
  },
  "checkInDate": "2023-12-01",
  "checkOutDate": "2023-12-05",
  "email": "itsoueleni@gmail.com",
  "guestName": "John Doe",
  "host": {
    "email": "itsoueleni96@gmail.com",
    "name": "Emily Smith"
  },
  "status": "CREATED"
}
```

Parameter content type: application/json

Figure 49: Test of the POST create reservation method.



Response content type: */*

Curl

```
curl -X POST "http://localhost:8080/reservation" -H "accept: */*" -H "Content-Type: application/json" -d "{
\"accommodation\": { \"isAvailable\": true, \"description\": \"Cozy Mountain Cabin\" }, \"checkInDate\": \"2023-12-
01\", \"checkOutDate\": \"2023-12-05\", \"email\": \"itsoueleni@gmail.com\", \"guestName\": \"John Doe\", \"host\":
{ \"email\": \"itsoueleni96@gmail.com\", \"name\": \"Emily Smith\" }, \"status\": \"CREATED\"}"
```

Request URL

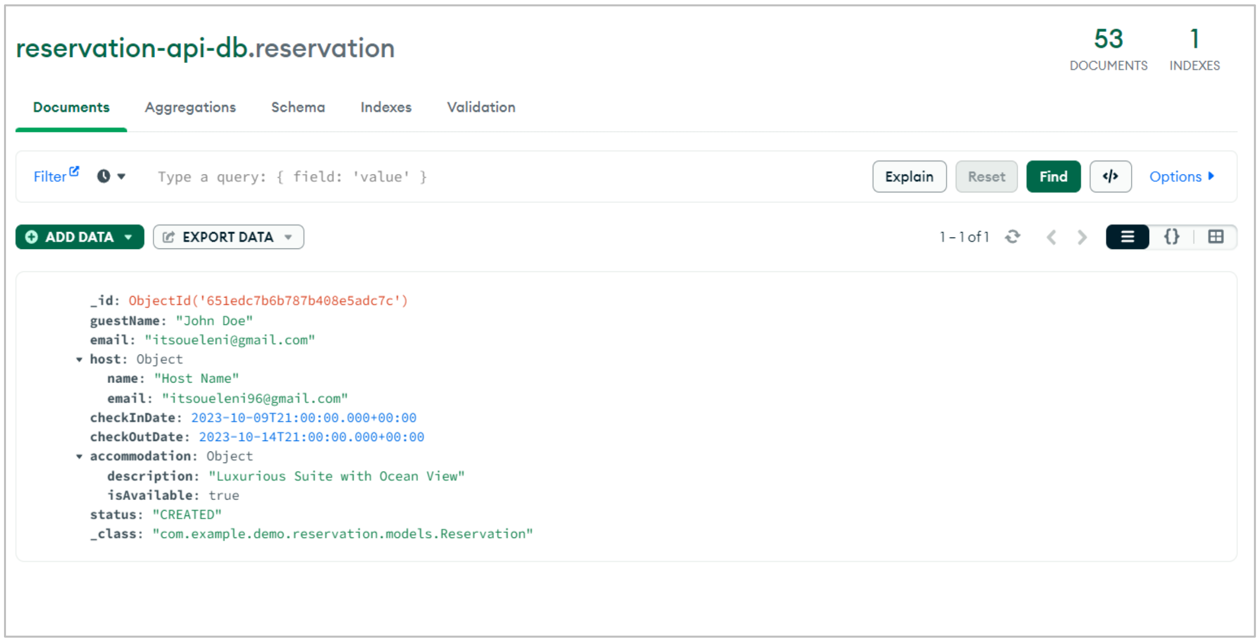
```
http://localhost:8080/reservation
```

Server response

Code	Details
201	Response headers

```
connection: keep-alive
content-length: 0
date: Tue10 Oct 2023 17:04:10 GMT
keep-alive: timeout=60
location: http://localhost:8080/reservation/6525840a47d0e659eb5b7fd3
```

Figure 50: Test of the POST create reservation method using Swagger UI- Check response status code and response body.



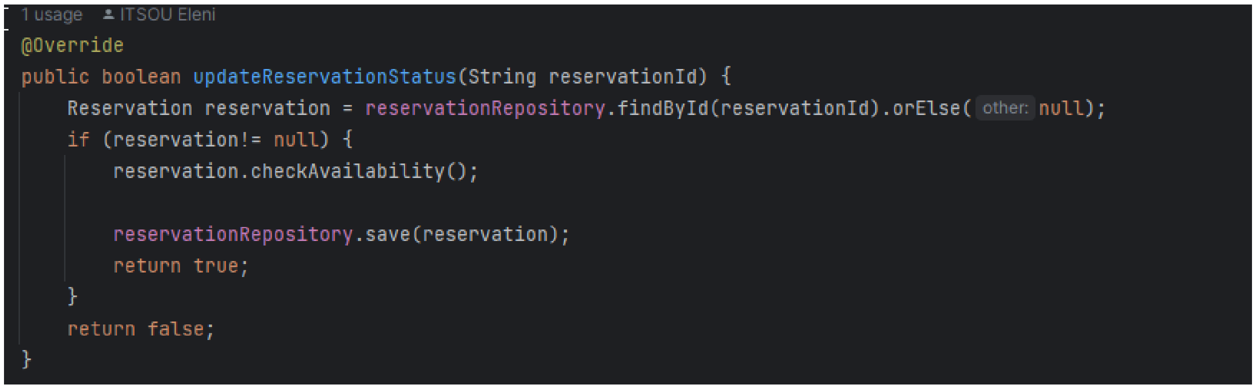
```
_id: ObjectId('651edc7b6b787b408e5adc7c')
guestName: "John Doe"
email: "itsoueleni@gmail.com"
▾ host: Object
   name: "Host Name"
   email: "itsoueleni96@gmail.com"
checkInDate: 2023-10-09T21:00:00.000+00:00
checkOutDate: 2023-10-14T21:00:00.000+00:00
▾ accommodation: Object
   description: "Luxurious Suite with Ocean View"
   isAvailable: true
status: "CREATED"
_class: "com.example.demo.reservation.models.Reservation"
```

Figure 51: Test of the POST create reservation method using Swagger UI- The reservation request is placed successfully in the database.

5.2.2 Update the Reservation Status

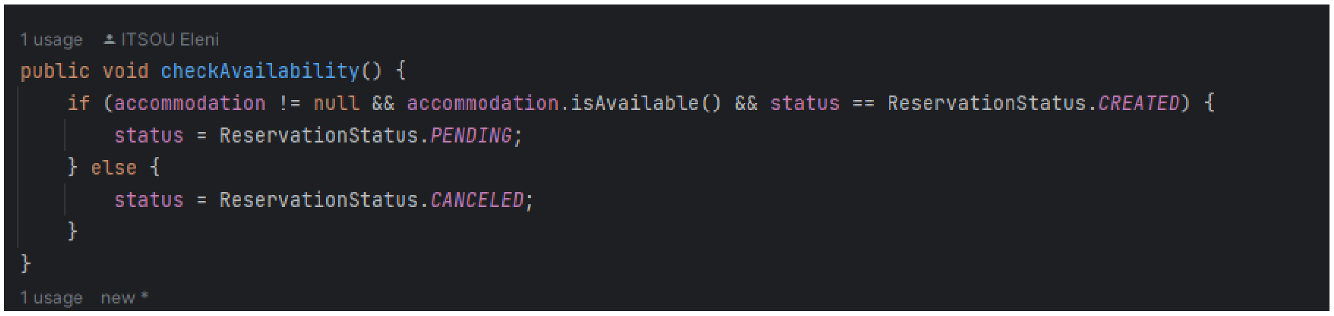
The updateReservationStatus method determines whether a reservation with the provided ID exists in the database. Depending on the availability of the ID, the update operation can change the status to 'PENDING' or if not available to 'CANCELLED'. This function is mapped to a URL with the use of @PutMapping annotation which allows the update of the reservation status using an HTTP PUT request.

UpdateReservationStatus method is critical in handling reservation statuses in the ReservationService implementation class (Figure 52). This method expects the reservationId, and it starts when attempting to find the corresponding id in the repository layer. When a reservation is discovered, it is stored as a variable. Following that, the method invokes check availability method from the reservation object. This includes the logic of determining the availability of the reservation and updating the reservation status accordingly. By using the reservationRepository.save(reservation) method, amended reservation object is saved to the repository after the reservation status has been suitably adjusted.



```
1 usage  ITSOU Eleni
@Override
public boolean updateReservationStatus(String reservationId) {
    Reservation reservation = reservationRepository.findById(reservationId).orElse( other: null);
    if (reservation!= null) {
        reservation.checkAvailability();

        reservationRepository.save(reservation);
        return true;
    }
    return false;
}
```

Figure 52: UpdateReservationStatus Method in ServiceImpl class

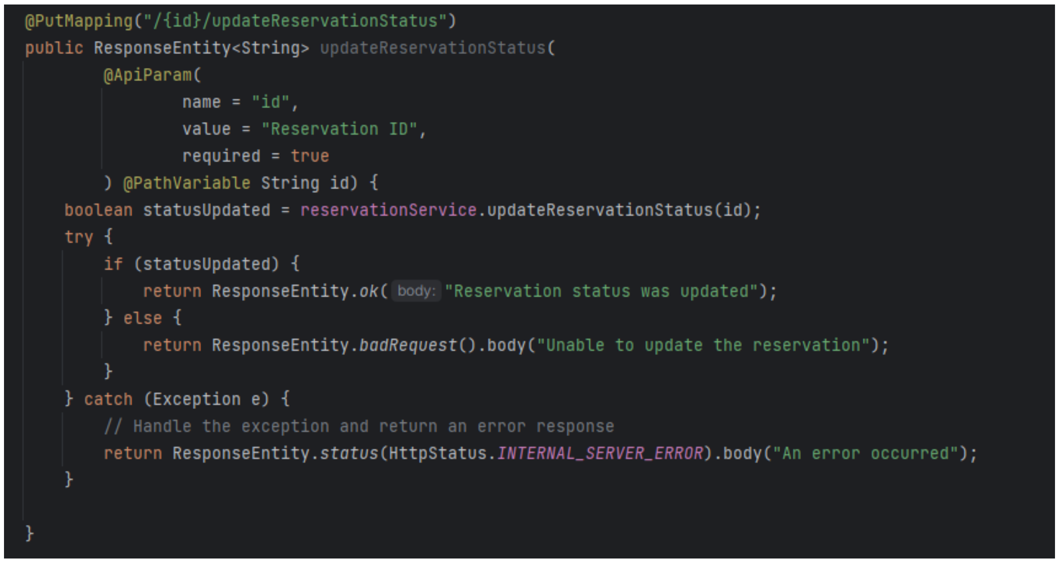


```
1 usage  ITSOU Eleni
public void checkAvailability() {
    if (accommodation != null && accommodation.isAvailable() && status == ReservationStatus.CREATED) {
        status = ReservationStatus.PENDING;
    } else {
        status = ReservationStatus.CANCELED;
    }
}
1 usage  new *
```

Figure 53: Check Availability method in Reservation object

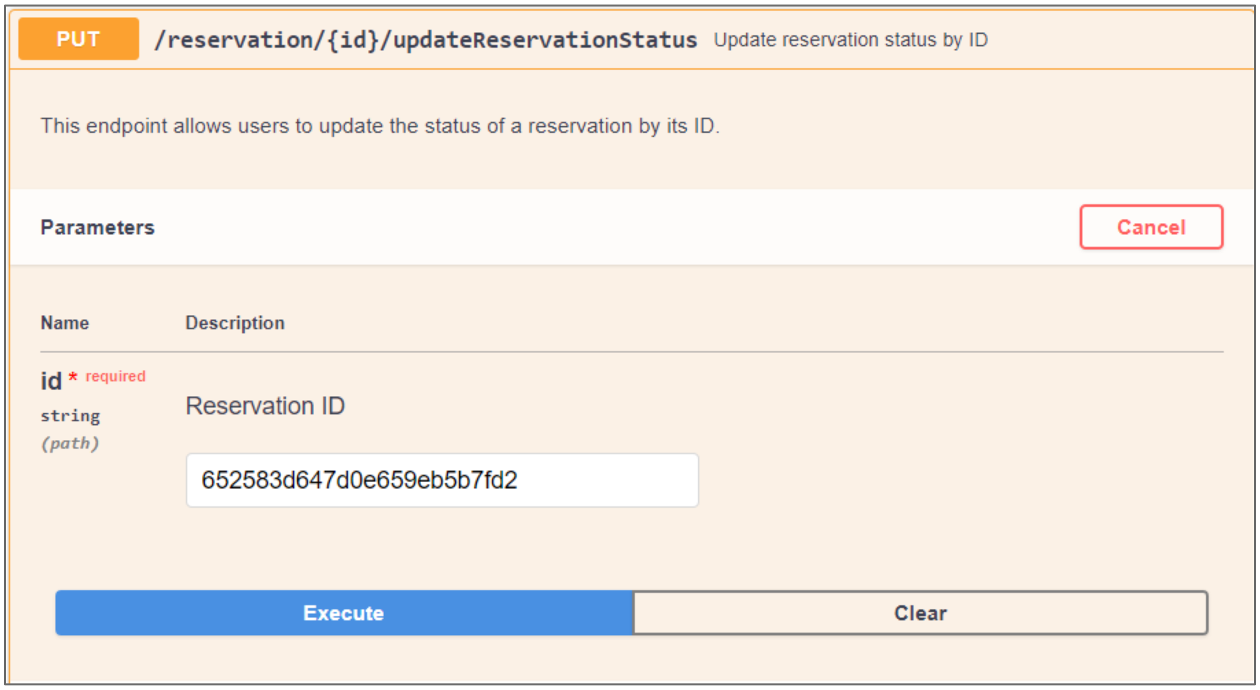
This functionality is available in the controller using the @PutMapping annotation waiting for PUT HTTP requests at /id/updateReservationStatus. The id variable which is a component of the URL represents the reservation id of the reservation object. When a PUT request is received then the updateReservation method is called for each specific reservation id. Then with the reservationService.updateReservationStatus(id) the service layer logic is activated.

The Boolean variable statusUpdated is utilized within the method to determine whether the status of the reservation is updated. If the update id successful, then a response entity is returned with HTTP status 200. Else, a bad request HTTP status is retrieved, and the reservation is not updated. To handle exceptions, a try-catch block is used. The exception is handled by an HTTP status 500 response and an appropriate error message is displayed.



```java
@PutMapping("/{id}/updateReservationStatus")
public ResponseEntity<String> updateReservationStatus(
        @ApiParam(
                name = "id",
                value = "Reservation ID",
                required = true
        ) @PathVariable String id) {
    boolean statusUpdated = reservationService.updateReservationStatus(id);
    try {
        if (statusUpdated) {
            return ResponseEntity.ok( body: "Reservation status was updated");
        } else {
            return ResponseEntity.badRequest().body("Unable to update the reservation");
        }
    } catch (Exception e) {
        // Handle the exception and return an error response
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred");
    }

}
```

Figure 54: PUT update reservation status method as defined in the Reservation Controller class.



PUT /reservation/{id}/updateReservationStatus Update reservation status by ID

This endpoint allows users to update the status of a reservation by its ID.

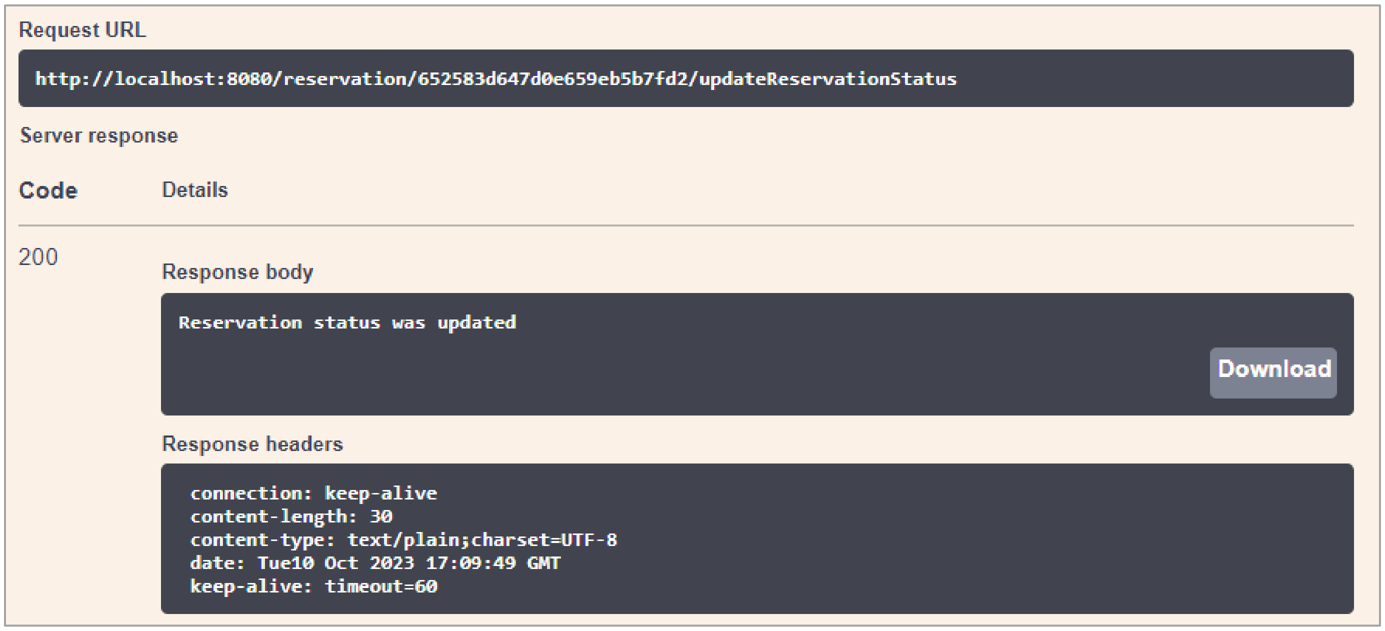
Parameters

Name	Description
id * required string (path)	Reservation ID 652583d647d0e659eb5b7fd2

Figure 55: Test of the PUT update reservation status method using Swagger UI-Provide input JSON data.

We can test the PUT /reservation/id/updateReservationStatus endpoint by using the Swagger UI and send a request. This is done by clicking the 'Trying it out' button. At this point, we can provide the reservationId and click the 'Execute' button (Figure 55). Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code '200' is returned as well as the expected response body (Figure 56).

After the response is retrieved, we can also check that the reservation status of this specific reservation id has been updated successfully in the MongoDB database.



Request URL

```
http://localhost:8080/reservation/652583d647d0e659eb5b7fd2/updateReservationStatus
```

Server response

Code | Details

200

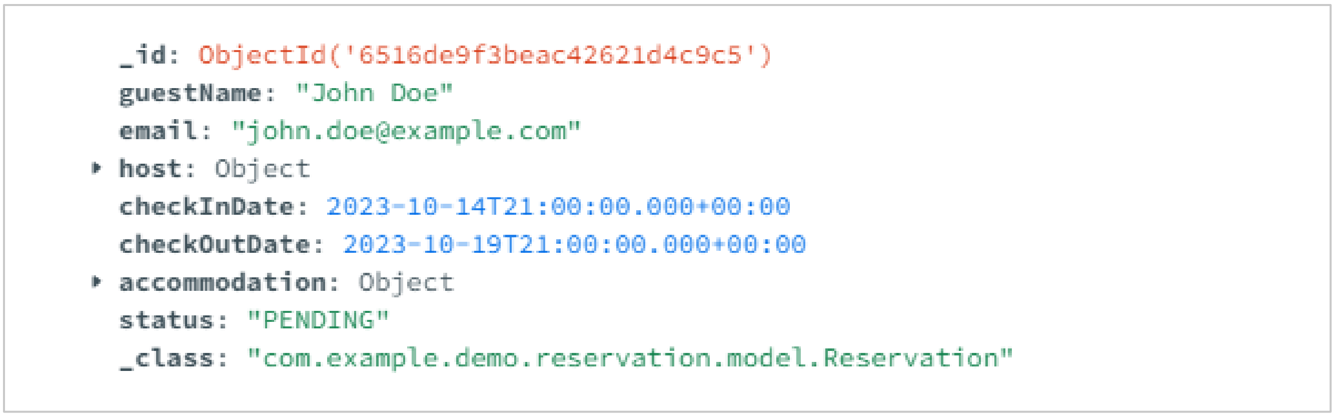
Response body

```
Reservation status was updated
```

Response headers

```
connection: keep-alive
content-length: 30
content-type: text/plain;charset=UTF-8
date: Tue10 Oct 2023 17:09:49 GMT
keep-alive: timeout=60
```

Figure 56: Test of the PUT update reservation status method using Swagger UI- Check response status code and response body.

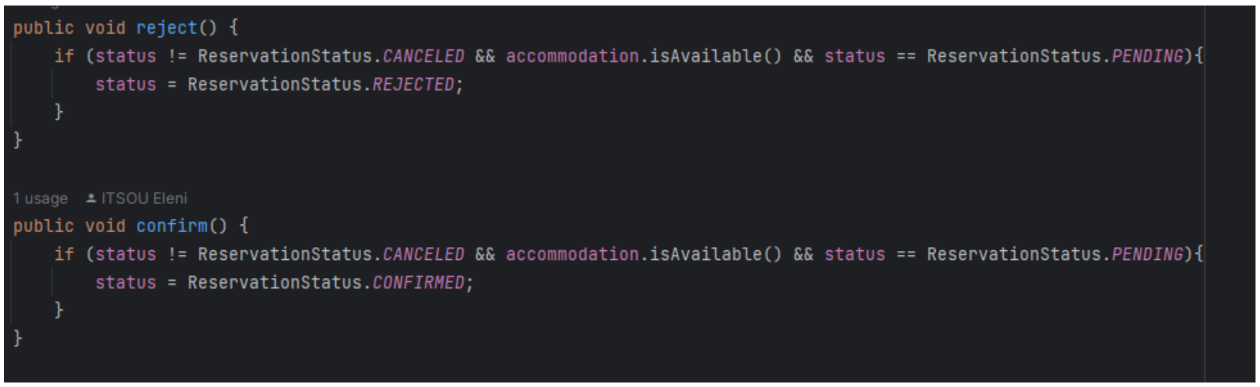


```
  _id: ObjectId('6516de9f3beac42621d4c9c5')
  guestName: "John Doe"
  email: "john.doe@example.com"
▸ host: Object
  checkInDate: 2023-10-14T21:00:00.000+00:00
  checkOutDate: 2023-10-19T21:00:00.000+00:00
▸ accommodation: Object
  status: "PENDING"
  _class: "com.example.demo.reservation.model.Reservation"
```

Figure 57: Test of the PUT update reservation status method using Swagger UI- The reservation request has been updated in the database.

5.2.3 Confirm or Reject a Reservation Request

Two methods are described in the ReservationService interface, the rejectReservation and the confirmReservation, that handle rejection and confirmation of a reservation by the host respectively. By taking the reservationId, these methods fetch the reservation from the repository and update the status if the conditions are met (Figure 59). More specifically, if a reservation is found in state 'PENDING' 'reject' or 'confirm' methods are called from the reservation object and update the status in 'CONFIRMED' or 'REJECTED' (Figure 58). Then the object is again saved in the repository.



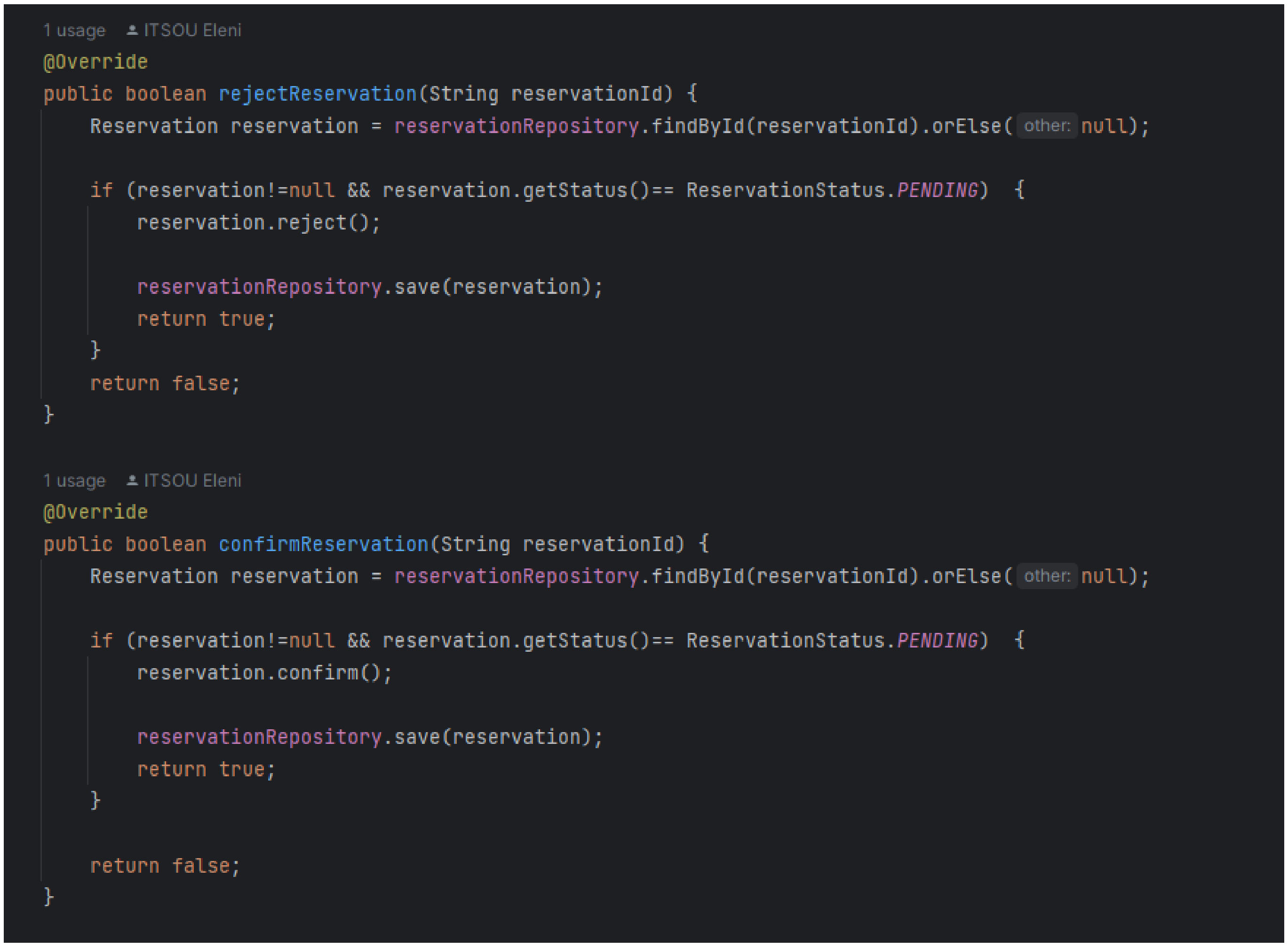
```
public void reject() {
    if (status != ReservationStatus.CANCELED && accommodation.isAvailable() && status == ReservationStatus.PENDING){
        status = ReservationStatus.REJECTED;
    }
}

1 usage   ITSOU Eleni
public void confirm() {
    if (status != ReservationStatus.CANCELED && accommodation.isAvailable() && status == ReservationStatus.PENDING){
        status = ReservationStatus.CONFIRMED;
    }
}
```

Figure 58: Confirm and reject methods in the reservation object.

The rejectReservation and confirmReservation annotated PUT methods on the controller handle the routes, /id/rejectReservation and /id/confirmReservation respectively (Figure 60). Like the update of the reservation status PUT method, the id variable which is a component of both URLs represents the reservation id of the reservation object, and the reservationServices's relevant functions are called. If the operation is successful, then a HTTP 200 OK status code is retrieved or else a 400-error response is delivered. Error handling is also used for catching unexpected issues and an HTTP 500 status is returned. This ensures that reservation rejection and confirmation actions are provided via RESTful APIs allowing the host to update the reservation status with suitable replies and indicating whether the operations were successful or unsuccessful.

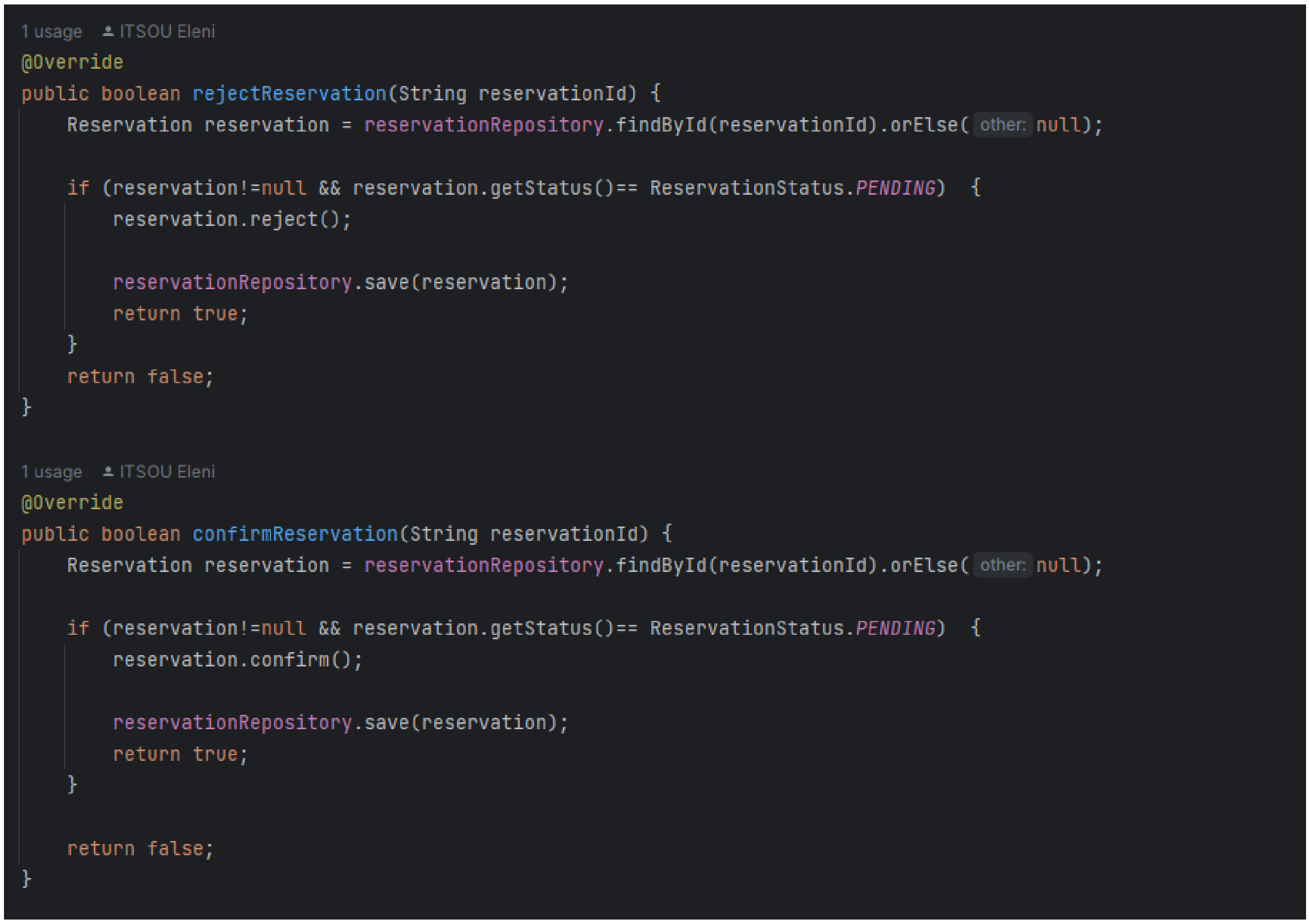
We can test the PUT /reservation/id/confirmReservation endpoint by using the Swagger UI and sending a request. This is done by clicking the 'Trying it out' button. At this point, we can provide the reservationId and click the 'Execute' button (Figure 61). Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code ‘200’ is returned as well as the expected response body (Figure 62).

After the response is retrieved, we can also check that the reservation status of this specific reservationId has been updated successfully in the MongoDB database (Figure 63).



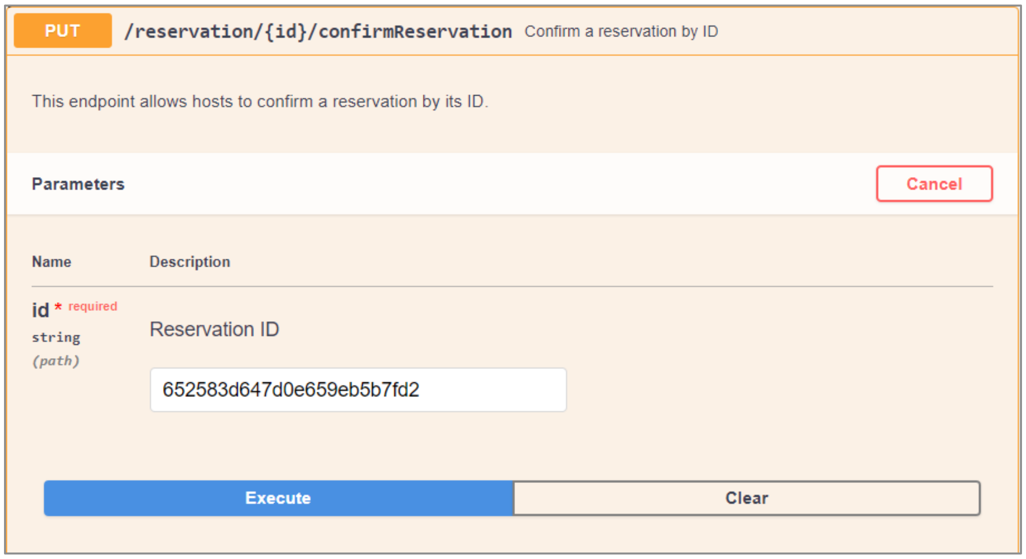
```java
1 usage   ITSOU Eleni
@Override
public boolean rejectReservation(String reservationId) {
    Reservation reservation = reservationRepository.findById(reservationId).orElse( other: null);

    if (reservation!=null && reservation.getStatus()== ReservationStatus.PENDING)  {
        reservation.reject();

        reservationRepository.save(reservation);
        return true;
    }
    return false;
}

1 usage   ITSOU Eleni
@Override
public boolean confirmReservation(String reservationId) {
    Reservation reservation = reservationRepository.findById(reservationId).orElse( other: null);

    if (reservation!=null && reservation.getStatus()== ReservationStatus.PENDING)  {
        reservation.confirm();

        reservationRepository.save(reservation);
        return true;
    }

    return false;
}
```

Figure 59: Reject and Confirm reservation methods in ServiceImpl class



```
1 usage   ITSOU Eleni
@Override
public boolean rejectReservation(String reservationId) {
    Reservation reservation = reservationRepository.findById(reservationId).orElse( other: null);

    if (reservation!=null && reservation.getStatus()== ReservationStatus.PENDING)  {
        reservation.reject();

        reservationRepository.save(reservation);
        return true;
    }
    return false;
}

1 usage   ITSOU Eleni
@Override
public boolean confirmReservation(String reservationId) {
    Reservation reservation = reservationRepository.findById(reservationId).orElse( other: null);

    if (reservation!=null && reservation.getStatus()== ReservationStatus.PENDING)  {
        reservation.confirm();

        reservationRepository.save(reservation);
        return true;
    }

    return false;
}
```

Figure 60: PUT reject and confirm reservation methods as defined in the ReservationController class.



PUT /reservation/{id}/confirmReservation Confirm a reservation by ID

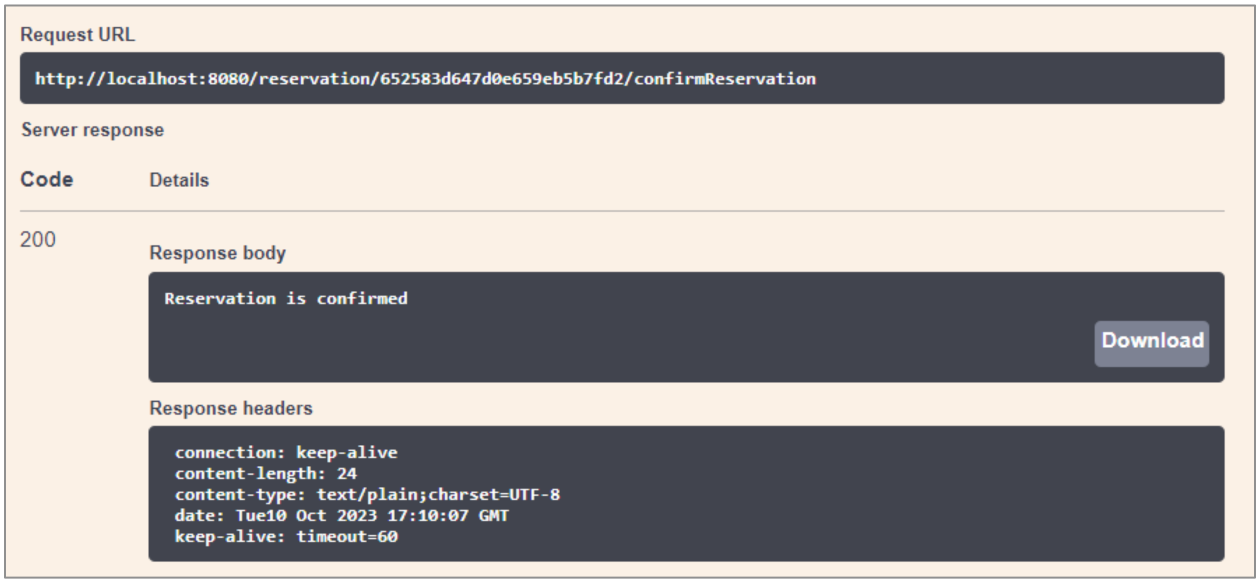
This endpoint allows hosts to confirm a reservation by its ID.

Parameters — Cancel

Name	Description
id * required string (path)	Reservation ID 652583d647d0e659eb5b7fd2

Execute — Clear

Figure 61: Test of the PUT confirm reservation method using Swagger UI-Provide input JSON data.


Request URL

```
http://localhost:8080/reservation/652583d647d0e659eb5b7fd2/confirmReservation
```

Server response

Code Details

200

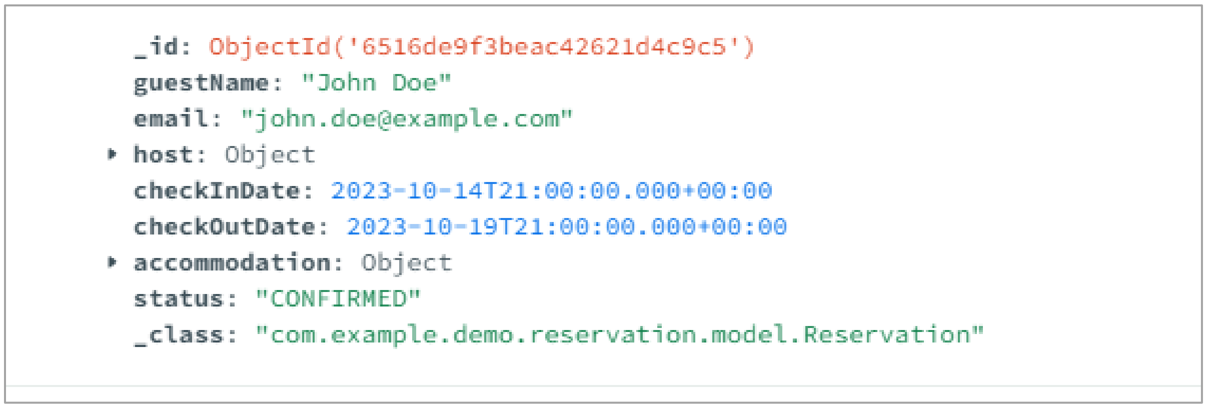
Response body

```
Reservation is confirmed
```

Response headers

```
connection: keep-alive
content-length: 24
content-type: text/plain;charset=UTF-8
date: Tue10 Oct 2023 17:10:07 GMT
keep-alive: timeout=60
```

Figure 62:Test of the PUT confirm reservation method using Swagger UI- Check response status code and response body.

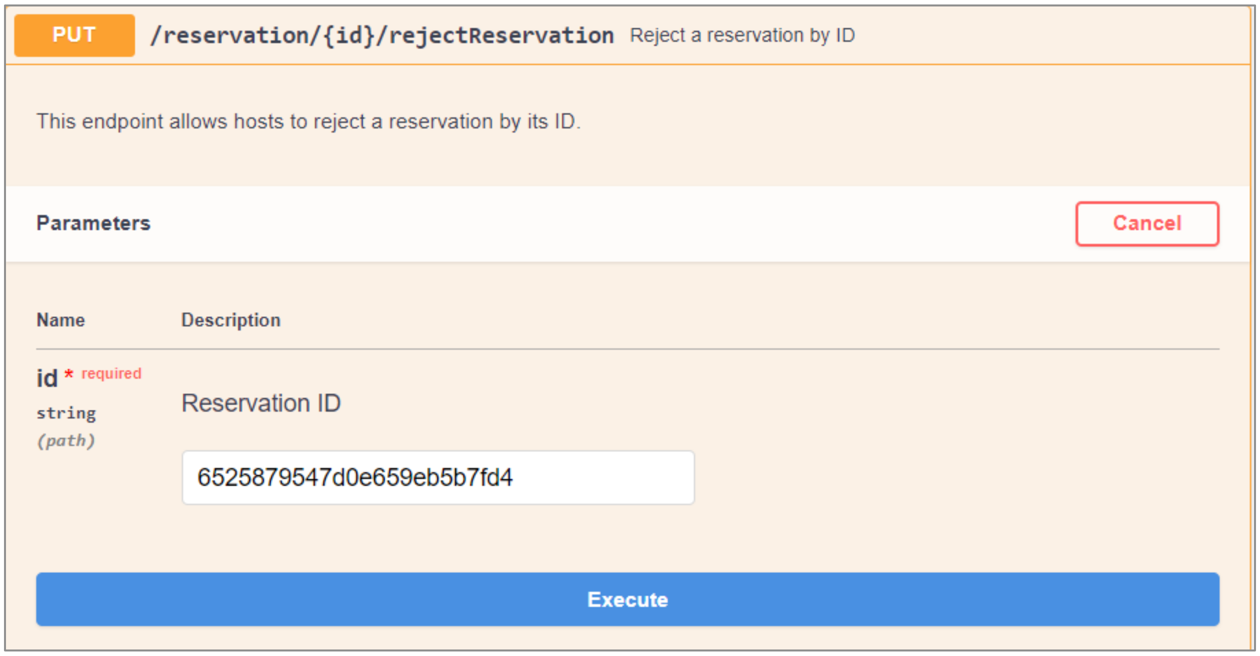


```
_id: ObjectId('6516de9f3beac42621d4c9c5')
guestName: "John Doe"
email: "john.doe@example.com"
▸ host: Object
checkInDate: 2023-10-14T21:00:00.000+00:00
checkOutDate: 2023-10-19T21:00:00.000+00:00
▸ accommodation: Object
status: "CONFIRMED"
_class: "com.example.demo.reservation.model.Reservation"
```

Figure 63:Test of the PUT confirm reservation status method using Swagger UI- The reservation request has been updated in the database.

Subsequently we can test the PUT /reservation/id/rejectReservation endpoint by using the Swagger UI and sending a request. This is done by clicking the 'Trying it out' button. At this point, we can provide the reservationId and click the 'Execute' button (Figure 64). Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code '200' is returned as well as the expected response body (Figure 65).

After the response is retrieved, we can also check that the reservation status of this specific reservationId has been updated successfully in the MongoDB database (Figure 66).

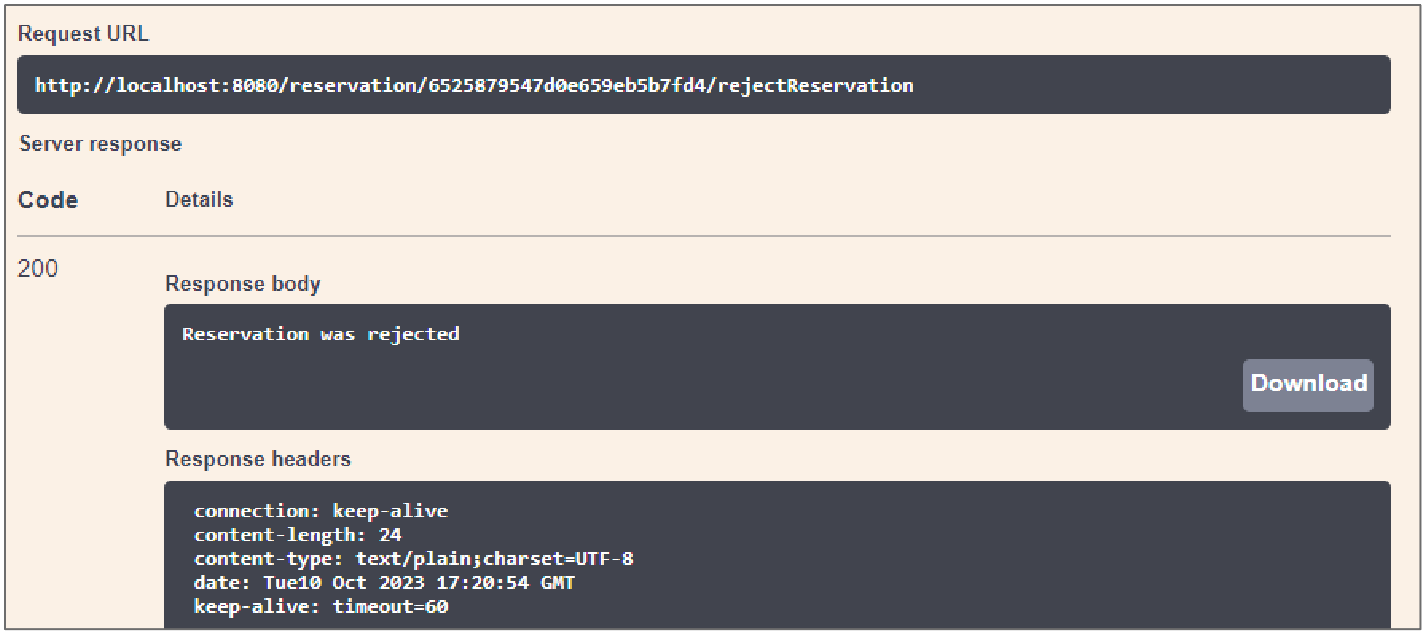

PUT `/reservation/{id}/rejectReservation` Reject a reservation by ID

This endpoint allows hosts to reject a reservation by its ID.

Parameters

Name	Description
id * required string *(path)*	Reservation ID 6525879547d0e659eb5b7fd4

Figure 64: Test of the PUT reject reservation method using Swagger UI-Provide input JSON data.


Request URL

```
http://localhost:8080/reservation/6525879547d0e659eb5b7fd4/rejectReservation
```

Server response

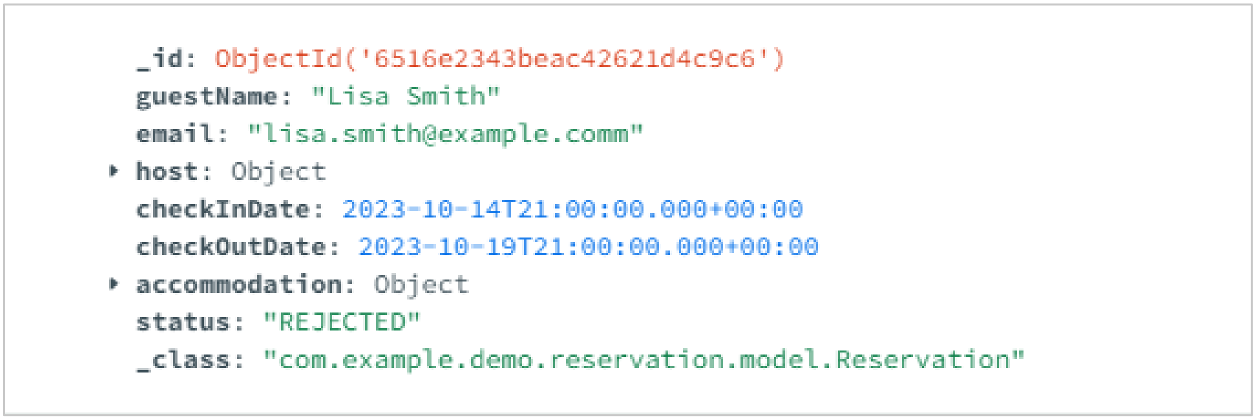
Code	Details
200	Response body

```
Reservation was rejected
```

Response headers

```
connection: keep-alive
content-length: 24
content-type: text/plain;charset=UTF-8
date: Tue10 Oct 2023 17:20:54 GMT
keep-alive: timeout=60
```

Figure 65: Test of the PUT reject reservation method using Swagger UI- Check response status code and response body.

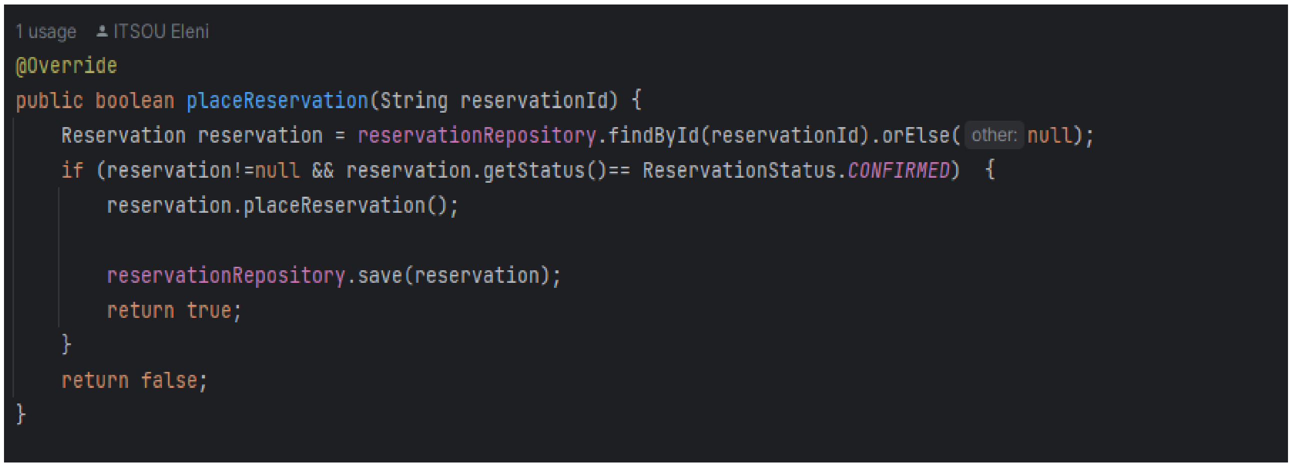


```
  _id: ObjectId('6516e2343beac42621d4c9c6')
  guestName: "Lisa Smith"
  email: "lisa.smith@example.comm"
▸ host: Object
  checkInDate: 2023-10-14T21:00:00.000+00:00
  checkOutDate: 2023-10-19T21:00:00.000+00:00
▸ accommodation: Object
  status: "REJECTED"
  _class: "com.example.demo.reservation.model.Reservation"
```

Figure 66: Test of the PUT reject reservation status method using Swagger UI- The reservation request has been updated in the database.

5.2.4 Place a new Reservation Operation

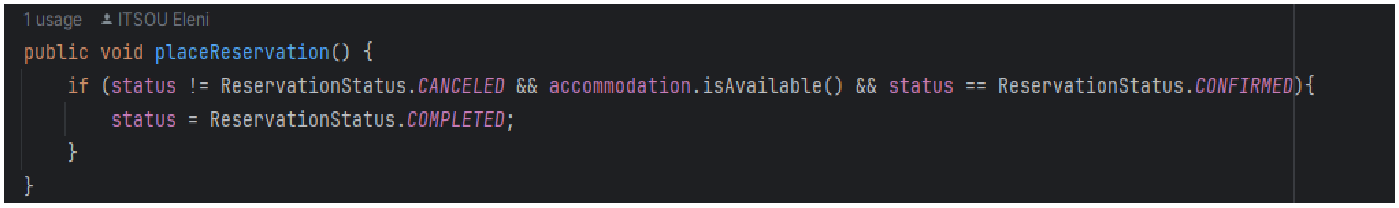
For placing a reservation method in the database, the placeReservation method is constructed in the service layer. More specifically, it is utilized to update the status of a reservation found (Figure 67). If the reservation status is 'CONFIRMED' the placeReservation method is called from the reservation object (Figure 68) and the status changes to 'COMPLETED. After that the object is saved again in the repository.



```
1 usage  ITSOU Eleni
@Override
public boolean placeReservation(String reservationId) {
    Reservation reservation = reservationRepository.findById(reservationId).orElse( other: null);
    if (reservation!=null && reservation.getStatus()== ReservationStatus.CONFIRMED)  {
        reservation.placeReservation();

        reservationRepository.save(reservation);
        return true;
    }
    return false;
}
```

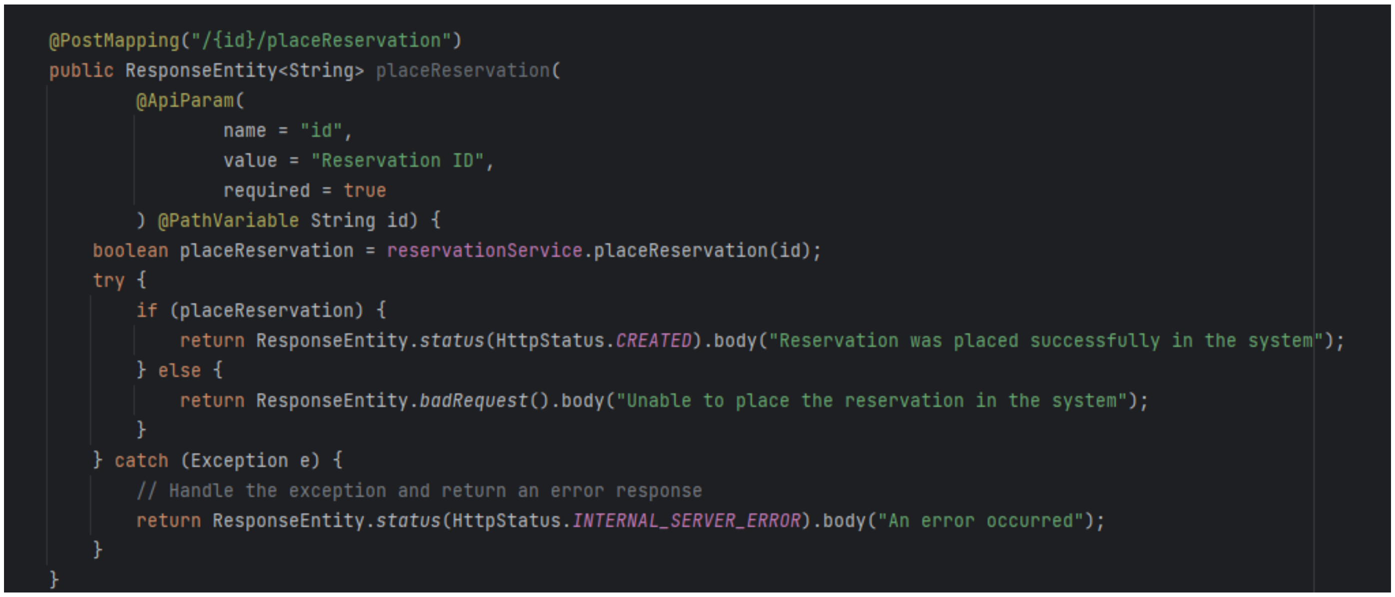
Figure 67: Place reservation method in ServiceImpl class

In the controller, HTTP POST requests to the path /id/plcaeReservation are handled. The id is taken from the URL and the reservationService's placeReservationMethod is called. Then the reservation is successfully placed, HTTP 201 response code is returned. Else a 400-error response is delivered. There is also an exception handling for unforeseen errors where a HTTP 500 error response is returned.



```
1 usage  ITSOU Eleni
public void placeReservation() {
    if (status != ReservationStatus.CANCELED && accommodation.isAvailable() && status == ReservationStatus.CONFIRMED){
        status = ReservationStatus.COMPLETED;
    }
}
```

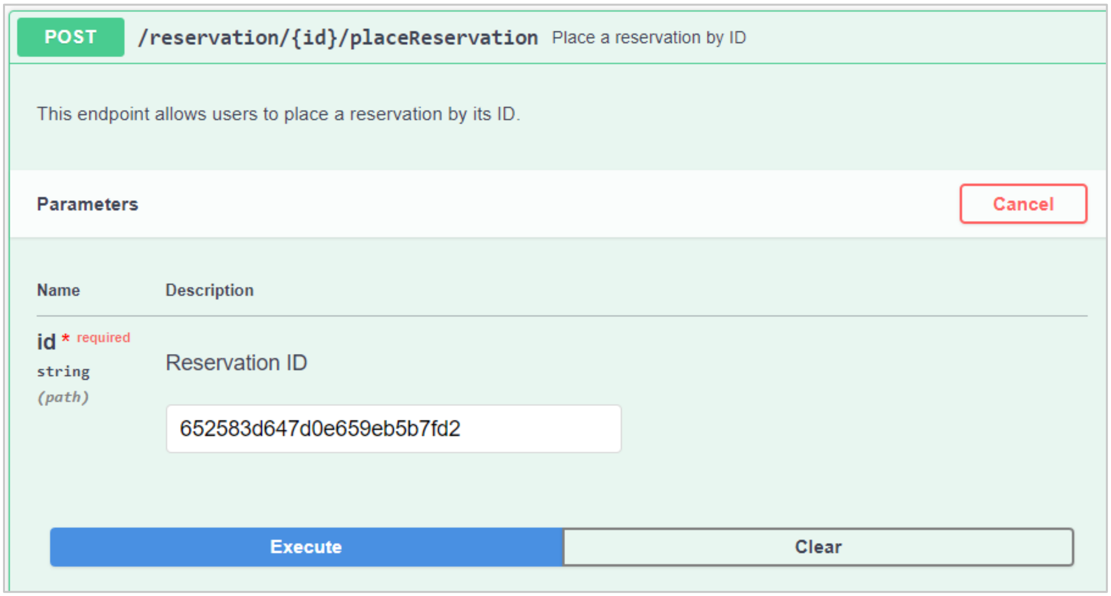
Figure 68: Place Reservation method in Reservation object



```
@PostMapping("/{id}/placeReservation")
public ResponseEntity<String> placeReservation(
        @ApiParam(
                name = "id",
                value = "Reservation ID",
                required = true
        ) @PathVariable String id) {
    boolean placeReservation = reservationService.placeReservation(id);
    try {
        if (placeReservation) {
            return ResponseEntity.status(HttpStatus.CREATED).body("Reservation was placed successfully in the system");
        } else {
            return ResponseEntity.badRequest().body("Unable to place the reservation in the system");
        }
    } catch (Exception e) {
        // Handle the exception and return an error response
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred");
    }
}
```

Figure 69: POST place reservation method as defined in the ReservationController class.

We can test the PUT /reservation/id/placeReservation endpoint by using the Swagger UI and sending a request. This is done by clicking the 'Trying it out' button. At this point, we can provide the reservationId and click the 'Execute' button (Figure 70) Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code '201' is returned as well as the expected response body (Figure 71).

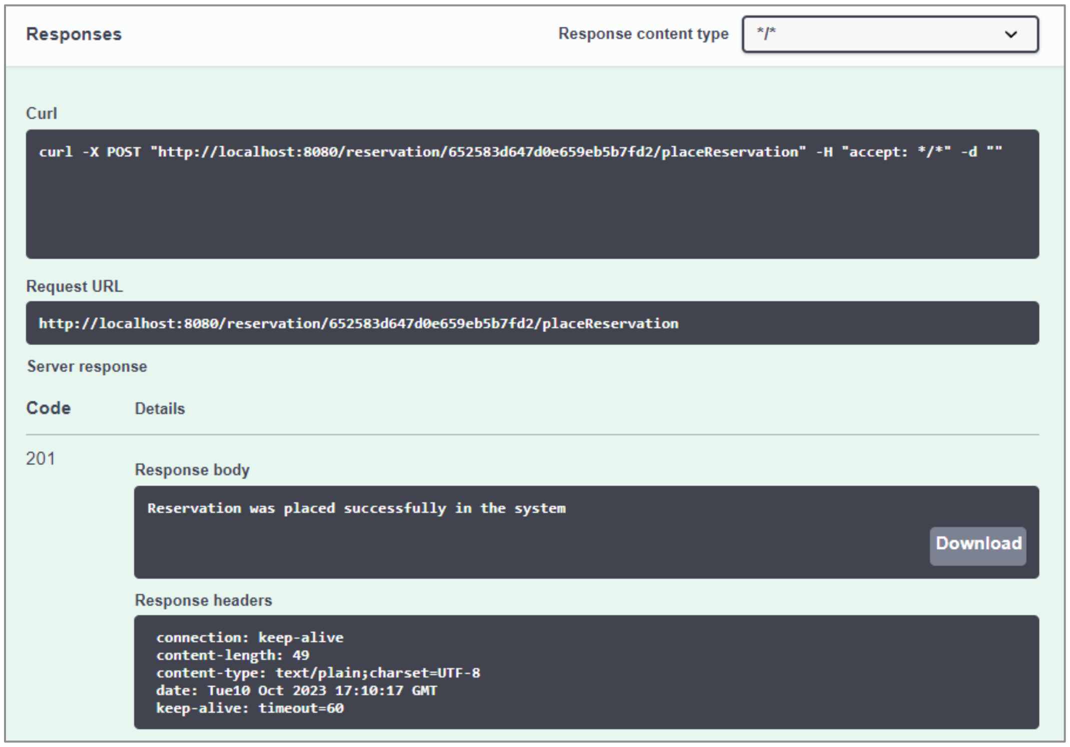


This endpoint allows users to place a reservation by its ID.

Reservation ID

652583d647d0e659eb5b7fd2

Figure 70: Test of the POST place reservation method using Swagger UI-Provide input JSON data.



```
curl -X POST "http://localhost:8080/reservation/652583d647d0e659eb5b7fd2/placeReservation" -H "accept: */*" -d ""
```

```
http://localhost:8080/reservation/652583d647d0e659eb5b7fd2/placeReservation
```

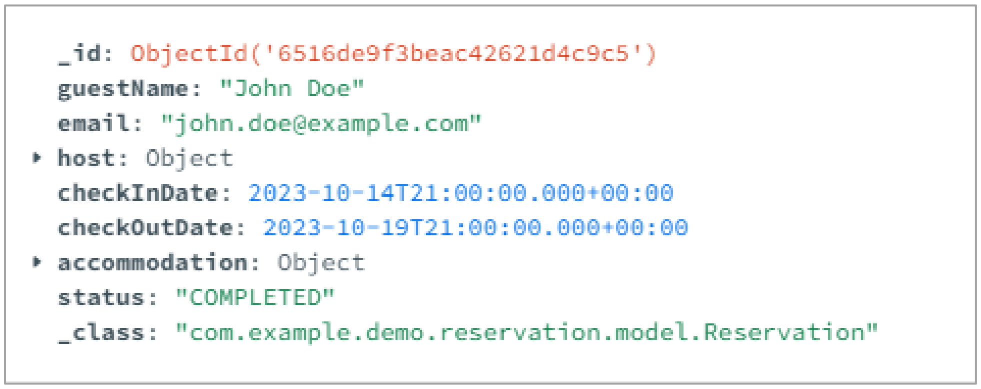
201

```
Reservation was placed successfully in the system
```

```
connection: keep-alive
content-length: 49
content-type: text/plain;charset=UTF-8
date: Tue10 Oct 2023 17:10:17 GMT
keep-alive: timeout=60
```

Figure 71: Test of the POST place reservation method using Swagger UI- Check response status code and response body.

After the response is retrieved, we can also check that the reservation status of this specific reservationId has been updated successfully in the MongoDB database (Figure 72).

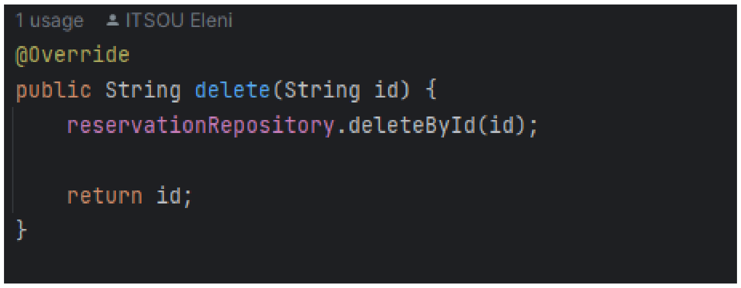


```
  _id: ObjectId('6516de9f3beac42621d4c9c5')
  guestName: "John Doe"
  email: "john.doe@example.com"
▸ host: Object
  checkInDate: 2023-10-14T21:00:00.000+00:00
  checkOutDate: 2023-10-19T21:00:00.000+00:00
▸ accommodation: Object
  status: "COMPLETED"
  _class: "com.example.demo.reservation.model.Reservation"
```

Figure 72: Test of the POST place reservation method using Swagger UI- The reservation status is updated successfully in the database.

5.2.5 Delete a Reservation Operation

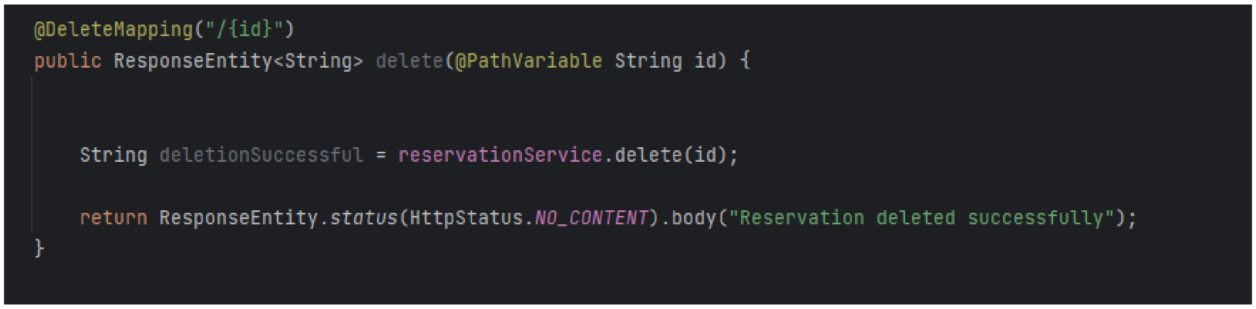
For deleting a reservation method in the database, the deleteReservation method is constructed in the service layer and corresponds to the @DeleteMApping function in the Controller. The deletedById method of the reservationRepository is used to remove the reservation from the specified id. The method returns the id of the reservation after the deletion (Figure 73).



```
1 usage   ITSOU Eleni
@Override
public String delete(String id) {
    reservationRepository.deleteById(id);

    return id;
}
```

Figure 73: Delete reservation method in ServiceImpl class.

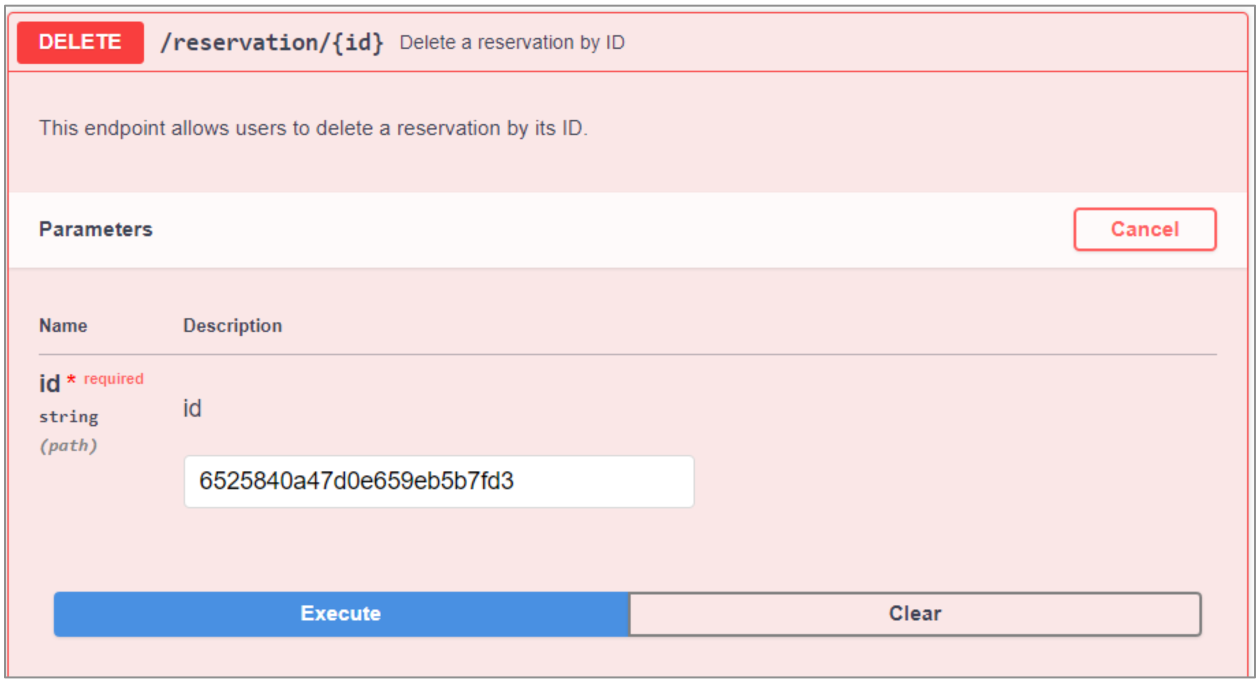
HTTP delete requests are handled by the controller method. The id is taken by the UR and the reservationServices's delete function is invoked to remove the reservation from the database. Following the deletion, a response object with the HTTP status no content is returned, indicating that the reservation was successfully deleted. These methods enable customers to erase reservations by sending HTTP erase requests and receiving an appropriate response that indicates the operation's success.



```java
@DeleteMapping("/{id}")
public ResponseEntity<String> delete(@PathVariable String id) {


    String deletionSuccessful = reservationService.delete(id);

    return ResponseEntity.status(HttpStatus.NO_CONTENT).body("Reservation deleted successfully");
}
```

Figure 74: DELETE, delete reservation method as defined in the Reservation Controller class.

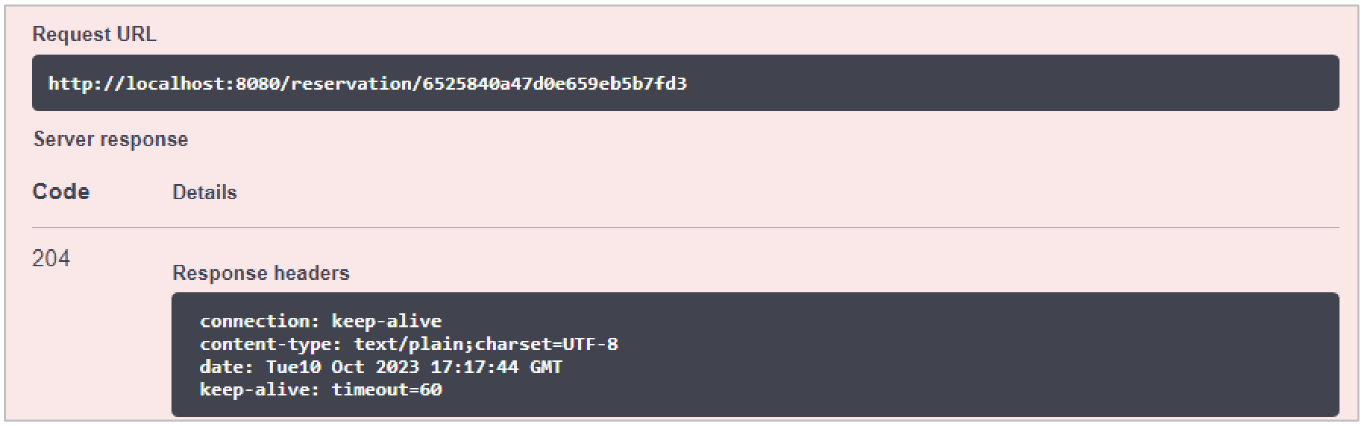
We can test the DELETE/reservation/id endpoint by using the Swagger UI and sending a request. This is done by clicking the 'Trying it out' button. At this point, we can provide the reservationId and click the 'Execute' button. Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code '204' is returned.



This endpoint allows users to delete a reservation by its ID.

Name	Description
id * required string (path)	id 6525840a47d0e659eb5b7fd3

Figure 75: Test of the DELETE reservation method using Swagger UI-Provide input JSON data.


Request URL

```
http://localhost:8080/reservation/6525840a47d0e659eb5b7fd3
```

Server response

Code Details

204

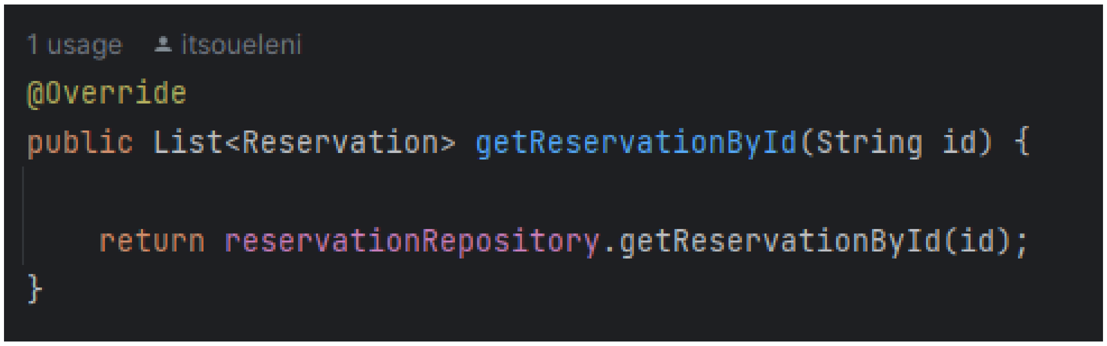
Response headers

```
connection: keep-alive
content-type: text/plain;charset=UTF-8
date: Tue10 Oct 2023 17:17:44 GMT
keep-alive: timeout=60
```

Figure 76: Test of the DELETE reservation method using Swagger UI- Check response status code and response body.

5.2.6 Get reservation by identification number.

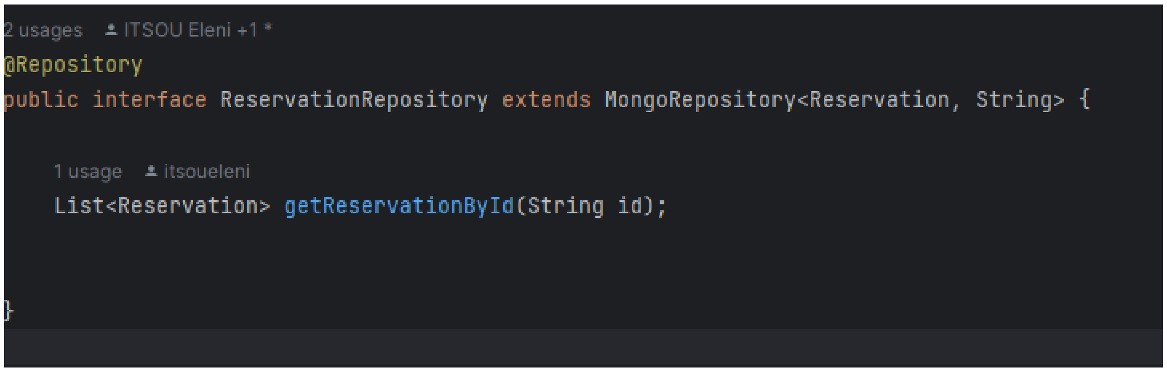
To get a reservation by its id, @GetMapping("/id"), is specified in the controller, which maps the incoming GET requests to path a variable supplied with the specific id. This controller function delegated the operation to the service layer method encapsulating the business logic. More specifically in the service implementation layer, getReservationById method has a return type of List and only accepts one parameter, String id.



```
@Override
public List<Reservation> getReservationById(String id) {

    return reservationRepository.getReservationById(id);
}
```

Figure 77: getReservationById method implemented in service implementation class

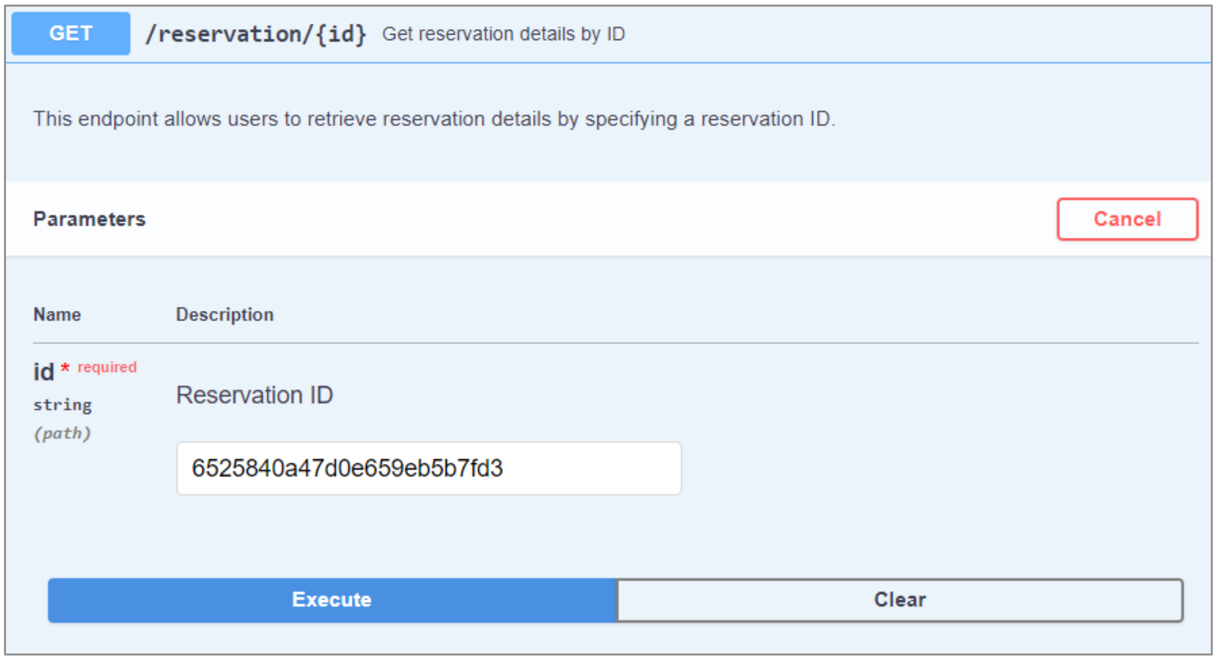
Its purpose is to fetch reservations using the specified id. For the above reason, the getReservationById method implemented in the repository layer is invoked, with the id as an input. Reservation Repository extends Mongo Repository by allowing it to communicate with a MongoDB database. Within the repository, the getReservationById(String id) function provides the query for retrieving reservations by their ID from the database (Figure 78).



```
2 usages   ITSOU Eleni +1 *
@Repository
public interface ReservationRepository extends MongoRepository<Reservation, String> {

    1 usage   itsoueleni
    List<Reservation> getReservationById(String id);


}
```

Figure 78: getReservationById in the Repository layer.

We can test the GET /reservation/id/ endpoint by using the Swagger UI and sending a request by using the reservation id Additionally, Swagger UI retrieves the response sent from the server and we can verify that the anticipated status code '200' is returned as well as the expected response body.



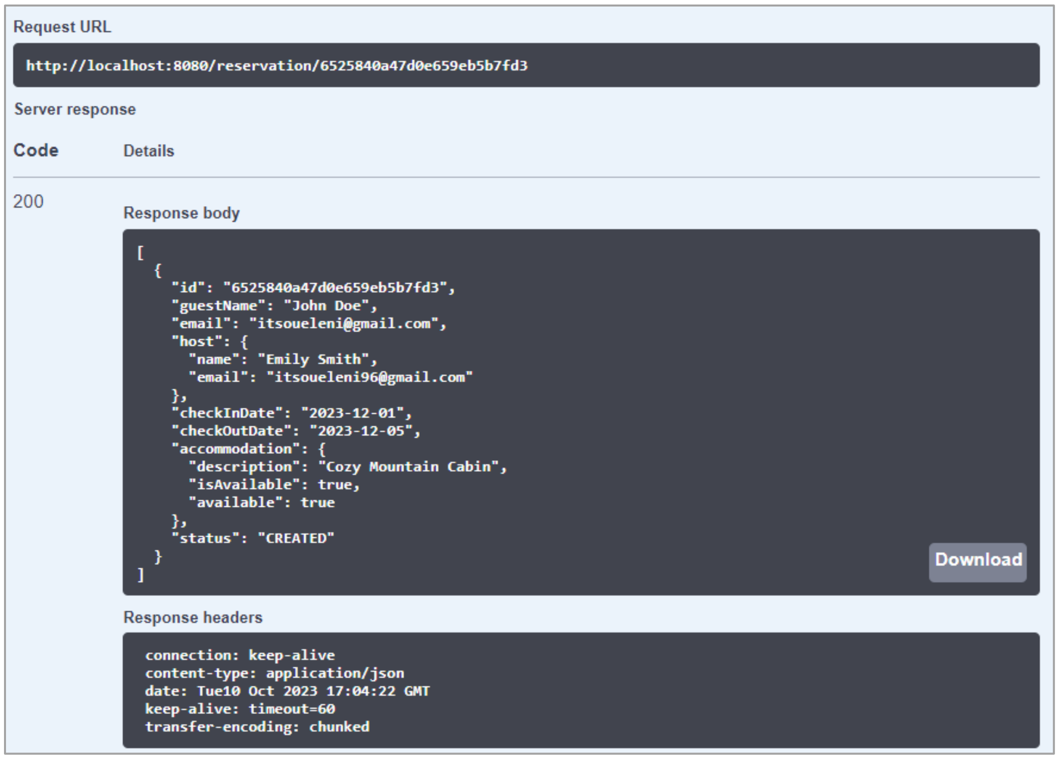
GET /reservation/{id} Get reservation details by ID

This endpoint allows users to retrieve reservation details by specifying a reservation ID.

Parameters

Name	Description
id * required string (path)	Reservation ID 6525840a47d0e659eb5b7fd3

Figure 79: Test of the GET reservation by id


Request URL

```
http://localhost:8080/reservation/6525840a47d0e659eb5b7fd3
```

Server response

Code Details

200 Response body

```
[
  {
    "id": "6525840a47d0e659eb5b7fd3",
    "guestName": "John Doe",
    "email": "itsoueleni@gmail.com",
    "host": {
      "name": "Emily Smith",
      "email": "itsoueleni96@gmail.com"
    },
    "checkInDate": "2023-12-01",
    "checkOutDate": "2023-12-05",
    "accommodation": {
      "description": "Cozy Mountain Cabin",
      "isAvailable": true,
      "available": true
    },
    "status": "CREATED"
  }
]
```

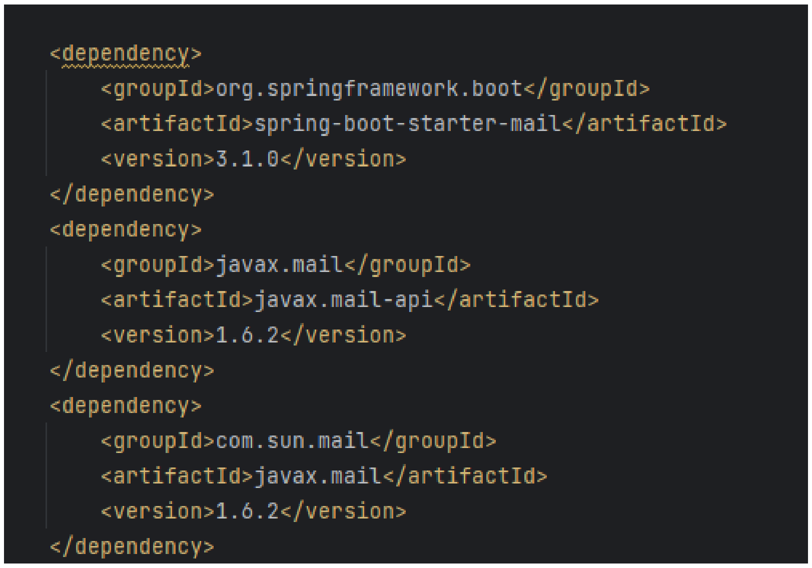
Response headers

```
connection: keep-alive
content-type: application/json
date: Tue10 Oct 2023 17:04:22 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Figure 80:Test of the GET reservation by id method using Swagger UI- Check response status code and response body.

5.2.7 Send a notification email via Java Mail API and SMTP protocol.

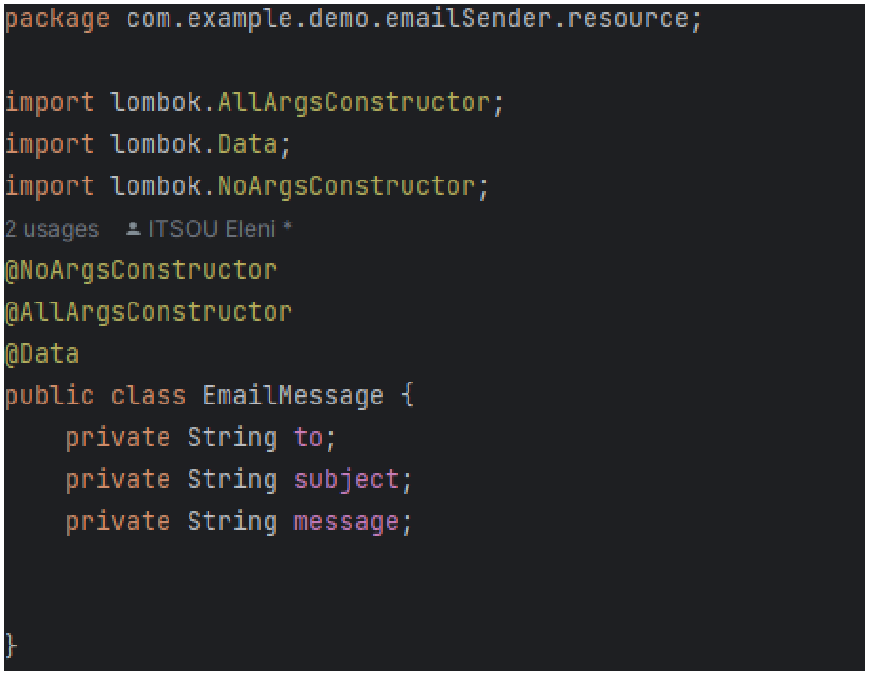
Oracle Corporation, the firm that produces Java technology, is the provider of JavaMail API. The JavaMail API is available on the Java platform for developing email and messaging applications. It includes classes for sending, receiving, and modifying email messages over SMTP (Simple Mail Transfer Protocol), IMAP (Internet Message Access Protocol), and POP3 (Post Office Protocol) [44]. Developers can utilize the JavaMail API in their projects by adding the JavaMail library. The library is available for download from the Oracle website or as a dependency in build management systems such as Maven. Dependencies needed for using JavaMail are the JavaMail API dependency, spring-boot-starter-mail and as well as the con.sun.mail implementation provider.



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-mail</artifactId>
    <version>3.1.0</version>
</dependency>
<dependency>
    <groupId>javax.mail</groupId>
    <artifactId>javax.mail-api</artifactId>
    <version>1.6.2</version>
</dependency>
<dependency>
    <groupId>com.sun.mail</groupId>
    <artifactId>javax.mail</artifactId>
    <version>1.6.2</version>
</dependency>
```

Figure 81: Dependencies needed for using the JavaMail API

The EmailMessage class was used to structure the email message, that has the to, subject and message email properties (Figure 82).

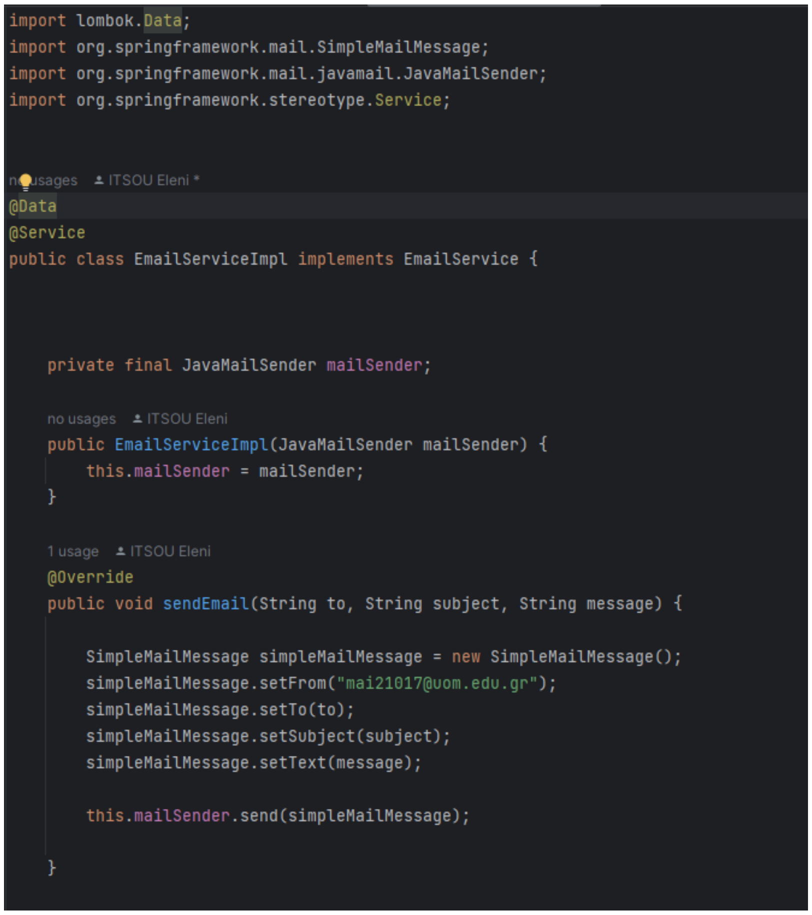


```java
package com.example.demo.emailSender.resource;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
2 usages   ITSOU Eleni *
@NoArgsConstructor
@AllArgsConstructor
@Data
public class EmailMessage {
    private String to;
    private String subject;
    private String message;


}
```

Figure 82: EmailMessage class

The EmailService interface is implemented by EmailServiceImpl class (Figure 83). To send emails, it employs JavaMailSender. The sendEmail method constructs a SimpleMailMessage, sets the sender, recipient, topic, and message, and then uses the mailSender to send the email. JavaMail API provides the SimpleMaiMessage class for creating plain text messages in SpringBoot framework. It enables developers to configure fundamental email attributes like in this case, as it is used for simple email purposes.

SimpleMailMessage is used for simple email communication without dealing with MIME messages complexity. SimpleMailMessage is often used in conjunction with SMPT (Simple Mail Transfer Protocol) to deliver simple messages. SMPT provides email transmission, while the basic content is produced by SimpleMailMessage.



```java
import lombok.Data;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;


no usages   ITSOU Eleni *
@Data
@Service
public class EmailServiceImpl implements EmailService {



    private final JavaMailSender mailSender;

    no usages   ITSOU Eleni
    public EmailServiceImpl(JavaMailSender mailSender) {
        this.mailSender = mailSender;
    }

    1 usage   ITSOU Eleni
    @Override
    public void sendEmail(String to, String subject, String message) {

        SimpleMailMessage simpleMailMessage = new SimpleMailMessage();
        simpleMailMessage.setFrom("mai21017@uom.edu.gr");
        simpleMailMessage.setTo(to);
        simpleMailMessage.setSubject(subject);
        simpleMailMessage.setText(message);

        this.mailSender.send(simpleMailMessage);

    }
```

Figure 83:Email Service Implementation class

An email Controller class was used to handle post requests with the /send-email endpoint. When a POST request is made to this endpoint including to subject and message as message attributes, an Email Message object is created, and passed in the EmailServices sendEmail method. To set up the SMTP server in gmail a two-step verification code needs to be set up for the google account. Then the security key generated can be used as password connecting the java application.

5.3 Chapter Summary

In this chapter, the Model-View-Controller (MVC) design pattern is used to create the Reservation REST API. The API is divided into 4 layers: The Repository layer, which handles the database communication, the service layer, which contains the business logic, and the Model layer which represents the application structure and entities. As a build automation tool, Apache Maven is used to streamline project development and dependency management.

The set-up procedure including how to create a project skeleton with the necessary dependencies in Spring initializer is thoroughly covered in the chapter. Building the REST API and connecting to the database require the Spring Web and Spring Data MongoDB dependencies respectively. Using Spring fox Swagger UI, the API documentation is implemented making it simple to test and view the endpoints.

In addition, the creation, update and confirmation, rejection, placement and deletion of reservations, as well as other API functions, is thoroughly discussed. Each endpoint is tested using the Swagger UI to ensure the correct operation. The chapter also shows how to send email notifications for various reservation related events using the JavaMail API and SMTP protocol. Overall, the chapter offers a thorough tutorial for creating a reservation API, covering key ideas, implementation specifics and testing techniques.

To sum up all the endpoints and their operations of the Reservation REST API are summarized in the following table:

HTTP Method	Endpoint	Operation	Success Response	Error Response
POST	/reservation	Create a new reservation.	201 Created (with reservation URI in header)	400 Bad Request, 500 Internal Server Error
GET	/reservation/{id}	Get reservation details by ID.	200 OK (with reservation details)	404 Not Found
PUT	/reservation/{id}/updateReservationStatus	Update reservation status by ID.	200 OK (with success message)	400 Bad Request, 500 Internal Server Error
PUT	/reservation/{id}/rejectReservation	Reject a reservation by ID.	200 OK (with success message)	400 Bad Request, 500 Internal Server Error
PUT	/reservation/{id}/confirmReservation	Confirm a reservation by ID.	200 OK (with success message)	400 Bad Request, 500 Internal Server Error
POST	/reservation/{id}/placeReservation	Place a reservation by ID.	201 Created (with success message)	400 Bad Request, 500 Internal Server Error
DELETE	/reservation/{id}	Delete a reservation by ID.	204 No Content	404 Not Found

Table 5.3-1: Reservation API endpoint summary

Chapter 6 : Business process orchestration using Postman

In the word of API development, Postman is a well-regarded tool for testing as well as automating HTTP requests. It gives developers the ability to carefully evaluate API endpoints, manage API requests and keep track of the responses. In addition, Postman can manage API request by using collections where is classifying and organizing the API requests resulting in accelerating the testing process. Moreover, process orchestration, can be achieved when API requests are created for each step of the reservation process and are chained together by using Postman variables and scripts features.

By leveraging Postman, complex inner working of three different reservation scenarios can be verified. These scenarios can act as performance indicators under different conditions, as well as illustrating the interactions of the different API methods. The first scenario which concludes in the 'Reservation confirmation' is the 'happy path' in which the reservation is successfully created and confirmed by the host. The second scenario is the 'Availability check' scenario which records the system handling of the reservations found as occupied in the system. Lastly, the 'Reservation rejection' records the host's rejection of a reservation.

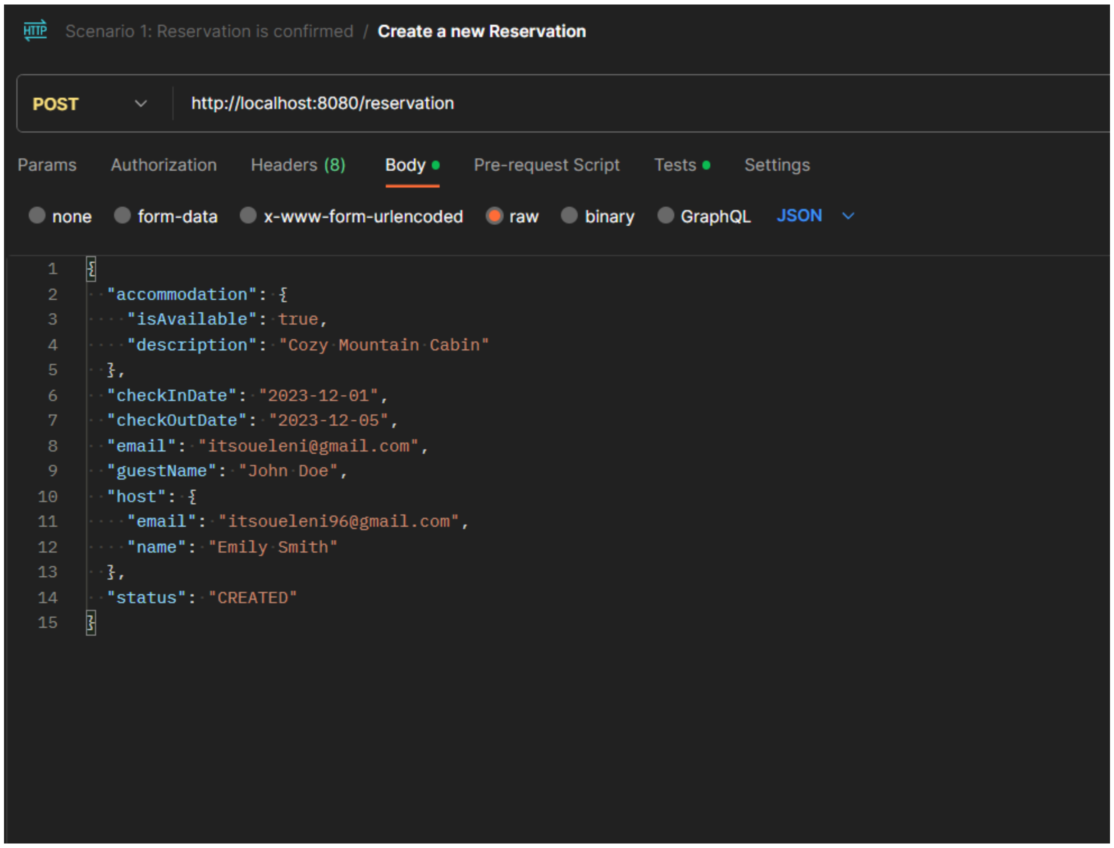
These three examples, cover a wide range of real word scenarios that a reservation system can run into, from simple operations to more complex ones that call for a particular error handling or require precise communication. By using these three scenarios the reservation system functionality can be verified efficiently, resulting in a good user experience.

6.1 Reservation confirmation process scenario

In the following detailed breakdown of the reservation confirmation, the intricate steps and interactions that occur within the system to achieve a reservation confirmation are displayed. The reservation confirmation process is a multi-step journey, involving several API requests that serve a specific purpose in the overall workflow. In the reservation confirmation scenario the API requests encompass a series of actions that signify that a guest reservation has been successfully created and confirmed by the host.

The structure of the collection that result in a Reservation confirmation, can be broken down as follows:

Create a new reservation method is the first API request, having the purpose of creating a new reservation. The request is a POST request, where the required data are presented in the JSON payload (Figure 84). In the 'Tests' section, a test script is used to determine if the correct response status is returned. This test serves to confirm that the API request was successful in creating, having a 201 created status code. The test will succeed if the status code is in fact 201 and will fail in any other case. Furthermore, the URL included in the header response can be saved as location variable (Figure 85). Get reservation id method is retrieved several times in the flow for verifying that the reservation status has been changed successfully.

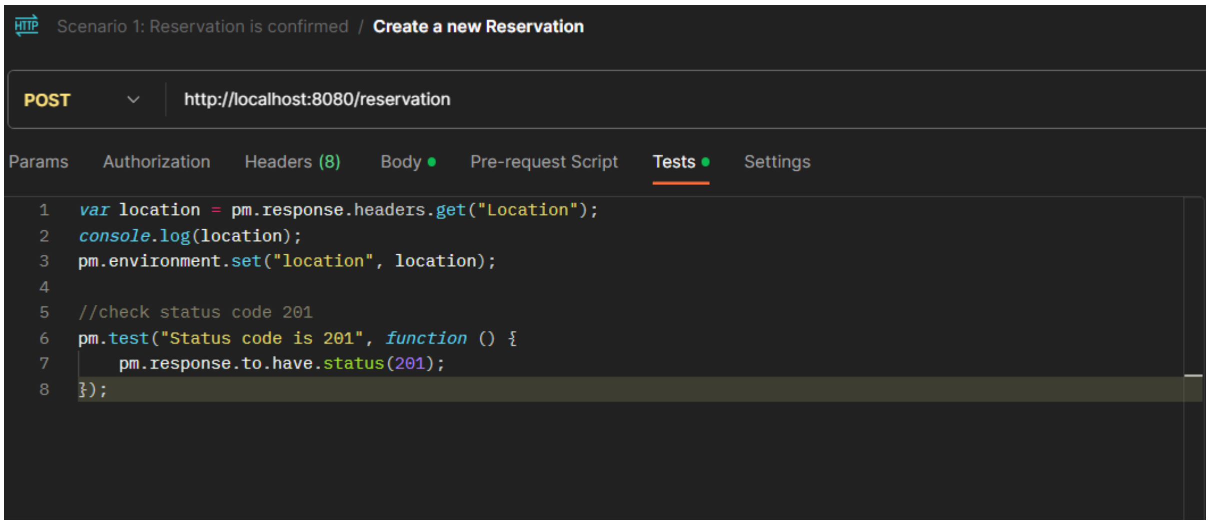


POST http://localhost:8080/reservation

Body: [✓ raw] JSON

```
{
  "accommodation": {
    "isAvailable": true,
    "description": "Cozy Mountain Cabin"
  },
  "checkInDate": "2023-12-01",
  "checkOutDate": "2023-12-05",
  "email": "itsoueleni@gmail.com",
  "guestName": "John Doe",
  "host": {
    "email": "itsoueleni96@gmail.com",
    "name": "Emily Smith"
  },
  "status": "CREATED"
}
```

Figure 84: Reservation confirmation business process scenario- test the create reservation POST method using Postman.

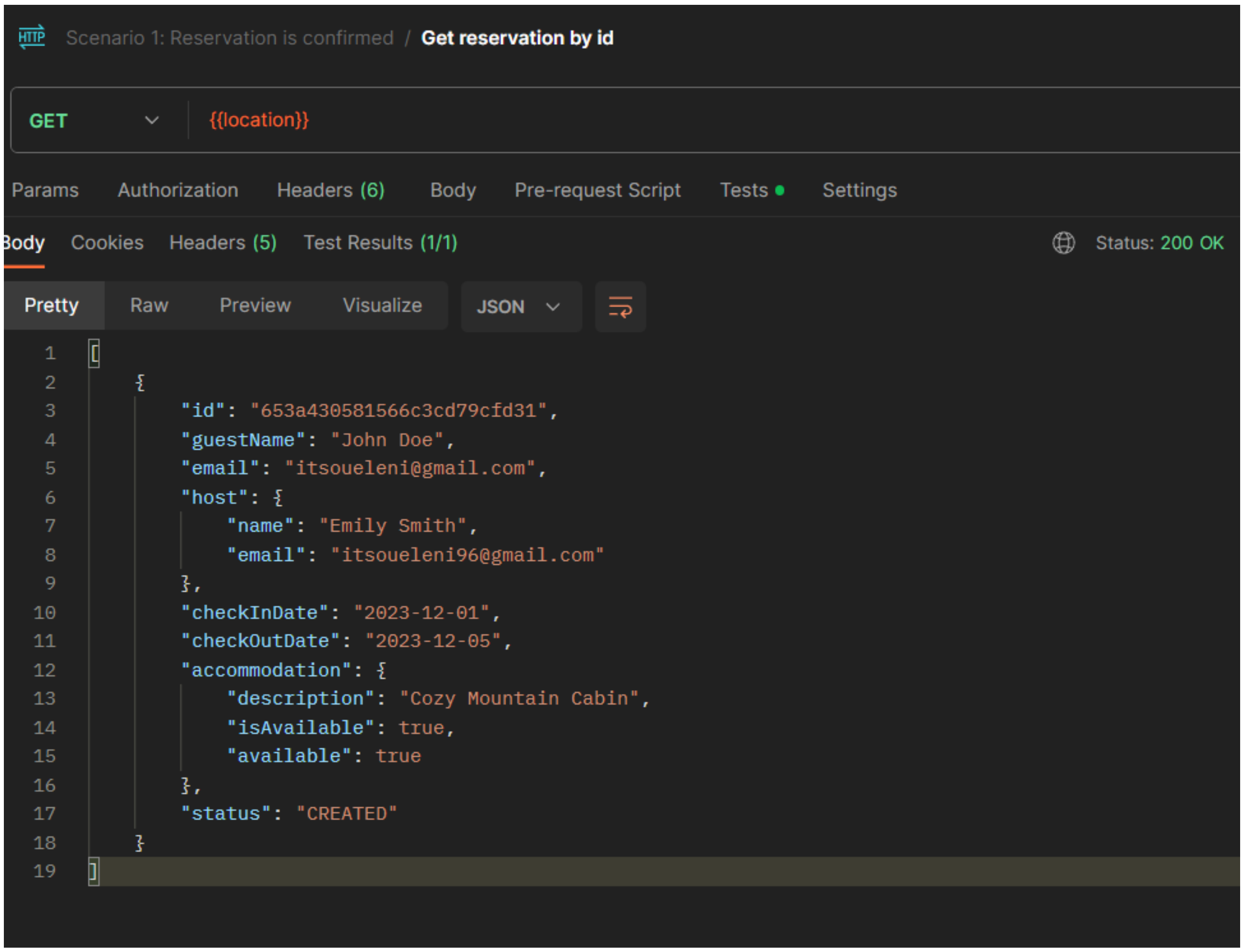


Scenario 1: Reservation is confirmed / Create a new Reservation
POST
http://localhost:8080/reservation
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings

```
var location = pm.response.headers.get("Location");
console.log(location);
pm.environment.set("location", location);

//check status code 201
pm.test("Status code is 201", function () {
    pm.response.to.have.status(201);
});
```

Figure 85: Reservation confirmation business process scenario- test scripts in Create a new reservation post method.

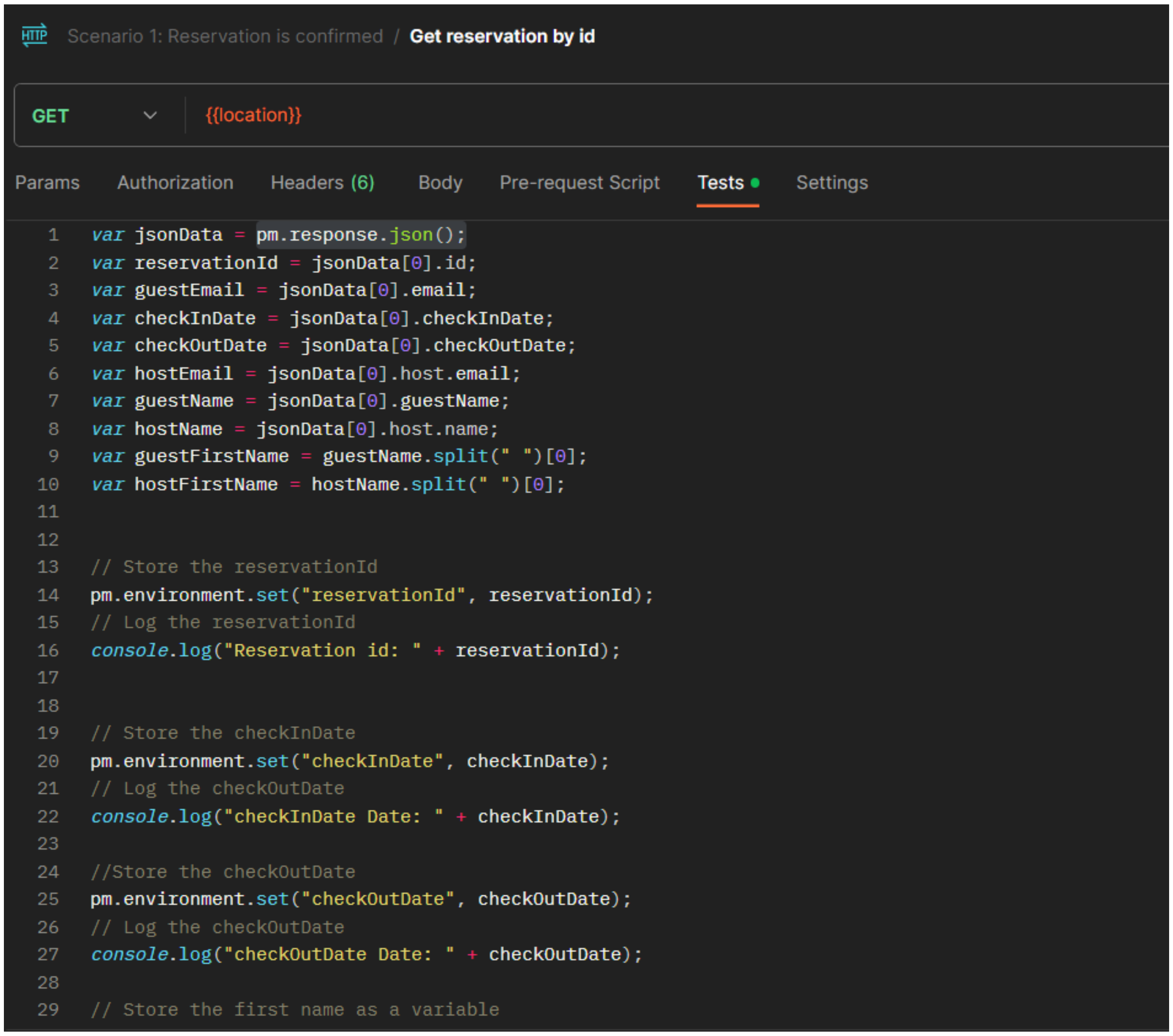
Get reservation by id method is the second method used in the process scenario and is used to obtain the reservation details by its reservation id. The URL is for initiating the request and is retrieved by using the Location variable that was saved in the previous request (Figure 86). In the test section of Postman, a test script is used to test that the correct status code is returned, and several variable assignments are specified. More specifically, by using pm.response.json() JavaScript function, the JSON response is parsed and saved in a variable. Next, various variables extracted from the Json data object are assigned as variables, and then stored as Postman environment variables.



```
[
    {
        "id": "653a430581566c3cd79cfd31",
        "guestName": "John Doe",
        "email": "itsoueleni@gmail.com",
        "host": {
            "name": "Emily Smith",
            "email": "itsoueleni96@gmail.com"
        },
        "checkInDate": "2023-12-01",
        "checkOutDate": "2023-12-05",
        "accommodation": {
            "description": "Cozy Mountain Cabin",
            "isAvailable": true,
            "available": true
        },
        "status": "CREATED"
    }
]
```

Figure 86: Reservation confirmation business process scenario- test the reservation by id GET method using Postman.

Update the Reservation status method is later consequently used for updating the reservation status to ‘Pending’ if the accommodation availability has been verified. A test script has also been initiated checking that the expected status code is returned. The connection with the previous request is achieved as the reservation id previously saved is utilized in the request URL (Figure 88).



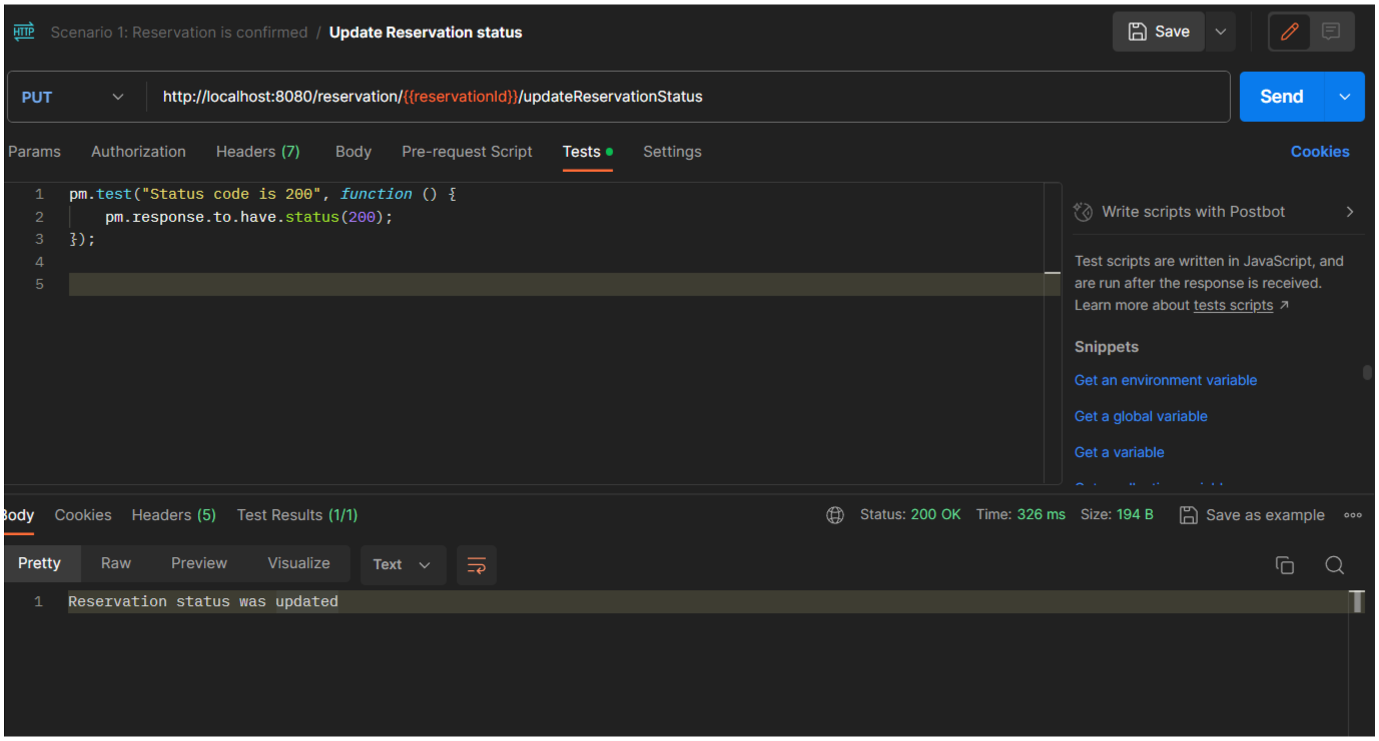
```
var jsonData = pm.response.json();
var reservationId = jsonData[0].id;
var guestEmail = jsonData[0].email;
var checkInDate = jsonData[0].checkInDate;
var checkOutDate = jsonData[0].checkOutDate;
var hostEmail = jsonData[0].host.email;
var guestName = jsonData[0].guestName;
var hostName = jsonData[0].host.name;
var guestFirstName = guestName.split(" ")[0];
var hostFirstName = hostName.split(" ")[0];


// Store the reservationId
pm.environment.set("reservationId", reservationId);
// Log the reservationId
console.log("Reservation id: " + reservationId);


// Store the checkInDate
pm.environment.set("checkInDate", checkInDate);
// Log the checkOutDate
console.log("checkInDate Date: " + checkInDate);

//Store the checkOutDate
pm.environment.set("checkOutDate", checkOutDate);
// Log the checkOutDate
console.log("checkOutDate Date: " + checkOutDate);

// Store the first name as a variable
```

Figure 87: Reservation confirmation business process scenario- variable assignment in get reservation id GET method.

In the upcoming requests, email notifications are being tested, where the recipient's email address, the message, and the subject are provided as query parameters to "http://localhost:8080/send-email" in the form of a POST request. More specifically, using the Java mail API post request, the guest as well as the host receive an email notification about the reservation upon submitting this request. In both cases the pre-request script section of Postman has been utilized for creating the email content, where variables declared in previous requests of the flow are included in the text body (Figure 89). Like in the previously submitted methods test scripts have been utilized for testing the expected status code. Both emails have been sent successfully to the applied user (Figure 90),(Figure 91).

Confirm reservation is the next method used utilizing the t reservation ID from the previous request to create a URL and sends a PUT request to that URL. Moreover 200 is the expected status code for the response which is tested with a test script (Figure 92).Get reservation by id is followed to confirm that the reservation status is updated to ‘Confirmed’, followed by the test script confirming the expected reservation status (Figure 93).

Send reservation confirmation to guest is followed, which is sent in the same way the notification of the availability has been sent. The request method is subsequently followed by the test script confirming the expected reservation status (Figure 94). Place reservation is the next method after the host confirms the reservation. The reservation ID from the previous request is used to generate a URL to which a POST request is sent. A successful response is anticipated to have a status code of 201 which is also tested with test script (Figure 95). Get reservation by id is followed to the reservation placement, confirming that the reservation has been placed in the system (Figure 96).



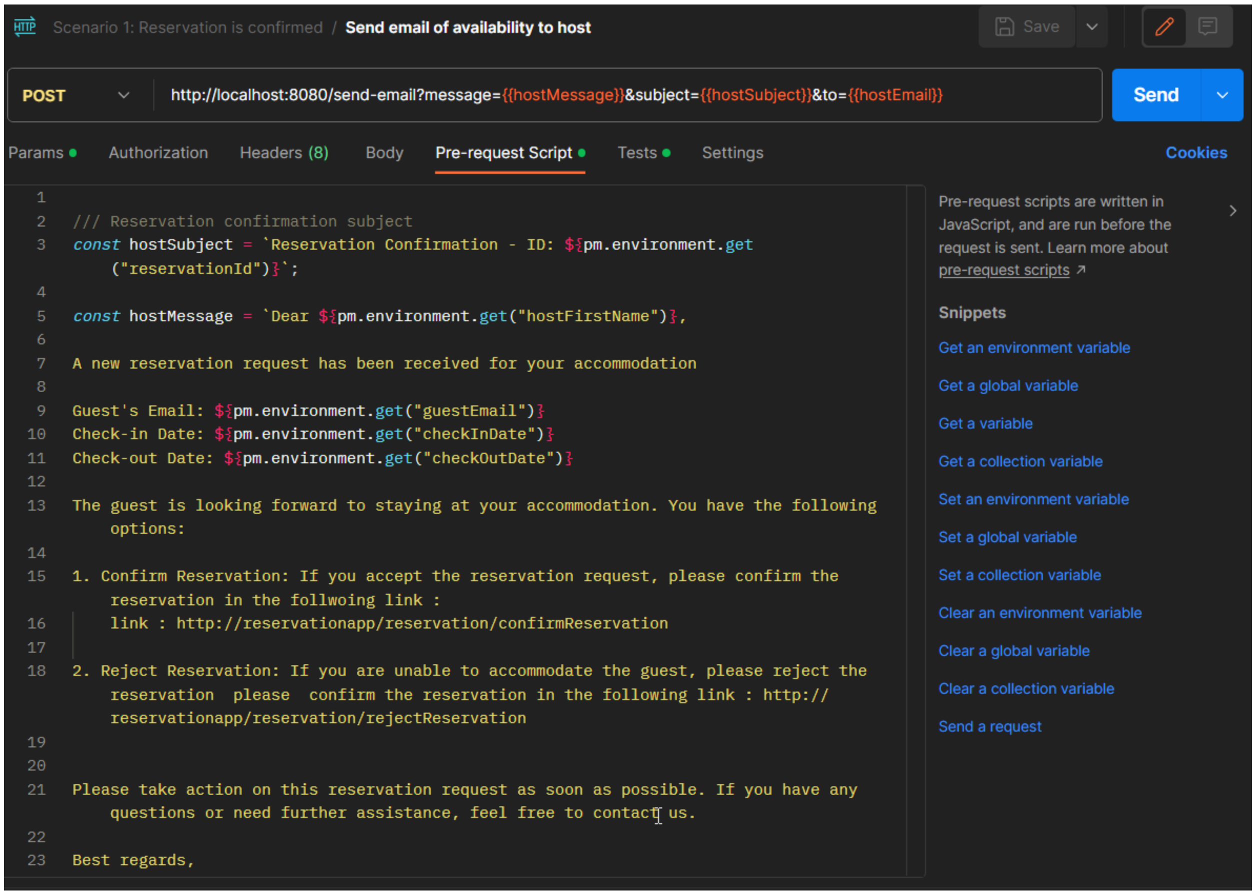
```
http://localhost:8080/reservation/{{reservationId}}/updateReservationStatus
```

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts ↗

```
Reservation status was updated
```

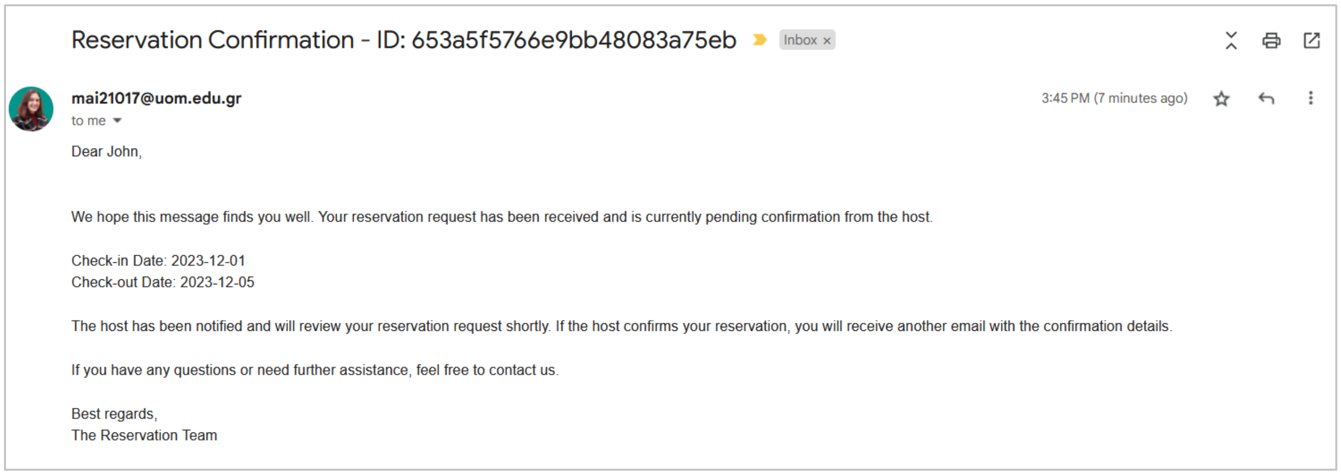
Figure 88: Reservation confirmation business process scenario- Test the Update Reservation status request.


Scenario 1: Reservation is confirmed / Send email of availability to host
Save
POST
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Pre-request scripts are written in JavaScript, and are run before the request is sent. Learn more about pre-request scripts ↗
Snippets
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Set a collection variable
Clear an environment variable
Clear a global variable
Clear a collection variable
Send a request

```
http://localhost:8080/send-email?message={{hostMessage}}&subject={{hostSubject}}&to={{hostEmail}}
```

```

/// Reservation confirmation subject
const hostSubject = `Reservation Confirmation - ID: ${pm.environment.get
    ("reservationId")}`;

const hostMessage = `Dear ${pm.environment.get("hostFirstName")},

A new reservation request has been received for your accommodation

Guest's Email: ${pm.environment.get("guestEmail")}
Check-in Date: ${pm.environment.get("checkInDate")}
Check-out Date: ${pm.environment.get("checkOutDate")}

The guest is looking forward to staying at your accommodation. You have the following
    options:

1. Confirm Reservation: If you accept the reservation request, please confirm the
    reservation in the follwoing link :
    link : http://reservationapp/reservation/confirmReservation

2. Reject Reservation: If you are unable to accommodate the guest, please reject the
    reservation  please  confirm the reservation in the following link : http://
    reservationapp/reservation/rejectReservation


Please take action on this reservation request as soon as possible. If you have any
    questions or need further assistance, feel free to contact us.

Best regards,
```

Figure 89: Reservation confirmation business process scenario- test the email notification API.



Reservation Confirmation - ID: 653a5f5766e9bb48083a75eb

mai21017@uom.edu.gr
to me

Dear John,

We hope this message finds you well. Your reservation request has been received and is currently pending confirmation from the host.

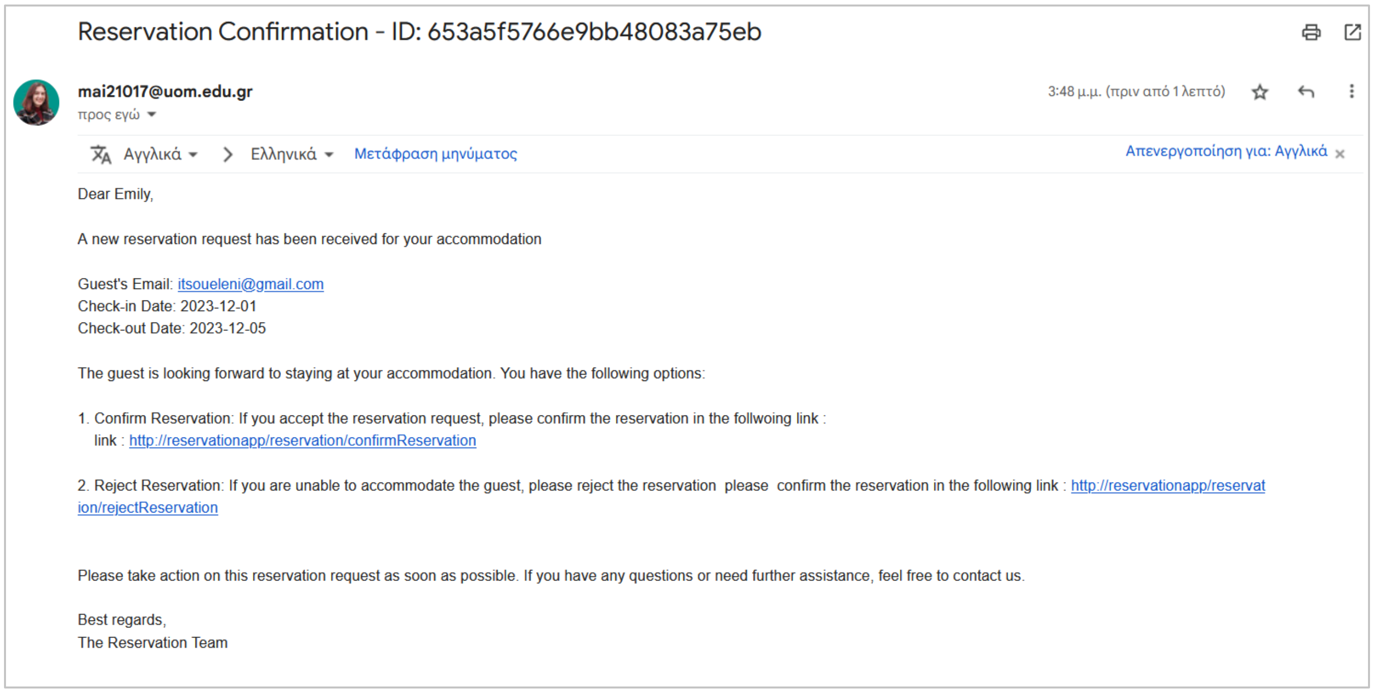
Check-in Date: 2023-12-01
Check-out Date: 2023-12-05

The host has been notified and will review your reservation request shortly. If the host confirms your reservation, you will receive another email with the confirmation details.

If you have any questions or need further assistance, feel free to contact us.

Best regards,
The Reservation Team

Figure 90: Email sent to guest.



Reservation Confirmation - ID: 653a5f5766e9bb48083a75eb

mai21017@uom.edu.gr
προς εγώ

3:48 μ.μ. (πριν από 1 λεπτό)

Αγγλικά > Ελληνικά Μετάφραση μηνύματος
Απενεργοποίηση για: Αγγλικά

Dear Emily,

A new reservation request has been received for your accommodation

Guest's Email: itsoueleni@gmail.com
Check-in Date: 2023-12-01
Check-out Date: 2023-12-05

The guest is looking forward to staying at your accommodation. You have the following options:

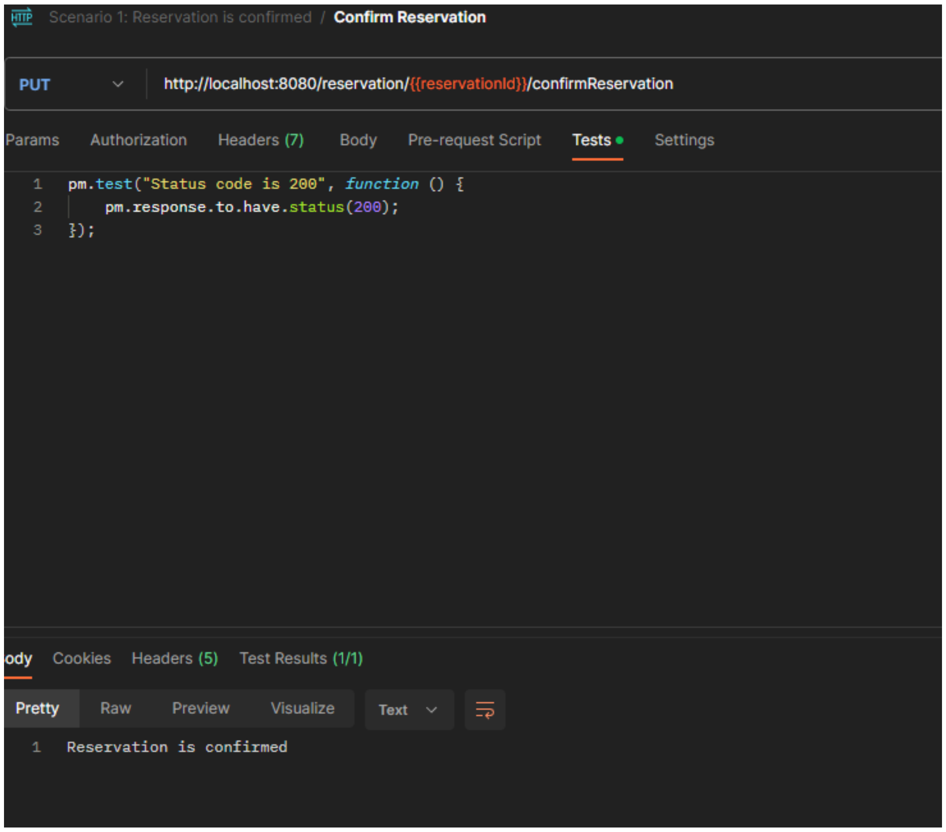
1. Confirm Reservation: If you accept the reservation request, please confirm the reservation in the follwoing link :
link : http://reservationapp/reservation/confirmReservation

2. Reject Reservation: If you are unable to accommodate the guest, please reject the reservation please confirm the reservation in the following link : http://reservationapp/reservation/rejectReservation

Please take action on this reservation request as soon as possible. If you have any questions or need further assistance, feel free to contact us.

Best regards,
The Reservation Team

Figure 91: Email sent to host.



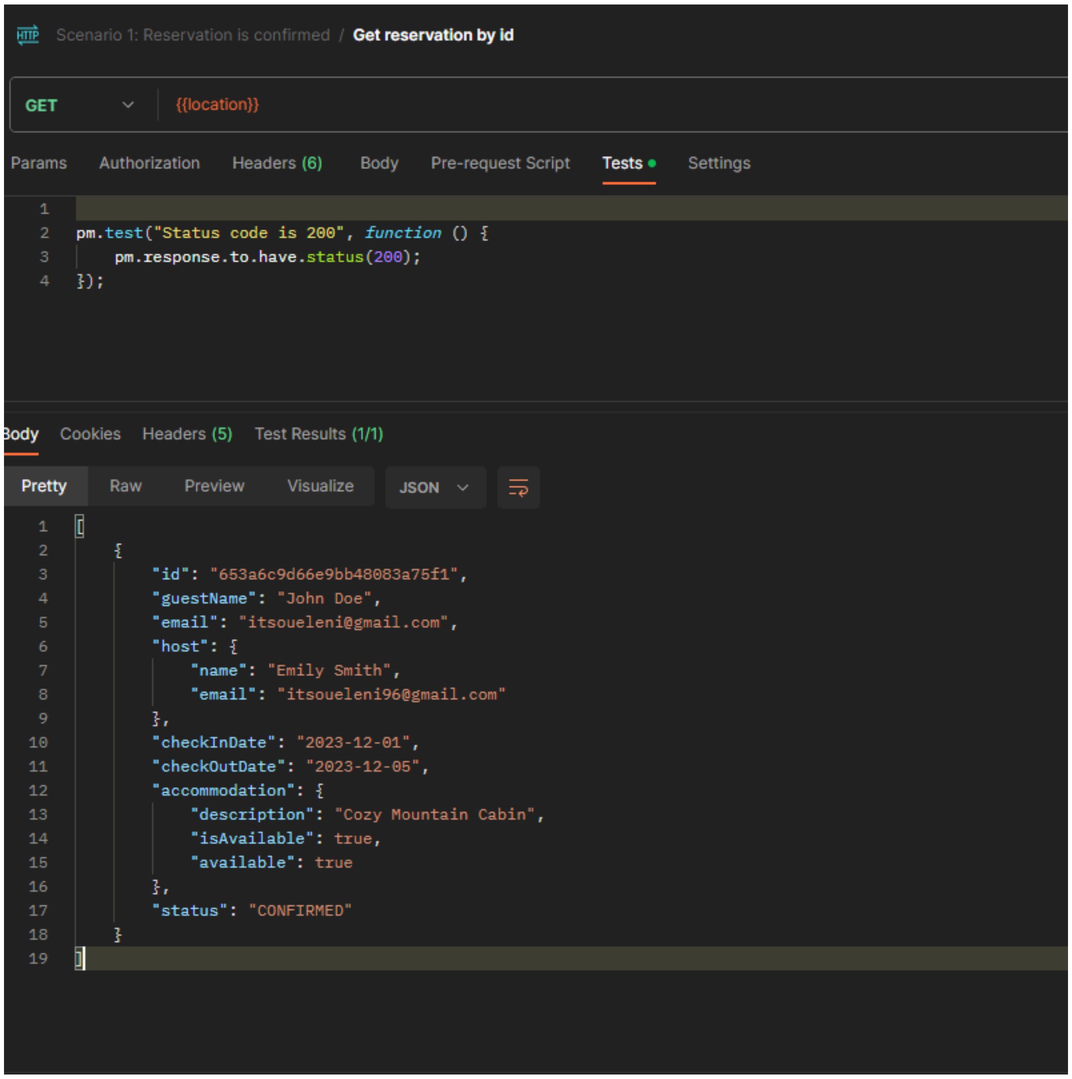
Scenario 1: Reservation is confirmed / Confirm Reservation

PUT http://localhost:8080/reservation/{{reservationId}}/confirmReservation

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

```
Reservation is confirmed
```

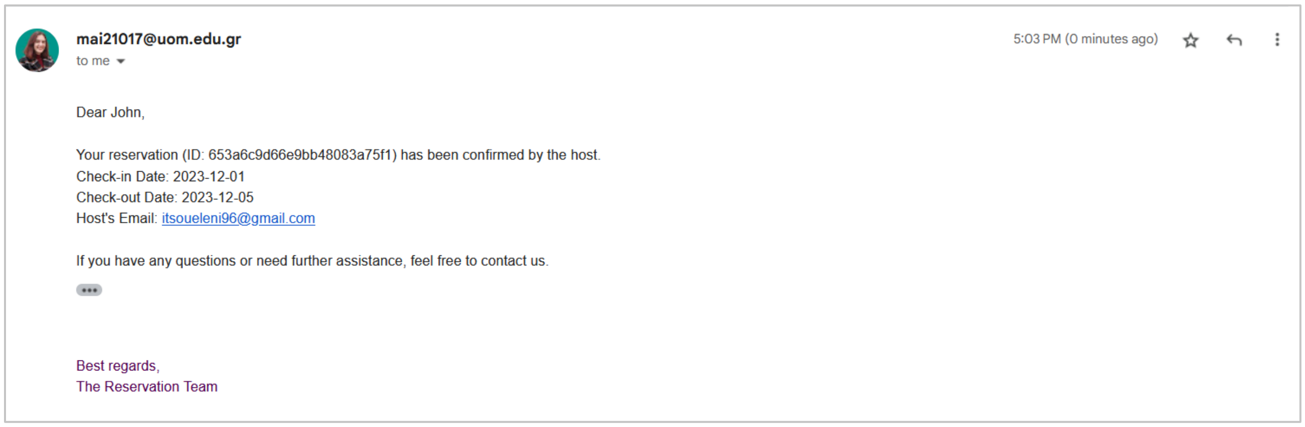
Figure 92: Reservation confirmation business process scenario- test the confirmation PUT method.



```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

```
[
    {
        "id": "653a6c9d66e9bb48083a75f1",
        "guestName": "John Doe",
        "email": "itsoueleni@gmail.com",
        "host": {
            "name": "Emily Smith",
            "email": "itsoueleni96@gmail.com"
        },
        "checkInDate": "2023-12-01",
        "checkOutDate": "2023-12-05",
        "accommodation": {
            "description": "Cozy Mountain Cabin",
            "isAvailable": true,
            "available": true
        },
        "status": "CONFIRMED"
    }
]
```

Figure 93: Reservation confirmation business process scenario- test the get reservation id confirmed status.


mai21017@uom.edu.gr
to me
5:03 PM (0 minutes ago)

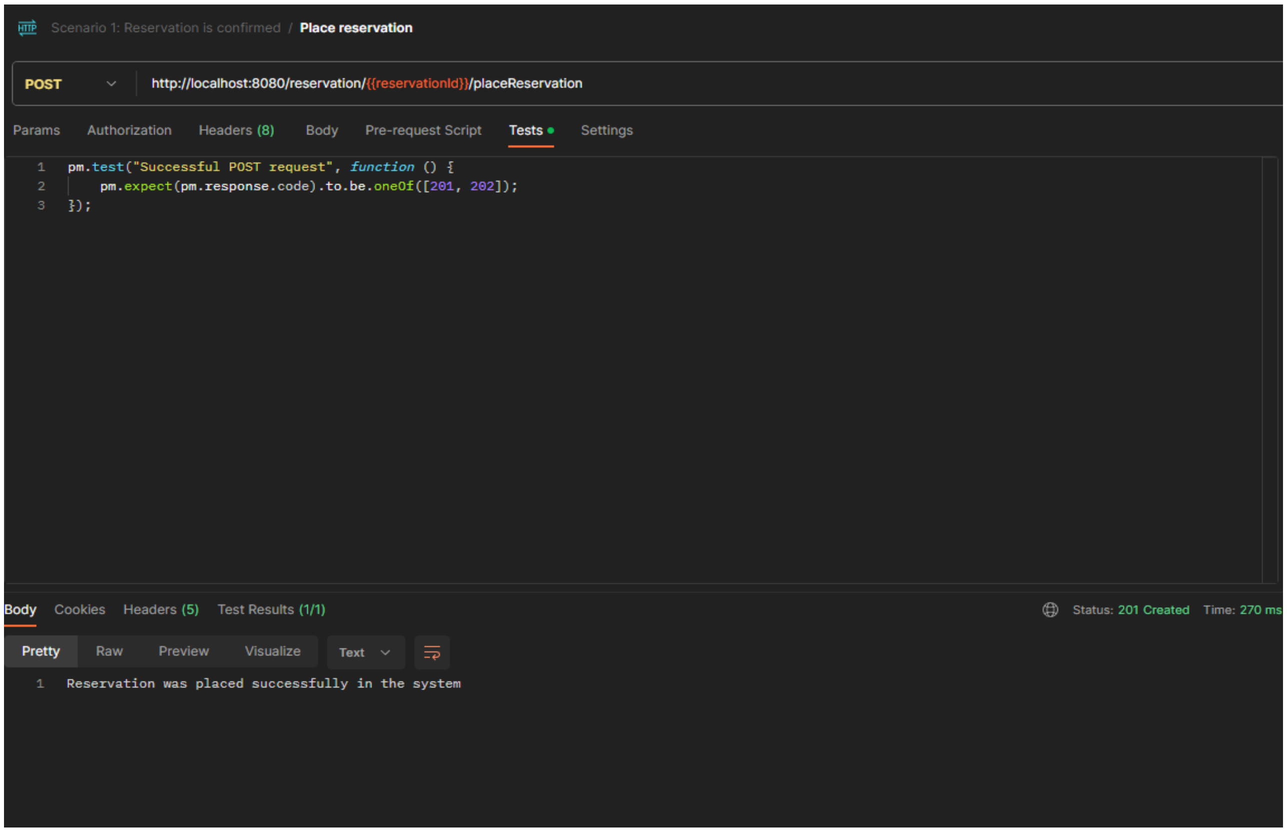
Dear John,

Your reservation (ID: 653a6c9d66e9bb48083a75f1) has been confirmed by the host.
Check-in Date: 2023-12-01
Check-out Date: 2023-12-05
Host's Email: itsoueleni96@gmail.com

If you have any questions or need further assistance, feel free to contact us.

Best regards,
The Reservation Team

Figure 94: Confirmation email sent to guest.


Scenario 1: Reservation is confirmed / Place reservation

POST http://localhost:8080/reservation/{{reservationId}}/placeReservation

Params Authorization Headers (8) Body Pre-request Script Tests Settings

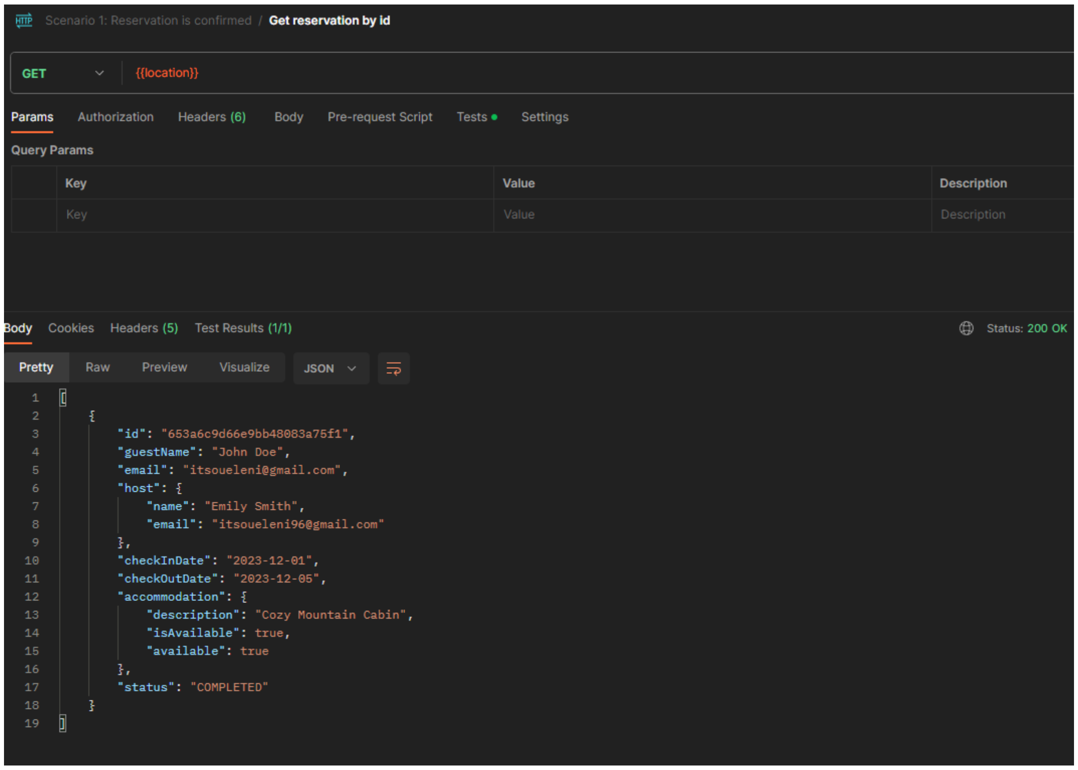
```
pm.test("Successful POST request", function () {
    pm.expect(pm.response.code).to.be.oneOf([201, 202]);
});
```

Body Cookies Headers (5) Test Results (1/1) Status: 201 Created Time: 270 ms

Pretty Raw Preview Visualize Text

```
Reservation was placed successfully in the system
```

Figure 95: Reservation confirmation business process scenario- test the place reservation POST method.


Scenario 1: Reservation is confirmed / Get reservation by id

GET {{location}}

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	Key	Value	Description
	Key	Value	Description

Body Cookies Headers (5) Test Results (1/1) Status: 200 OK

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": "653a6c9d66e9bb48083a75f1",
        "guestName": "John Doe",
        "email": "itsoueleni@gmail.com",
        "host": {
            "name": "Emily Smith",
            "email": "itsoueleni96@gmail.com"
        },
        "checkInDate": "2023-12-01",
        "checkOutDate": "2023-12-05",
        "accommodation": {
            "description": "Cozy Mountain Cabin",
            "isAvailable": true,
            "available": true
        },
        "status": "COMPLETED"
    }
]
```

Figure 96: Reservation confirmation business process scenario- test the get reservation id completed status.

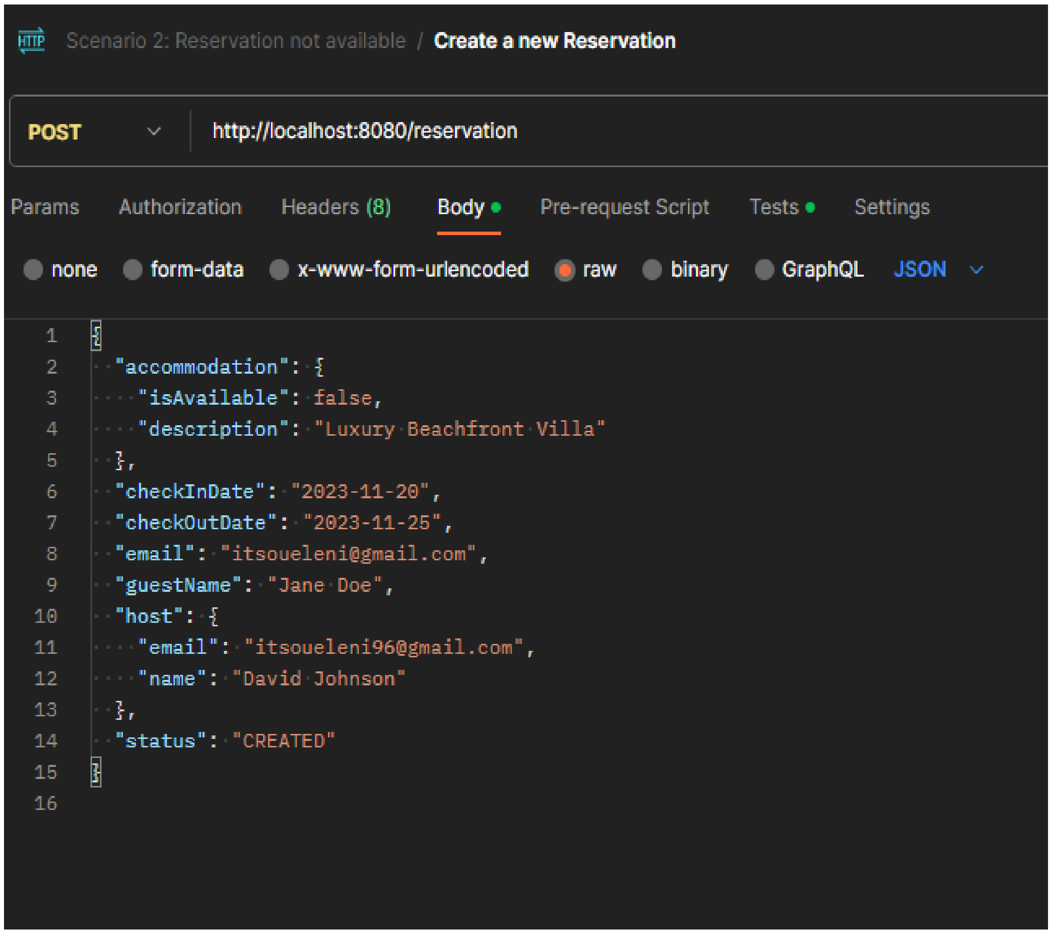
6.2 Reservation non-availability process scenario

This section highlights the scenario of booking an accommodation that is already defined as non-available in the reservation application due to simultaneous booking or delay in updating the availability status. This reservation request is handled by setting a rejection email to the guest and deleting the reservation request from the reservation system.

The structure of the collection that result in a Reservation confirmation non-availability, is broken down as follows:

- Create a reservation: A post request is created with the body entailing details about the reservation with an accommodation that is not available. The status code is also confirmed with a test script. Furthermore, the location of the reservation id URL is assigned to an environment variable (Figure 97).
- Get reservation by id (Reservation Status-Created): Like in previous scenario, the reservation id URL was retrieved from the previous request, and all information parsed from the JSON object are defined as environment variables (Figure 98).
- Update reservation status: In this method, the reservation status that is retrieved from the previous request, is updated to as the accommodation is not available for the type of period requested, with the testing confirming that the response status code is 200.
- Get reservation by id (Reservation Status-Pending): The method is retrieved again for verifying that the reservation status is of the reservation id retrieved from the previous request is updated to cancelled.
- Send non-availability email: The POST request method of the email message is used for sending a non-availability message to the guest. In the upcoming request, the recipient's email address, the message, and the subject are provided as query parameters to the request method. In the pre-request script section of Postman has been utilized for creating the email content, where variables declared in previous requests of the flow are included in the text body (Figure 99).
- Cancel reservation: The delete request method for removing a canceled reservation from the database of the reservation application is sent, containing the reservation id from the

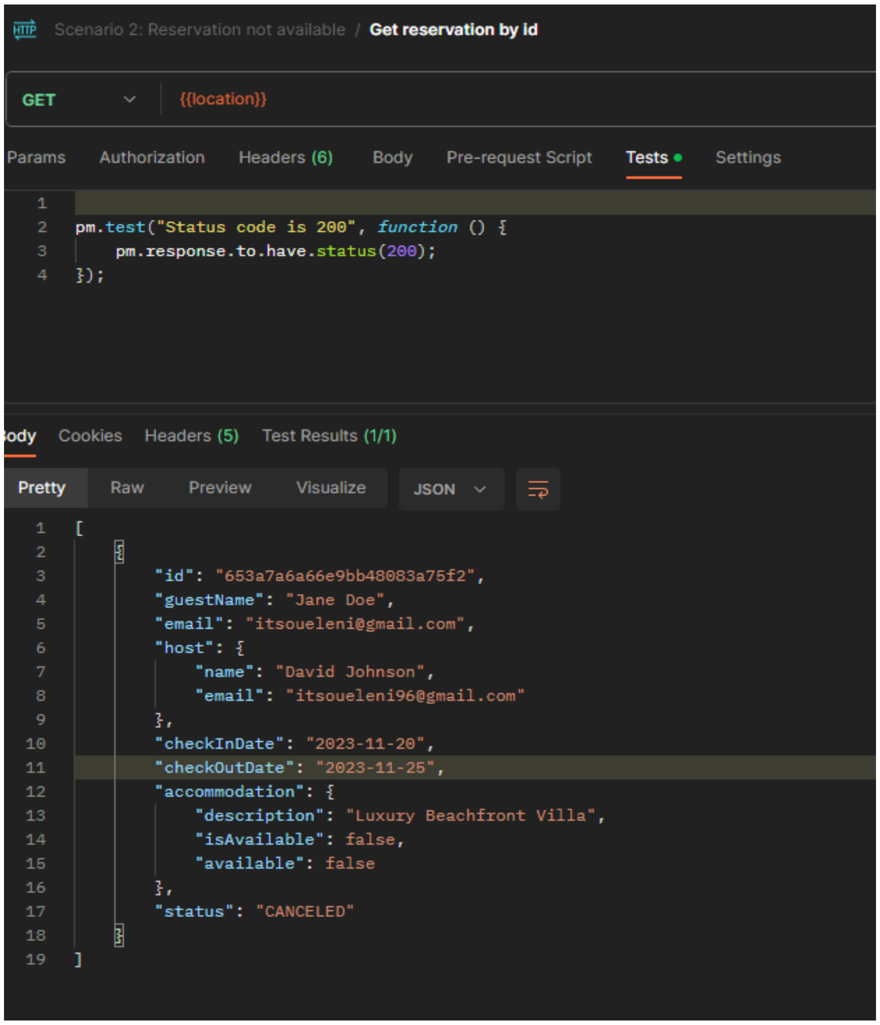
previous request. With this method the operation of a reservation cancelation is verified successfully as well as 204 expected status code (Figure 100).


Scenario 2: Reservation not available / Create a new Reservation
POST
Params Authorization Headers (8) Body Pre-request Script Tests Settings
none form-data x-www-form-urlencoded raw binary GraphQL JSON

POST http://localhost:8080/reservation

```
{
  "accommodation": {
    "isAvailable": false,
    "description": "Luxury Beachfront Villa"
  },
  "checkInDate": "2023-11-20",
  "checkOutDate": "2023-11-25",
  "email": "itsoueleni@gmail.com",
  "guestName": "Jane Doe",
  "host": {
    "email": "itsoueleni96@gmail.com",
    "name": "David Johnson"
  },
  "status": "CREATED"
}
```

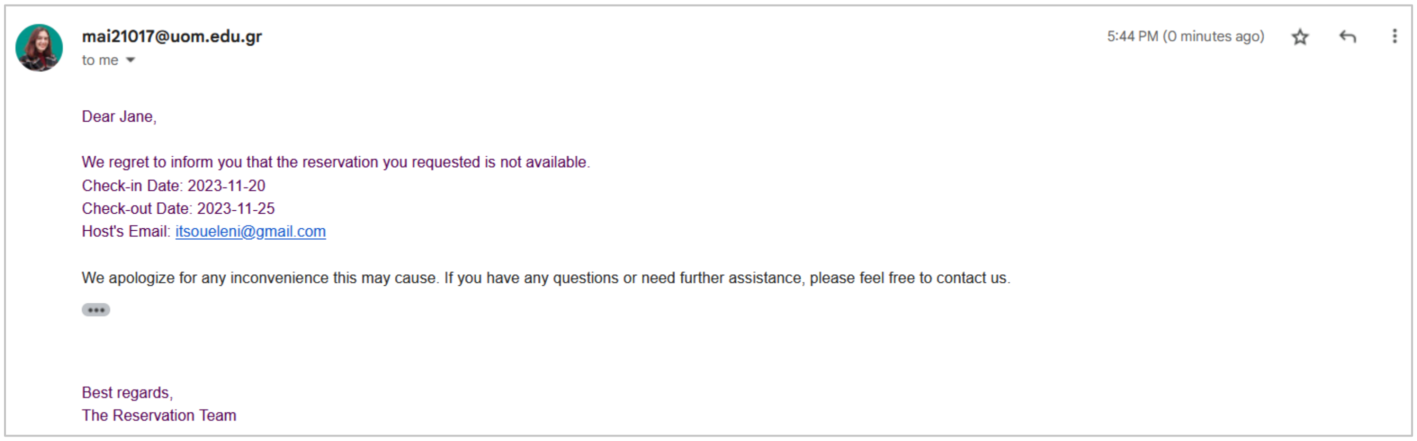
Figure 97: Reservation non-availability business process scenario- Create a new reservation request.



```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
```

```
[
    {
        "id": "653a7a6a66e9bb48083a75f2",
        "guestName": "Jane Doe",
        "email": "itsoueleni@gmail.com",
        "host": {
            "name": "David Johnson",
            "email": "itsoueleni96@gmail.com"
        },
        "checkInDate": "2023-11-20",
        "checkOutDate": "2023-11-25",
        "accommodation": {
            "description": "Luxury Beachfront Villa",
            "isAvailable": false,
            "available": false
        },
        "status": "CANCELED"
    }
]
```

Figure 98: Reservation non-availability business process scenario- Get reservation by id.



mai21017@uom.edu.gr
to me

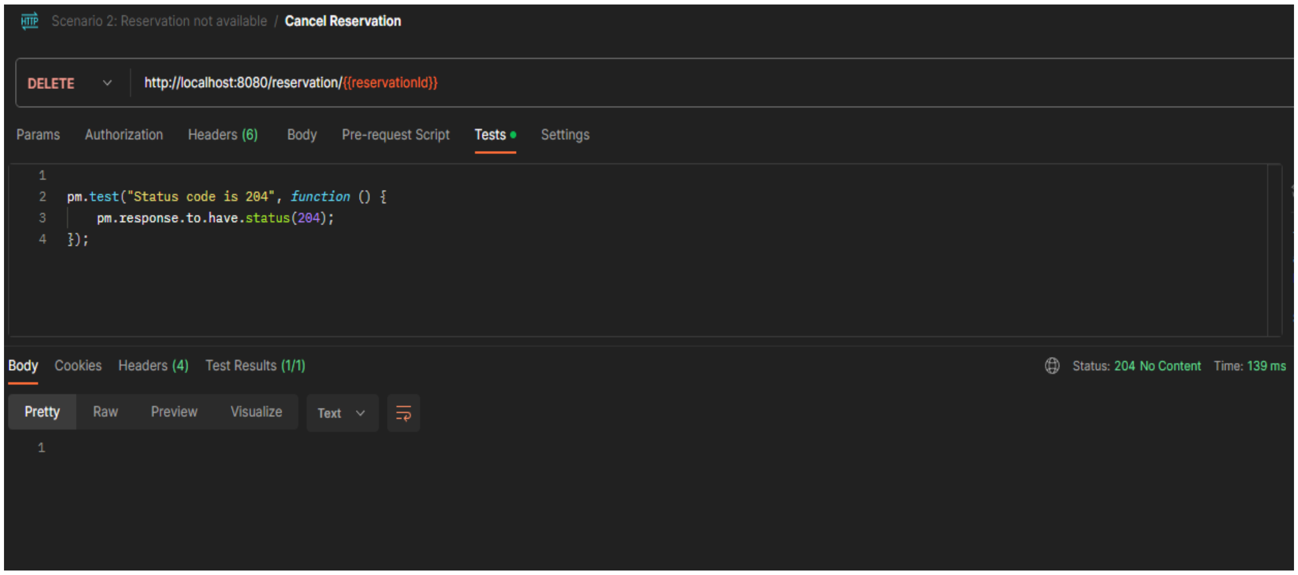
Dear Jane,

We regret to inform you that the reservation you requested is not available.
Check-in Date: 2023-11-20
Check-out Date: 2023-11-25
Host's Email: itsoueleni@gmail.com

We apologize for any inconvenience this may cause. If you have any questions or need further assistance, please feel free to contact us.

Best regards,
The Reservation Team

Figure 99: Email sent when the reservation is not available.



```
http://localhost:8080/reservation/{{reservationId}}
```

```
pm.test("Status code is 204", function () {
    pm.response.to.have.status(204);
});
```

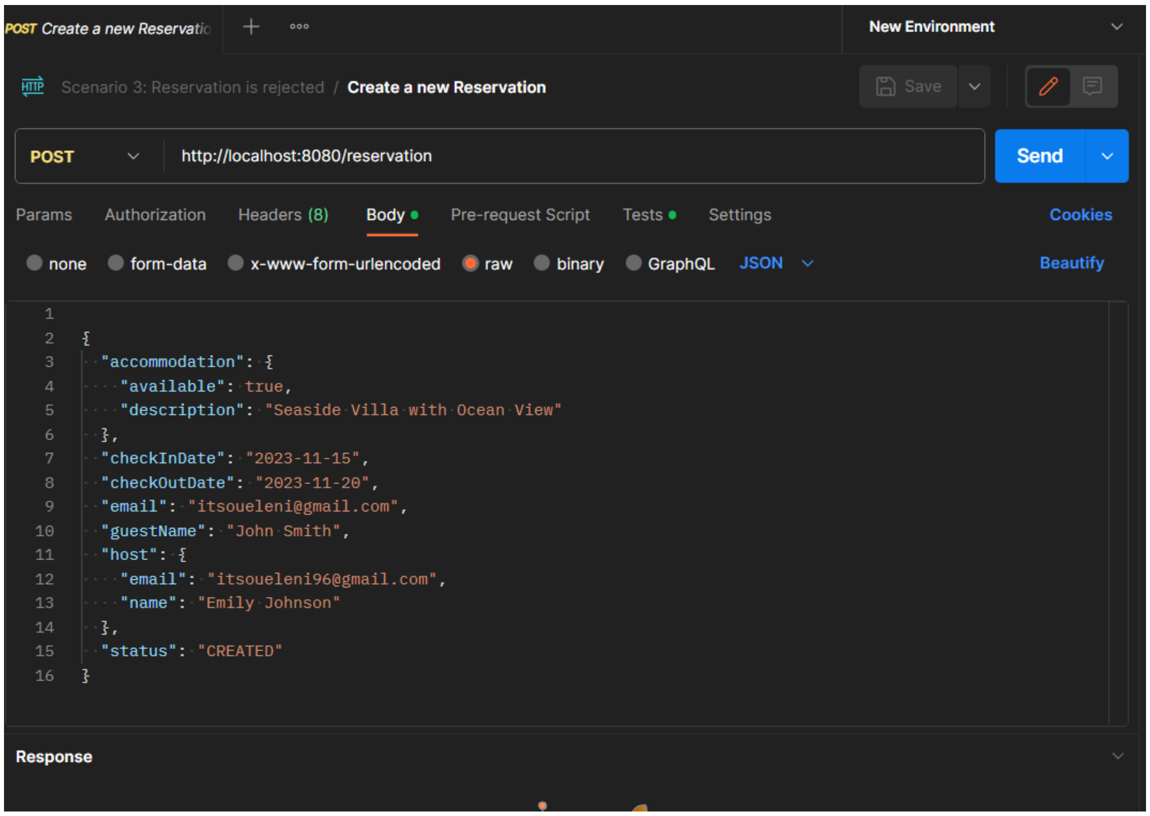
```
Status: 204 No Content   Time: 139 ms
```

Figure 100: Reservation non-availability business process scenario- Cancel reservation request method.

6.3 Reservation rejection process scenario

In this section, the scenario of reservation rejection is being explored. Rejection handling can be crucial in maintaining customer satisfaction. The steps of the workflow in the rejection process can be broken down as follows:

- Create a new reservation: Like previously the POST request method is used to initiating the reservation process. Request body includes all the necessary booking details. Similarly with the previous scenarios, the response status is tested and the location header of the response including the reservation id URL is stored as an environment variable (Figure 101).



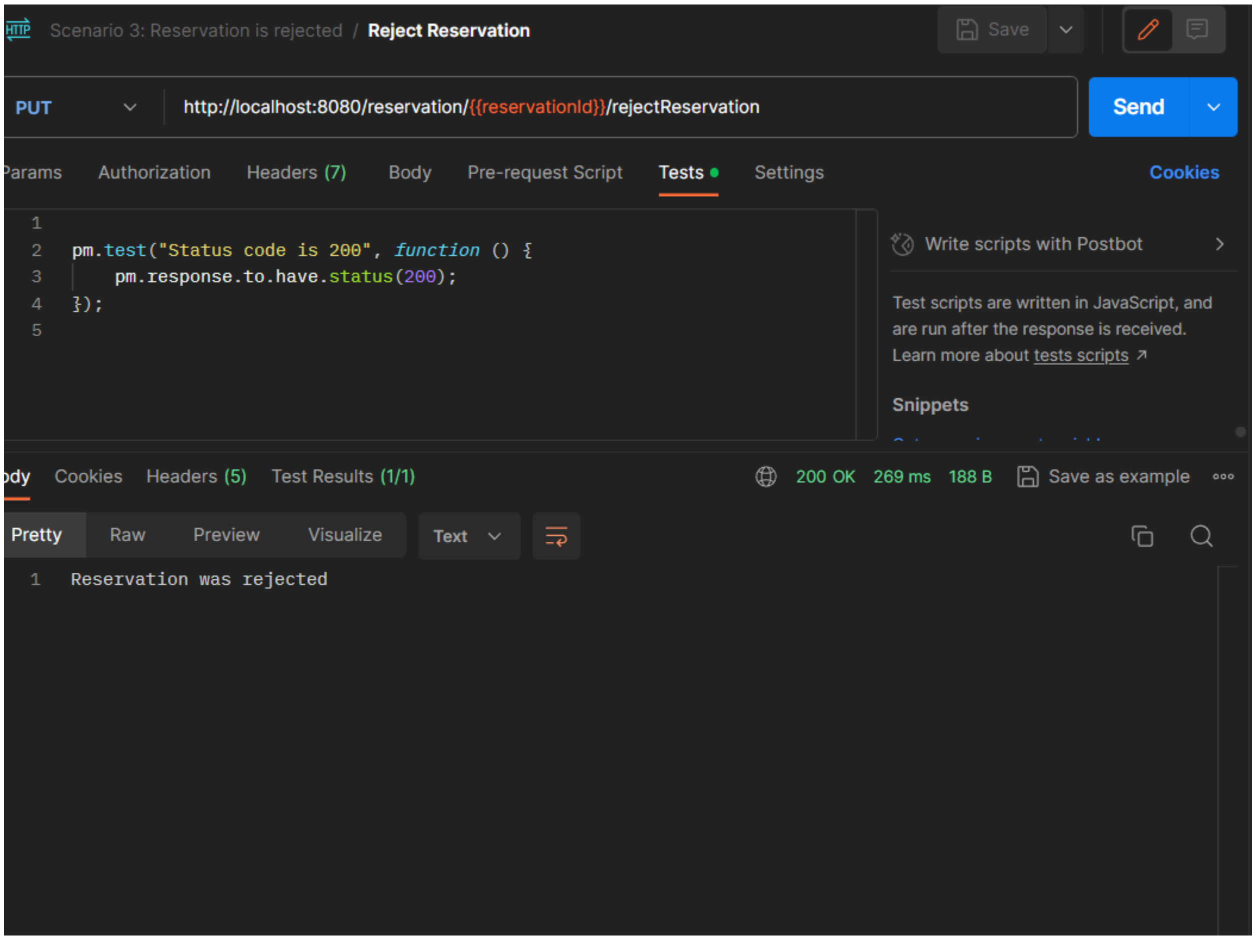
POST http://localhost:8080/reservation

```json

{
  "accommodation": {
    "available": true,
    "description": "Seaside Villa with Ocean View"
  },
  "checkInDate": "2023-11-15",
  "checkOutDate": "2023-11-20",
  "email": "itsoueleni@gmail.com",
  "guestName": "John Smith",
  "host": {
    "email": "itsoueleni96@gmail.com",
    "name": "Emily Johnson"
  },
  "status": "CREATED"
}
```

Figure 101: Reservation rejection business process scenario- test the create reservation POST method using Postman.

- Get reservation by id (Reservation status- Created): Following the reservation request creation, the new reservation id URL location is tested. This request retrieves information by the request initiated. Like in previous scenarios, the location variable is used as the request URL, and the response status is tested with the use of a test script.
- Update the reservation status: Similarly, to the happy path scenario by using the update request the availability is verified and the status of the request changes.
- Get reservation by id (Reservation status- Pending): Consequently, the updated status is verified by sending the get reservation id request and all the information about the reservation are retrieved.
- Send email of availability Guest/Host: The request method of sending an email is sent simultaneously to the guest and the host involved in the process. In both cases, the guestMessage, guestSubject, and guestEmail environment variables are used as request variables to the URL to construct the email, and the response status is tested. In the pre-request scripts of both the emails, the environment variables are being utilized for constructing the email message.

- Reject a reservation: Rejecting a reservation request is done with the following http://localhost:8080/reservation/{{reservationId}}/rejectReservation endpoint, that receives a PUT request. The reservation id request parameter is retrieved from the previous requests and the expected status code is verified via a test script (Figure 102).



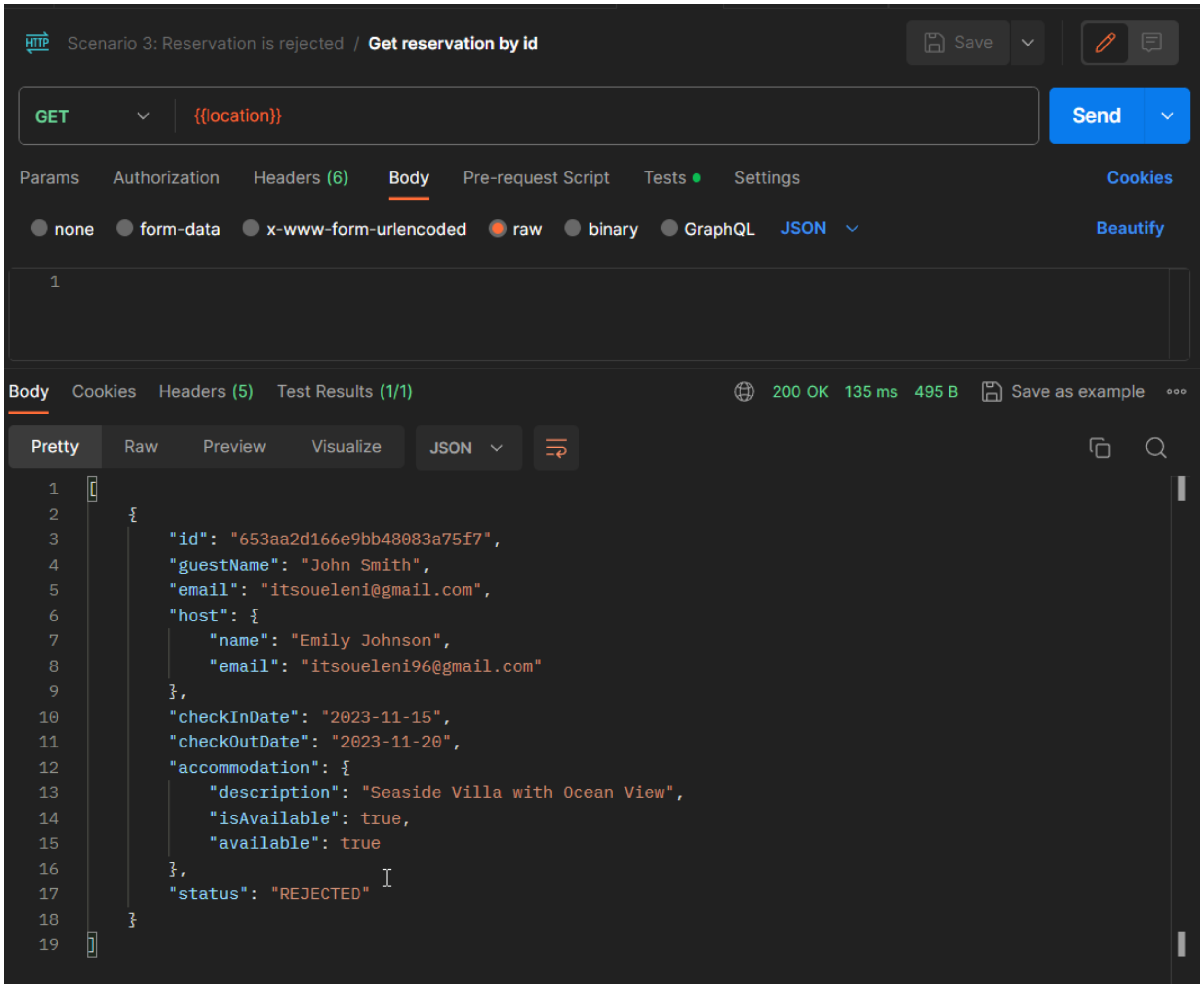
```

pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

```

Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts ↗

```
Reservation was rejected
```

Figure 102: Reservation rejection business process scenario- test reservation rejection PUT method using Postman.

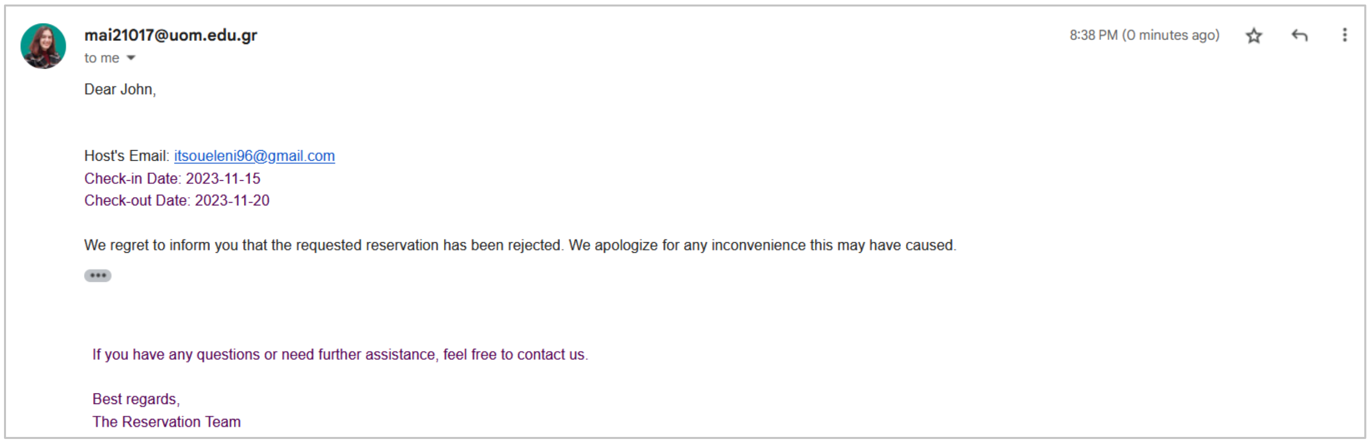
- Get reservation by id (Reservation status- Rejected): Following the previous request, the updated status is verified by sending the get reservation id request and all the information about the reservation are retrieved (Figure 103).



```
[
    {
        "id": "653aa2d166e9bb48083a75f7",
        "guestName": "John Smith",
        "email": "itsoueleni@gmail.com",
        "host": {
            "name": "Emily Johnson",
            "email": "itsoueleni96@gmail.com"
        },
        "checkInDate": "2023-11-15",
        "checkOutDate": "2023-11-20",
        "accommodation": {
            "description": "Seaside Villa with Ocean View",
            "isAvailable": true,
            "available": true
        },
        "status": "REJECTED"
    }
]
```

Figure 103: Reservation rejection business process scenario- test reservation by id rejected status.

- Send email of rejection to guest: Via this request method an email is informing the guest that their reservation has been declined. In the pre-request script section, the Message, subject, and recipient query parameters which are included in the request URL are defined as well as the environment variables are used to create the email content (Figure 104).
- Cancel reservation: The http://localhost:8080/reservation/{{reservationId}} endpoint receives a DELETE request. A testing script is used to verify that the 204 (No Content) is returned.



mai21017@uom.edu.gr
to me

8:38 PM (0 minutes ago)

Dear John,

Host's Email: itsoueleni96@gmail.com
Check-in Date: 2023-11-15
Check-out Date: 2023-11-20

We regret to inform you that the requested reservation has been rejected. We apologize for any inconvenience this may have caused.

If you have any questions or need further assistance, feel free to contact us.

Best regards,
The Reservation Team

Figure 104: Reservation rejection email to guest.

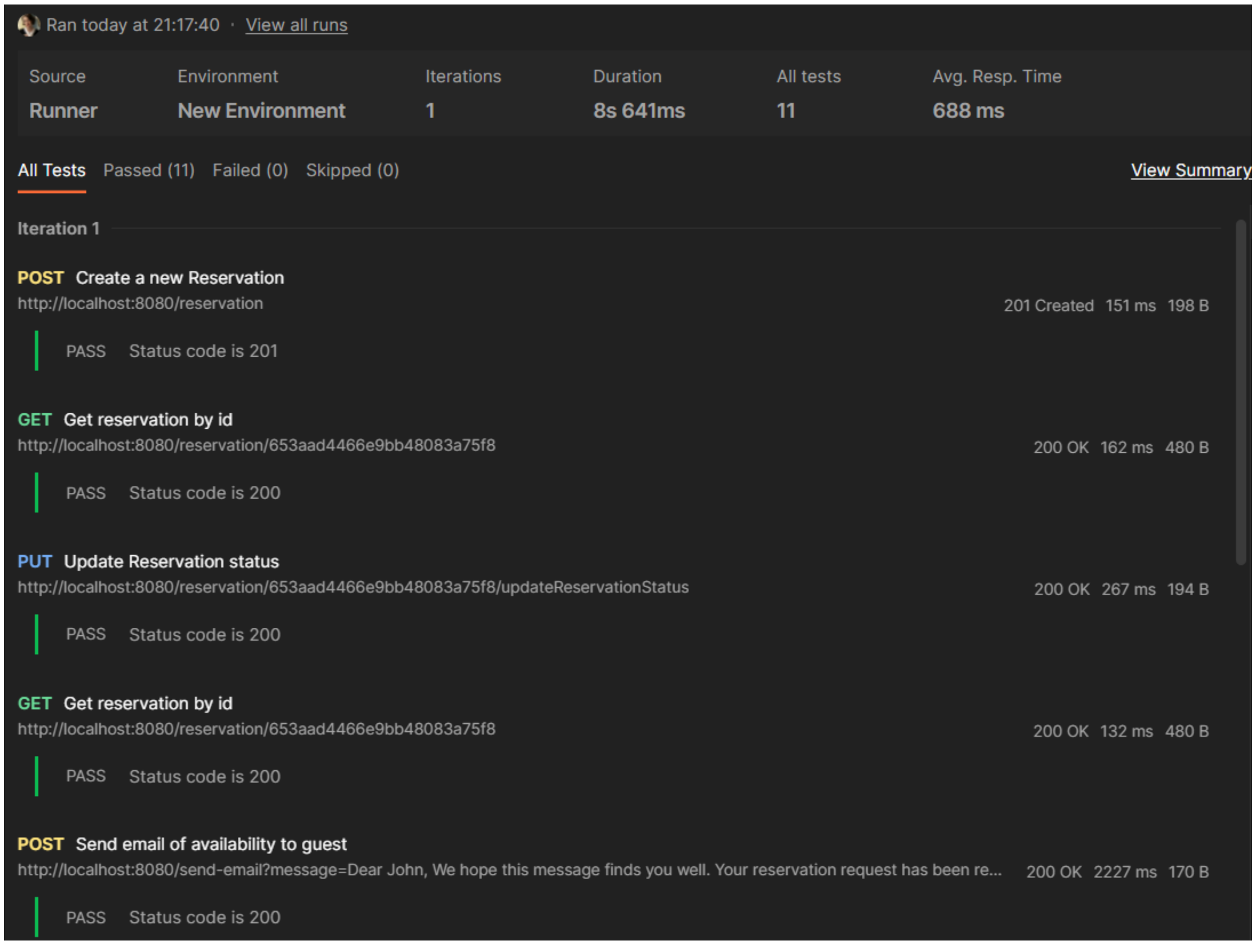
6.4 Chapter Summary

The chapter explores three separate reservation scenarios, each of which clarifies interactions between various API methods and acts as a performance indicator under different settings. The Reservation Confirmation Process Scenario (Happy Path) is a multi-step process that begins with utilizing a POST request to create a new reservation. The next stages are to confirm the reservation, update the reservation status, send email notifications to the guest and host, and enter the reservation into the system. Every step is verified by test scripts.

The following scenario is the Reservation Non-Availability Process. This example shows how a reservation request for an unavailable accommodation is handled by the system. The process involves establishing a reservation, monitoring its progress, making necessary updates, notifying the guest via email about non-availability, and ultimately canceling the reservation. To make sure the anticipated results occur, each step is evaluated.

Subsequently the reservation rejection process scenario focuses on the steps involved when a reservation request has been turned down. Making a new reservation and checking its status comes first. Email notifications are sent to both the host and the guest, and the reservation status is updated. Following the cancellation of the reservation, the updated status is verified, and the guest receives a refusal email.

In addition, API request workflow included in each scenario can run automatically using postman. This functionality can be vital in finding problems and verifying system functionality, due to extensive test reporting, script integration, status code verification, and response validation features that are presented. As a result, when a collection is automatically run, a report with the test results is presenting verifying that the API workflow works as expected.



Ran today at 21:17:40 · View all runs

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	New Environment	1	8s 641ms	11	688 ms

All Tests   Passed (11)   Failed (0)   Skipped (0)   View Summary

Iteration 1

```
POST  Create a new Reservation
http://localhost:8080/reservation                                            201 Created  151 ms  198 B
    PASS    Status code is 201

GET  Get reservation by id
http://localhost:8080/reservation/653aad4466e9bb48083a75f8                   200 OK  162 ms  480 B
    PASS    Status code is 200

PUT  Update Reservation status
http://localhost:8080/reservation/653aad4466e9bb48083a75f8/updateReservationStatus   200 OK  267 ms  194 B
    PASS    Status code is 200

GET  Get reservation by id
http://localhost:8080/reservation/653aad4466e9bb48083a75f8                   200 OK  132 ms  480 B
    PASS    Status code is 200

POST  Send email of availability to guest
http://localhost:8080/send-email?message=Dear John, We hope this message finds you well. Your reservation request has been re...   200 OK  2227 ms  170 B
    PASS    Status code is 200
```

Figure 105: Run collection summary.

In conclusion, Postman is a priceless instrument for fully testing and automating API procedures. In order to ensure the effective testing of the reservation system's operation and a favorable user experience, these three scenarios depict real-world cases, ranging from successful reservations to handling non-availability and rejection.

Chapter 7 : Discussions and Conclusions

7.1 Thesis Overview

Starting with an historical analysis, this research project follows the development of business process management (BPM) from Total Quality Management and Six Sigma Techniques to Business Process Reengineering (BPR) and continuous improvement strategies. Moreover, it demonstrates how the rise of BPM, particularly with the introduction of IT systems such as Workflow Management Systems and Enterprise Resource Planning (ERP) systems, led to the development of Process-Driven Applications (PDAs). This thesis also emphasizes how crucial it is for developers, business analysts, and domain experts to work together while implementing process-driven applications. It draws attention to how process engines are used to manage service orchestration and human workflow.

Moving forward, the importance of collaboration and choreography diagrams is explored to illustrate participant interactions within a BPMN framework. Collaboration diagrams clarify the relationships between various parties in a business process. White-box collaboration diagrams shed light on each participant's internal operations. On the other hand, choreography diagrams concentrate on participant interactions. They function as contracts that specify the messages to be sent and received, focusing mainly on the message exchange and the logical progression of these interactions.

The booking procedure, which is inspired by Airbnb, serves as a real-world example to demonstrate these ideas. Through both collaboration and choreography diagrams, the research demonstrates how a reservation application serves as a coordinator between guests and hosts. The choreography diagram focuses on message exchanges between participants without digging into their underlying processes, but the collaboration diagram clarifies the specific flow of interactions between parties.

This thesis also investigates how crucial REST APIs are to the application creation, especially in terms of scalability, flexibility, and portability. The study introduces the idea of REST annotations inside a business process and goes on to examine how REST APIs might be integrated into BPMN choreographies. In addition, HTTP request/response exchanges and email notifications can be represented using an example involving a reservation application, emphasizing the importance of links in these interactions. It is also highlighted that RESTful interactions into BPMN choreography diagrams facilitate efficient description and visualization of interactions, bridging the gap between RESTful API implementation and process modeling. The result is the construction of reliable, interoperable, and effective applications, offering valuable guidance to developers and business analysts.

The Model-View-Controller (MVC) design pattern is used in a separate chapter to construct a Reservation REST API. The API is divided into four layers: the Repository layer, the Service layer, which handles business logic, and the Model layer, which represents the organization and entities of the application. As a build automation tool, Apache Maven is used to streamline dependency management and project development. The chapter offers in-depth explanations of how to create a project skeleton with the required dependencies, set up a REST API, use Spring Web and Spring Data MongoDB requirements to connect to the database, and use Spring Fox Swagger UI to build API documentation.

The Reservation REST API's endpoints and their functions are all summarized in a summary table. The study examines three reservation scenarios, each depicting interactions between different API operations. The thesis emphasizes how important it is to use Postman to automate API request workflows. This tool, which makes use of comprehensive test reporting, script integration, status code verification, and response validation features, is extremely helpful in locating problems and confirming system functionality. The automatic execution of a collection ensures that the API workflow operates as intended by producing a report with test results.

This study's findings emphasize the importance of Postman as a tool for thoroughly testing and automating API operations, which ultimately helps to ensure the smooth operation of the reservation API in conjunction with email notifications provided by JavaMail API. The investigation covers everything from the fundamental ideas of business process management (BPM) to collaboration and choreography diagrams utilization in the reservation REST API development and documentation.

7.2 Research Contribution

The current work has presented an innovative method for augmenting BPMN choreography diagrams with REST-specific annotations, which facilitates the transfer from choreography models to RESTful HTTP conversations. This method ensures compatibility with current modeling tools, and adherence to the formal specification of the BPMN standard by employing annotations to capture additional information.

In addition, important insights are gained from the representation of participant interactions using collaboration and choreographic diagrams. The study also emphasizes the value of REST APIs for developing applications and provides helpful advice on how to integrate them with BPMN choreographies. The usage of Postman automation and the Model-View-Controller (MVC) paradigm are examples of best practices for developing APIs.

Moreover, the integration of email notifications further enhances the understanding of application communication. These contributions collectively advance knowledge and practices in BPM and modern application development.

7.3 Research Limitations and Future Work

Even though it offers insightful information this study has a few limitations. To begin with, the example study used in the research is a reservation application, which might not accurately reflect the complexity of other business processes. Furthermore, although being a crucial factor in practical applications, the study does not go into detail about API security and the possible privacy issues related to REST API integration.

Future research could focus on several areas to expand on this study. First of all, the scope of the case studies can be broadened to more sectors as this approach can be expanded to various scenarios such as e-commerce where HTTP requests and email notification play a vital role. Specifically, research can focus on identifying the stakeholders involved in the transition from choreography models to executable REST-based implementations. Understanding the roles and responsibilities of various stakeholders will further facilitate this transition process, offering a more comprehensive framework for the integration of RESTful interactions within BPMN choreography diagrams.

Furthermore, given the growing significance of privacy and data protection laws, investigating the security implications, and creating best practices for safe REST API integration in BPM are essential. Additionally, to make sure that BPM and REST API integration methods stay current and useful, it's critical to stay up to date on developing technologies and market trends.

7.4 Conclusions

This study explored the integration of REST APIs with BPMN choreographies and introduced REST annotations into a business process. It also demonstrated the critical function of links in email notifications and HTTP request/response exchanges using a real-world scenario involving a reservation application.

Moreover, this study showcased how RESTful interactions might be used to bridge the gap between process modeling and RESTful API implementation by enabling the efficient description and visualization of interactions when incorporated into BPMN choreography diagrams. Furthermore, a whole chapter was devoted to building a Reservation REST API using the Model-View-Controller (MVC) design pattern. The four layers of this API were the Model layer, which represented the entities and structure of the application, the Service layer, which managed business logic, and the Repository layer, which handled database communication. Apache Maven, a powerful build automation tool, to expedite project development and dependency management.

The overview of the Reservation REST API's endpoints and features, as demonstrated by three reservation scenarios, is the study's product. The focus was on leveraging Postman, a program praised for its extensive testing and validation features, to automate API request workflows. The API operated flawlessly thanks to the automated test collection execution, which also produced insightful test reports for validation.

In conclusion, this study tour has illuminated the complex domain of BPM, choreography and collaboration diagrams, RESTful APIs, and how they can be incorporated into application development.

Bibliography

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-33143-5.

[2] K. Vergidis, 'Business process optimisation using an evolutionary multi-objective framework', Cranfield University, 2008.

[3] E. A. Stohr and J. L. Zhao, 'Workflow Automation: Overview and Research Issues', *Information Systems Frontiers*, vol. 3, no. 3, pp. 281–296, 2001, doi: 10.1023/A:1011457324641.

[4] M. Hammer, J. Vom Brocke, and M. Rosemann, 'What is business process management? Handbook on business process management 1', *are J. v. Brocke and M. Rosemann. Berlin, published in Springer*, 2010.

[5] J. F. Chang, *Business Process Management Systems: Strategy and Implementation*, 1st ed. Auerbach Publications, 2005. doi: 10.1201/9781420031362.

[6] S. Conger, 'Six Sigma and Business Process Management', in *Handbook on Business Process Management 1*, Springer Berlin Heidelberg, 2010, pp. 127–148. doi: 10.1007/978-3-642-00416-2_6.

[7] S. I. Chang, D. C. Yen, C. C. Chou, H. C. Wu, and H. P. Lee, 'Applying six sigma to the management and improvement of production planning procedure's performance', *Total Quality Management and Business Excellence*, vol. 23, no. 3–4, pp. 291–308, 2012, doi: 10.1080/14783363.2012.657387.

[8] M. Hammer, 'Reengineering work: Don't automate, obliterate', *Harv Bus Rev*, vol. 68, no. 4, pp. 104–112, Jul. 1990.

[9] M. Hammer and J. Champy, *Reengineering the Corporation: A Manifesto for Business Revolution*. New York: Harper Collins, 1993.

[10] J. vom Brocke and M. Rosemann, 'Business Process Management', in *Wiley Encyclopedia of Management*, 2015, pp. 1–9. doi: https://doi.org/10.1002/9781118785317.weom070213.

[11] K. P. McCormack, 'The development of a measure of business process orientation and its link to the interdepartmental dynamics construct of market orientation', *Nova Southeastern University ProQuest Dissertations Publishing*, 1999.

[12] T. H. Davenport and J. E. Short, 'The New Industrial Engineering: Information Technology and Business Process Redesign', *Sloan Manage Rev*, vol. 31, no. 4, pp. 11–27, 1990.

[13] K. P. McCormack, *Business process maturity: theory and application*. North Carolina , 2007.

[14] M. Hammer and S. Steve, 'How Process Enterprises Really Work', *Harv Bus Rev*, vol. 77, no. 6, pp. 108–118, Nov. 1999.

[15] J. vom Brocke and J. Mendling, *Business Process Management Cases*. Springer International Publishing, 2018. doi: 10.007/978-3-319-58307-5.

[16] W. M. P. van der Aalst, 'Business Process Management: A Comprehensive Survey', *ISRN Software Engineering*, vol. 2013, pp. 1–37, Feb. 2013, doi: 10.1155/2013/507984.

[17] A. Keller and J. Moormann, 'The challenge to determine a company's process maturity: a case study from the financial services industry', *European Journal of Management Issues*, vol. 25, no. 2, pp. 85–91, Jun. 2017, doi: 10.15421/191712.

[18] B. Rücker and J. Freund, *Real-Life BPMN* , (4th edition). 2019.

[19] E. Schäffer, V. Stiehl, P. K. Schwab, A. Mayr, J. Lierhammer, and J. Franke, 'Process-Driven Approach within the Engineering Domain by Combining Business Process Model and Notation (BPMN) with Process Engines', *Procedia CIRP*, vol. 96, pp. 207–212, Jan. 2021, doi: 10.1016/J.PROCIR.2021.01.076.

[20] V. Stiehl, R. Raw, and P. Smith, *Process-driven applications with BPMN*. 2014. doi: 10.1007/978-3-319-07218-0.

[21] E. ; Schäffer, S. ; Shafiee, T. ; Frühwald, and J. Franke, 'General rights A development approach towards user-centered front-ends for knowledge-based engineering configurators: a study within planning of robot-based automation solutions'.

[22] M. Chen, D. Zhang, and L. Zhou, 'Empowering collaborative commerce with Web services enabled business process management systems', *Decis Support Syst*, vol. 43, no. 2, pp. 530–546, Mar. 2007, doi: 10.1016/J.DSS.2005.05.014.

[23] G. Aagesen and J. Krogstie, 'BPMN 2.0 for Modeling Business Processes', *Handbook on Business Process Management 1: Introduction, Methods, and Information Systems*, pp. 219–250, Mar. 2015, doi: 10.1007/978-3-642-45100-3_10.

[24] M. Chinosi and A. Trombetta, 'BPMN: An introduction to the standard', *Comput Stand Interfaces*, vol. 34, no. 1, pp. 124–134, Jan. 2012, doi: 10.1016/J.CSI.2011.06.002.

[25] P. and A. D. Genon Nicolas and Heymans, 'Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation', in *Software Language Engineering*, S. and van den B. M. Malloy Brian and Staab, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 377–396.

[26] 'Camunda Academy BPMN 2.0 course'. Accessed: Apr. 10, 2023. [Online]. Available: https://academy.camunda.com/camunda-bpmn

[27] GRNET, 'National Registry of Administrative Public Services MITOS', Hellenic Republic Ministry of Interior.

[28] A. Nikaj, S. Mandal, C. Pautasso, and M. Weske, 'From Choreography Diagrams to RESTful Interactions', May 2016, pp. 3–14. doi: 10.1007/978-3-662-50539-7_1.

[29] M. Weske, *Business Process Management Concepts,Languages,Architectures*. Berlin: Springer, 2007. Accessed: May 14, 2023. [Online]. Available: http://www.untag-smd.ac.id/files/Perpustakaan_Digital_1/BUSINESS%20Business%20Process%20Management%20Concepts,%20Languages,%20Architectures.pdf

[30] A. Nikaj, M. Weske, and J. Mendling, 'Semi-automatic derivation of RESTful choreographies from business process choreographies', *Softw Syst Model*, vol. 18, no. 2, 2019, doi: 10.1007/s10270-017-0653-2.

[31] Orbus Software, 'BPMN Diagrams: Collaboration'. Accessed: Sep. 09, 2023. [Online]. Available: https://www.youtube.com/watch?v=lotrvHYtGhk

[32] J. Ladleif, A. Von Weltzien, and M. Weske, 'chor-js: A Modeling Framework for BPMN 2.0 Choreography Diagrams', 2019. [Online]. Available: https://github.com/bpmn-io/bpmn-js

[33] Amit Rana, 'Airbnb – Hospitality & Accommodation Redefined', Code Brew Labs. Accessed: Oct. 15, 2023. [Online]. Available: https://www.code-brew.com/airbnb-hospitality-accommodation-redefined/

[34] 'MDN Web Docs Glossary: Definitions of Web-related terms ', MDN Web Docs. Accessed: May 15, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/API

[35] R. Fielding, 'Architectural Styles and the Design of Network-based Software Architectures', 2000.

[36] W3 Schools, 'XML SOAP'. Accessed: Oct. 07, 2023. [Online]. Available: https://www.w3schools.com/xml/xml_soap.asp#:~:text=A%20SOAP%20Example,example.org%2Fstock%22.

[37] F. Haupt, F. Leymann, and C. Pautasso, 'A Conversation Based Approach for Modeling REST APIs', in *Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015*, Institute of Electrical and Electronics Engineers Inc., Jul. 2015, pp. 165–174. doi: 10.1109/WICSA.2015.20.

[38] Camunda blog, 'What is REST API in Java? Guide with Examples', Camunda Organization. Accessed: Sep. 10, 2023. [Online]. Available: https://camunda.com/blog/2023/09/what-is-rest-api-in-java-guide-with-examples/

[39] T. Barton and C. Seel, 'Business process as a service-status and architecture', *Enterprise modelling and information systems architectures-EMISA 2014*, 2014.

[40] A. Nikaj, 'RESTful Choreographies', Hasso Plattner Institute, University of Potsdam, Potsdam, Germany, 2019.

[41] Tom Collings, 'Controller-Service-Repository'. Accessed: Sep. 25, 2023. [Online]. Available: https://tom-collings.medium.com/controller-service-repository-16e29a4684e5

[42] 'Apache Maven- Feature Summary'. Accessed: Sep. 28, 2023. [Online]. Available: https://maven.apache.org/maven-features.html

[43] 'Maven Directory Structure'. Accessed: Sep. 28, 2023. [Online]. Available: https://www.dineshonjava.com/maven-directory-structure/

[44] Oracle, 'JavaMail'. Accessed: Oct. 10, 2023. [Online]. Available: https://www.oracle.com/java/technologies/javamail.html