End-Of-Studies Project

In partial fulfillment of the requirements for a

Master's Degree

In

Software Engineering

Presented by

Riadh Laabidi

Notification Center

Defended on September 2023 in front of the committee members:

Mr.	President
Mrs.	Examiner
Mrs. Asma Kerkeni	Thesis Supervisor
Mr. Fahmi Boumaiza	Professional Supervisor

————— **Academic Year: 2022 / 2023** —————

For everyone who has ever been a part of my life.

I dedicate this work.

Abstract

This report outlines the design and implementation of a notification center as a part of the "Convergence" project, which aims to drive digital transformation within the insurance and finance sectors while improving client engagement and optimizing internal processes. The notification center provides a scalable solution for managing and delivering tailored notifications through various channels, facilitating timely and personalized interactions. This project was made during an end-of-studies internship in order to obtain the Master's degree in Software Engineering at the Higher Institute of Informatics and Mathematics of Monastir (ISIMM).

Keywords: Client Engagement, Notification Channels, User Segmentation, Spring Boot, Angular, PostgreSQL, Docker.

Acknowledgment

I would like to express my sincere gratitude and appreciation to my academic supervisor, **Mrs. Asma Kerkeni**, for her invaluable guidance, insightful advice, and unwavering support throughout this internship journey. Her expertise, enthusiasm, and dedication have been a constant source of inspiration and motivation.

I would also like to extend my deepest thanks to my technical supervisor, **Fahmi Boumaiza**, for his technical mentorship, and guidance throughout the project. His knowledge and expertise in software development were of great help in shaping this project and ensuring its success, and I have learned a lot from him.

I am grateful to both my supervisors for providing me with the opportunity to work on this challenging and exciting project, which has enriched my knowledge and skills in software development. Their support and mentorship have been crucial to my academic and professional growth.

Finally, I would like to thank the entire team at **Satoripop** for their cooperation, encouragement, and support throughout this journey. Their insightful feedback and suggestions made my work enjoyable and productive.

Contents

List of Figures

List of Tables

Acronyms

CI/CD Continuous Integration and Continuous Delivery.

CRUD Create, Retrieve, Update, and Delete.

CTR Click-through Rates.

DAO Data Access Object.

HTTP Hypertext Transfer Protocol Secure.

HTTPS Hypertext Transfer Protocol Secure.

JPA Java Persistence API.

JSON JavaScript Object Notation.

MVC Model-View-Controller.

MVVM Model-View-ViewModel.

SMTP Simple Mail Transfer Protocol.

UML Unified Modeling Language.

Introduction

In today's rapidly evolving digital landscape, the insurance, banking, and finance sectors are facing unprecedented challenges and opportunities. As customers increasingly expect seamless and personalized experiences, it has become imperative for companies in these industries to undergo a digital transformation. This shift not only enhances customer engagement and satisfaction but also boosts operational efficiency and employee productivity.

Recognizing this need, the "Convergence" project was initiated at Satoripop to provide innovative solutions that facilitate the digital transformation process, enabling organizations to stay competitive, deliver exceptional customer experiences, and achieve sustainable growth in the ever-changing marketplace.

Within the "Convergence" ecosystem, various tools and sub-projects are assembled, each playing a crucial role in this shared mission. Among these modules, we will be focusing on the Notification Center, as it plays a pivotal role in enhancing communication by providing a centralized system for timely and relevant notifications.

This report is structured in several chapters to provide a comprehensive overview of the project's progression. The first chapter introduces the project context, including a profile of the host organization and an overview of the project itself. The challenges and problems faced in the insurance and finance sectors are also assessed.

The second chapter discusses the initial sprint of the project, which revolves around gathering requirements, defining specifications and setting the groundwork to start the project execution.

Subsequent chapters delve into the specific releases of the project, outlining the specification, design and implementation steps. Each chapter provides a summary of the key findings and progress made during the respective phase.

Chapter I

Project Context

Introduction

This chapter presents the project context, including an overview of the host organization, the project objectives and the problem assessment. Afterwards, we will benchmark some existing solutions, and select a suitable work methodology for a successful project execution. By exploring the context surrounding the project, we aim to provide a deeper understanding of its relevance within the organization and the industry as a whole.

I.1 Host organization profile

Satoripop is a custom software development house that delivers a wide range of end-to-end, reliable software services and solutions for businesses across various industry verticals. Satoripop offers its services in many countries in Europe, Middle East and Africa.

Figure I.1: Satoripop Logo

Satoripop has structured its offering around four key service areas. As a solutions provider, and given that all interactions nowadays pass through IT systems, Satoripop is more than ever at the heart of customers' business.

Table I.1: Host Organization Details

Services	• **Custom development:** Web applications, Mobile applications, Application modernization • **Design UX/UI:** SEO, Netlinking, Optimization and assistance with content creation, Reporting • **Consulting & Framing:** Design thinking, Customer journey map, Wireframing, Prototypage • **Digital Marketing:** SEO, Netlinking, Optimization and assistance with content creation, Reporting
Phone number	Tunisia: +216 73 210 332
Address	Satoripop MEA, Blvd Hassouna Ayachi, Sousse 4000, Tunisia
Website	`https://www.satoripop.com`

I.2 Project overview

A notification center is a system that allows businesses to send and manage notifications to their customers or employees. The value proposition of a notification center is that it can help businesses communicate with their stakeholders more efficiently and effectively, leading to increased engagement and productivity.

In the retail sector as an example, a company could use the notification center to send SMS, email, push or chat alerts about product recalls, special promotions, or other important information to customers. By using a notification center, businesses can improve communication with their customers and ensure that important information is quickly and effectively distributed.

I.3 Problem assessment and challenges

Implementing a notification center system in the retail sector poses certain challenges and requires a comprehensive assessment of the existing scenario. The following issues were identified during the evaluation:

- **Limited Communication Channels:** Many businesses in the retail sector rely heavily on traditional communication methods such as flyers, physical notices, or in-store announcements. These methods often lack efficiency, reach, and real-time delivery, resulting in delayed or ineffective communication.

- **Information Overload:** Businesses need to disseminate various types of notifications, like product recalls or special promotions. However, the challenge is to manage and prioritize notifications to prevent overwhelming customers with exces-

sive information.

- **Personalization and Targeting:** Effective communication requires tailoring notifications to specific customer segments or individuals. The challenge lies in ensuring that customers receive relevant and personalized notifications based on their preferences, previous interactions, or location.

- **Multichannel Delivery:** Customers today expect to receive notifications through various channels such as SMS, email, push notifications, or social media. Managing multiple communication channels and ensuring consistent and synchronized delivery presents a significant challenge.

- **Privacy and Data Security:** With the increasing concerns about privacy and data protection, businesses must handle customer data securely while complying with privacy regulations. Safeguarding customer information and maintaining trust is crucial in implementing a notification center system.

I.4 Competitor benchmarking

In this section, we will evaluate and compare existing notification delivery solutions. By examining the features, functionalities, and performance of these solutions, we aim to identify the strengths and weaknesses of each competitor in order to inform our own development process.

I.4.1 Competitors selection

During our search for similar existing functionalities, we have carefully evaluated numerous solutions and identified the three most closely aligned with our specific needs. We will present an overview of these chosen solutions, highlighting their key characteristics and capabilities.

I.4.1.1 OneSignal

OneSignal is a popular notification service that supports multiple platforms including iOS, Android, and web. OneSignal offers a wide range of features, including segmenting users based on custom attributes, A/B testing, automation, and personalization. OneSignal also provides real-time analytics and delivery reports, allowing tracking and optimization of notification campaigns.
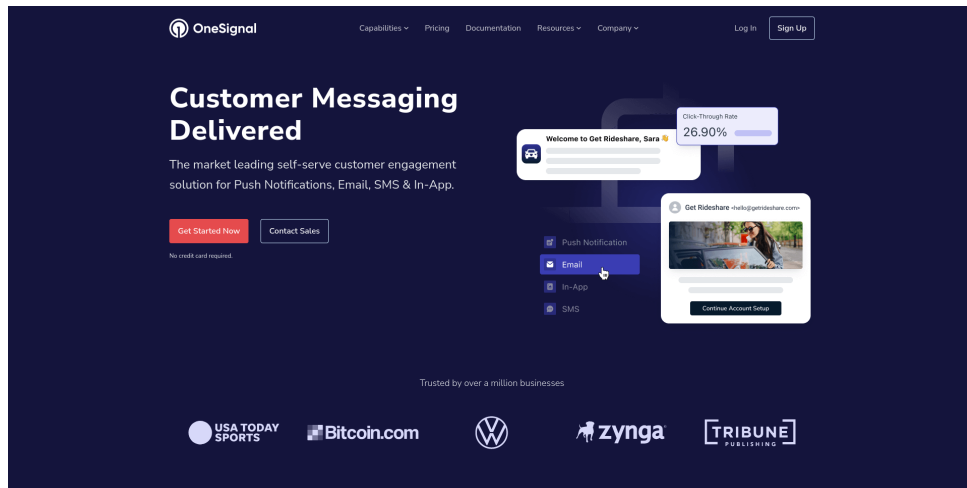
Figure I.2: OneSignal Homepage [4]

I.4.1.2 Pusher

Pusher is a cloud-based notification service that provides real-time messaging for mobile and web applications. It supports push notifications, in-app notifications, and allows integration with other services and platforms.
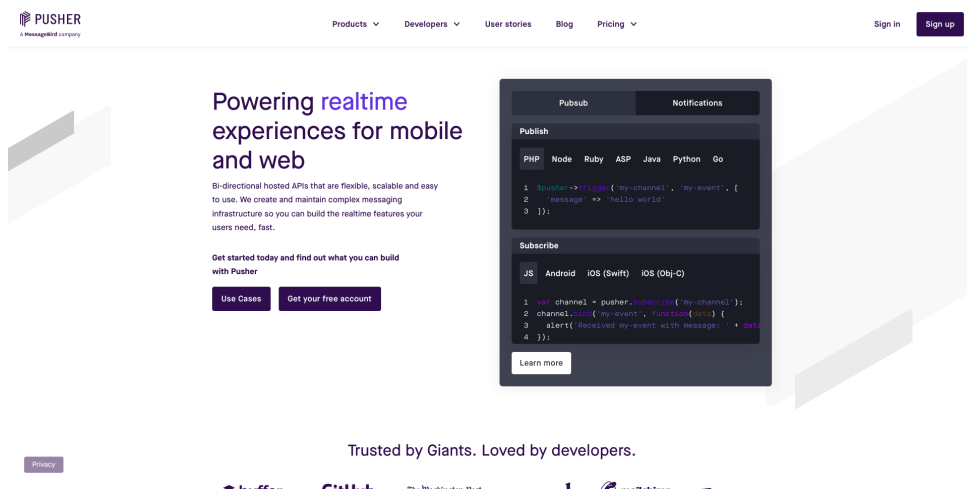
Figure I.3: Pusher Homepage [2]

I.4.1.3 Amazon SNS

Amazon SNS (Simple Notification Service) is a fully managed messaging service provided by Amazon Web Services (AWS) that enables you to send messages or notifications to a variety of distributed endpoints or clients.
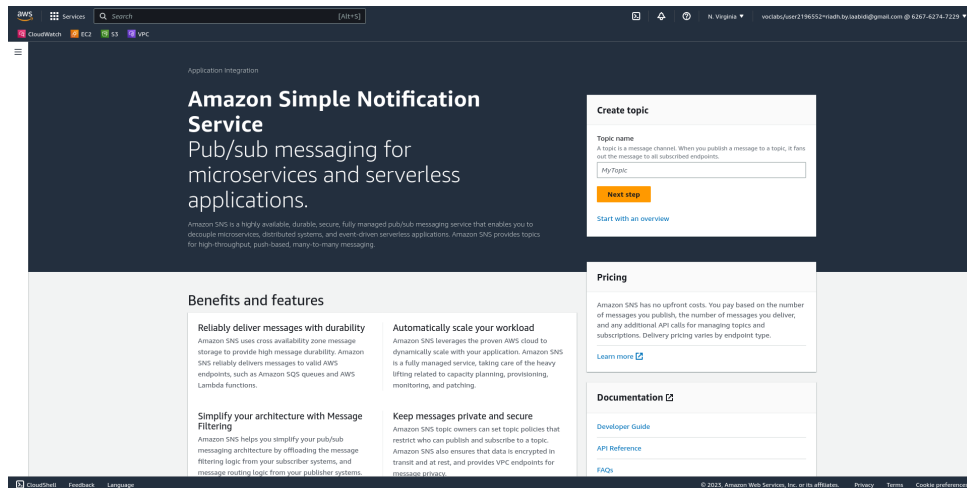
Figure I.4: Amazon SNS Service Homepage [1]

I.4.2   Comparison

We will conduct a comprehensive analysis of the selected existing solutions. By thoroughly examining their functionalities, strengths, and weaknesses, we aim to gain valuable insights into areas where improvements can be made to enhance user experience, streamline processes, and deliver a superior solution tailored to our specific requirements.

Our evaluation will be based on 6 criterias (Cx) that have been carefully selected as being the most relevant and aligned with the Convergence project requirements:

- **C1: Multi-channel delivery:** The ability to configure multiple channels (Email, SMS, Push, Chat) for notification delivery.

- **C2: Deliverability:** Ensuring that notifications and alerts are successfully delivered to users' devices.

- **C3: User Segmentation:** The ability to select and target specific groups of users based on criterias defined by the business requirements.

- **C4: Customization:** The ability to customize notifications content for targeted users accordingly.

- **C5: Analytics:** The ability to track different metrics during the delivery process in order to inform businesses about the effectiveness of notification campaings and improve user engagement.

- **C6: Cost:** Charge per sent notifications for a larger user base and high notification volume.

6

- **C7: Ease of use:** The ability to easily setup and configure the different steps of the notification delivery process for non technical users.

Table I.2: Comparison table

	OneSignal	**Pusher**	**Amazon SNS**
C1 Multi-channel delivery	Provides most of channels (Push, Email, SMS), except for the Chat channel	Provides Push (Android, iOS, Web) notifications only	Provides the ability to send email, SMS, and push notifications
C2 Deliverability	Low Click-through Rates (CTR) that drops over time due to inactive receivers leading to misinformation for senders about the deliverability of notifications.	Relies on a retry mecanism for failed sending	Sets an internal delivery retry policy to 50 times over 6 hours, for Simple Mail Transfer Protocol (SMTP), SMS, and mobile push endpoints
C3 User Segmentation	Provides a user segmentation feature based on various criteria	Does not provide a user segmentation feature	Does not support targeting specific segments of users
C4 Customization	**Limited**: Users can't fully customize the layout of the notification	**Limited**: Only a subject and a body, no dynamic content	**Limited**: Emails are intended to provide internal system alerts, not marketing messages.
C5 Analytics	Provides insights dashboard	Provides simple metrics, relies on third party services for in-depth analytics	No detailed insights on sent notifications, relies on other AWS services for metrics
C6 Cost	**249$/month** (for 50,000 subscribers from push notifications only)	**99$/month** for 50,000 subscribers	Charge per usage (Pay as you go)
C7 Ease of use	Mostly easy to use for non technical users	Mostly Programmatic, does not provide a fully featured web interface	Complex and time-consuming particularly for non technical users

I.5 Proposed solution

The benchmarking process allowed us to identify pain points and challenges commonly faced by businesses when implementing a notification center system. These challenges include the need for multi-channel notification delivery to ensure broad customer base reach, the demand for tailored and customized notifications to engage specific customer group effectively, the importance of gathering accurate analytics to measure the impact and inform businesses about the effectivenes of notifications campaings.

Based on these findings, we have concluded that developing a custom notification center tailored to our specific needs is the most suitable approach. By addressing the identified pain points and challenges, we can create a robust and user-friendly solution that empowers businesses to communicate more efficiently with their customers and employees.

The proposed notification center will incorporate multi-channel delivery options, advanced user segmentation, and personalized messaging capabilities. It will also prioritize customization and prioritization, enabling businesses to tailor notifications based on audience preferences and deliver essential information effectively.
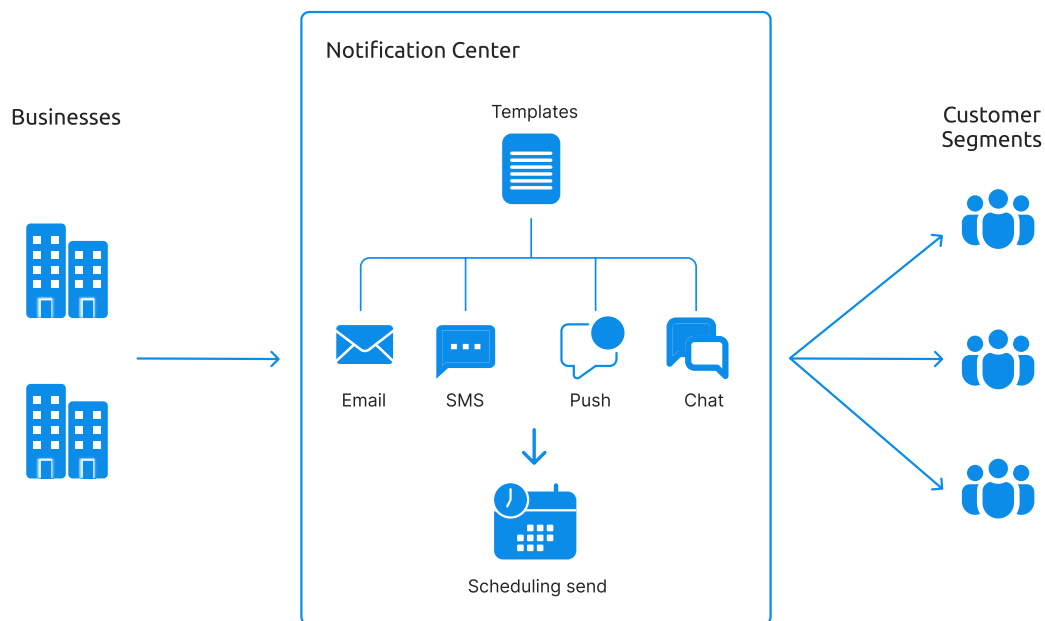
Figure I.5: Proposed solution illustration

I.6 Work methodology

With today's customers and businesses requiring rapid responses and changes, teams at Satoripop are embracing the Agile methodology, a project management approach that involves breaking the project into phases and emphasizes continuous collaboration and improvement, following a cycle of planning, executing, and evaluating.

One of the frameworks that helps practice building the mentioned agile priciples into work and that we will be focusing on is Scrum. we will provide an overview of this framework, highlighting its approach and key principles.

I.6.1 The scrum framework

Scrum is a lightweight framework that helps people, teams and organizations generate value through adaptive solutions for complex problems. It employs an iterative, incremental approach to optimize predictability and to control risk, while engages groups of people who collectively have all the skills and expertise to do the work and share or acquire such skills as needed.

I.6.2 Members of a scrum team

The fundamental unit of Scrum is a small team of people. The Scrum Team consists of one Scrum Master, one Product Owner, and Developers. Within a Scrum Team, there are no sub-teams or hierarchies. It is a cohesive unit of professionals focused on one objective at a time, the Product Goal [5].

- **Developers:** People in the Scrum Team that are committed to creating any aspect of a usable Increment each Sprint.

- **Product Owner:** One person that may represent the needs of many stakeholders in the Product Backlog. The Product Owner is accountable for effective Product Backlog management and maximizing the value of the product resulting from the work of the Scrum Team.

- **Scrum Master:** Deeply understands the work being done by the team and can help the team optimize their transparency and delivery flow. As the facilitator-in-chief, he/she schedules the needed resources (both human and logistical) for activities like sprint planning, stand-up meetings, sprint reviews, and sprint retrospectives.

I.6.3 Scrum events

The Sprint is a container for all other events. Each event in Scrum is a formal opportunity to inspect and adapt Scrum artifacts. These events are specifically designed to enable the transparency required.

- **The Sprint:** Sprints are the heartbeat of Scrum, where ideas are turned into value. They are fixed length events of one month or less to create consistency. A new Sprint starts immediately after the conclusion of the previous Sprint.

- **Sprint Planning** The work to be performed during the current sprint is planned

during this meeting by the entire development team. Specific user stories are then added to the sprint from the product backlog. These stories always align with the goal and are also agreed upon by the scrum team to be feasible to implement.

- **Daily Scrum:** A 15-minute event for the Developers of the Scrum Team held every working day of the Sprint to inspect progress toward the Sprint Goal and adapt the Sprint Backlog as necessary.

- **Sprint Review:** The purpose of the Sprint Review is to inspect the outcome of the Sprint and determine future adaptations. The Scrum Team presents the results of their work to key stakeholders and progress toward the Product Goal is discussed.

- **Sprint Retrospective:** The purpose of the Sprint Retrospective is to plan ways to increase quality and effectiveness. The Scrum Team identifies the most helpful changes to improve its effectiveness and these changes are addressed as soon as possible.

I.6.4 Scrum artifacts

Scrum's artifacts represent work or value. They are designed to maximize transparency of key information. Thus, everyone inspecting them has the same basis for adaptation.

- **Product Backlog:** The primary list of work that needs to get done. It's a dynamic list of features, enhancements, and fixes that acts as the input for the sprint backlog. This list is constantly revisited, re-prioritized and maintained by the Product Owner in case items may no longer be relevant or problems may get solved in other ways.

- **Sprint Backlog:** The list of items, user stories, or bug fixes, selected by the development team for implementation in the current sprint cycle. A sprint backlog may be flexible and can evolve during a sprint without compromising the fundamental sprint goal.

- **Increment:** A concrete stepping stone toward the Product Goal. Each Increment is additive to all prior Increments and thoroughly verified, ensuring that all Increments work together. In order to provide value, the Increment must be usable.

Summary

To summarize our findings in this chapter, we introduced the host organization, then outlined the problem assessment and identified its related challenges. Next we analyzed competitors to get a grasp of the major pain points that we are going to tackle, and finally we presented the adopted agile approach in our team to foster flexibility and collaboration.

The next chapter will introduce the Sprint 0, where we kickstart the project, establish goals, and set the roadmap for upcoming sprints.

Chapter II

Sprint 0: Requirements gathering and Specification

Introduction

In this chapter, we place significant focus on gathering requirements and transforming them into well-defined and documented specifications. This includes creating use cases and user stories, which provide valuable insights into system interactions and user workflows. Furthermore, we establish a prioritized product backlog, ensuring efficient resource allocation and timely delivery of the most critical and valuable features.

II.1 Requirements gathering

By conducting a comprehensive analysis of the requirements, we aim to bridge the gap between the stakeholders' vision and the actual implementation of the product or system. This analysis helps us define clear and concise specifications that serve as the foundation for the design, development, and testing phases of the project.

II.1.1 Identifying end-users

Identifying end-users is a crucial step in requirements analysis as it helps determine the needs, expectations, and constraints of the target audience. In our application, we were able to identify three types of actors:

- **Administrator**: Is responsible for overseeing user accounts, configuring and maintaining the resources needed by agents or employees to ensure the smooth operation of the system inside the organization.

- **Agent**: Is an employee inside the organization in charge of executing actions on the notification center: creating and scheduling notification campaigns including the creation of notification content, client base segmentation.

- **Client**: Is a customer who is going to be targeted by notifications from the business or organization they belong to. A customer should be able to receive notifications and set their preferences for receiving notifications from that business.

II.1.2 Functional requirements

Functional requirements define the specific actions, tasks, and behaviors that the product must be able to perform in order to meet the needs of its end-users. These requirements form the foundation of the system's functionality.

The functional requirements we captured for each actor are outlined below.

Authentication and Profile settings

- **Sign in:** A registered user should be able to access the system by providing valid credentials.

- **Edit profile settings:** A logged in user should be able to edit his profile settings.

- **View statistics:** A logged in user should be able to view notification activity metrics on his dashboard.

Administrator requirements

- **Manage agents:** An administrator should be able to add new agents, update, delete, desactivate accounts and reset passwords for existing agents.

- **Manage channels:** An administrator should be able to create new notification channels, update, delete, configure service providers for existing channels.

- **Manage topics:** An administrator should be able to create new notification topics, update, delete and configure topic's priority for existing ones.

Agent requirements

- **Manage templates:** An agent should be able to create new notification templates, update and delete existing ones.

- **Manage triggers:** An agent should be able to create new notification triggers, con-

figure the target audience the scheduling, update, change status and delete existing triggers.

- **Manage audiences:** An agent should be able create new segments of users based on a criteria, update and delete existing ones.

- **View logs:** An agent should be able to view logs of sent notifications and their statuses.

Client requirements

- **Manage notification preference:** A client should be able to edit his notification preferences, channels and frequency of receiving notifications.

- **View notification history:** A client should be able to checkout a history of his received notifications (for in-app notifications).

II.2 Scrum implementation overview

This section provides an overview of the Scrum implementation in our project It includes a global use case diagram, showcasing the system's high-level functionalities, followed by the presentation of the product backlog and the planning of the sprints.

II.2.1 Global use case diagram

Using UML use case diagrams to model the requirements allows for a visual representation of the interactions between actors and the system, providing a clear and concise way to specify the functionalities and behaviors expected from the product. The figure II.1 illustrates the global use case diagram we modeled for our notification system.

II.2.2 Product backlog

The Product Backlog serves as a dynamic and living artifact that captures and organizes the ever-evolving list of features, functionalities, and enhancements desired for a software product.

- **Epic:** Is a large body of work that can be broken down into a number of smaller stories.

- **User story:** Is an informal, general explanation of a software feature written from the perspective of the end user or customer, in the form of:
  "As a [persona], I [want to], [so that]."

14
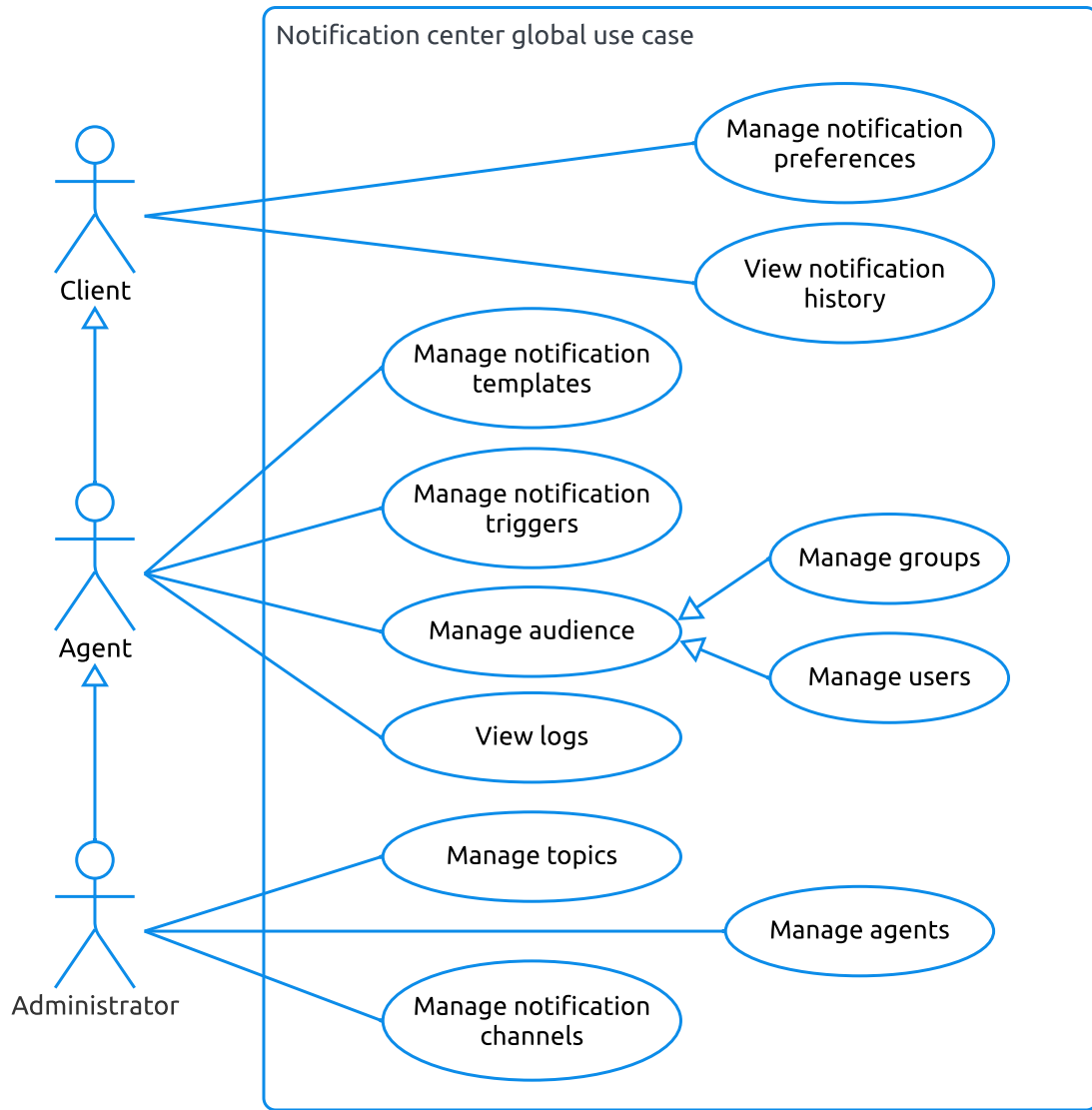
Figure II.1: Notification Center global use case diagram

Table II.1: Product backlog

Epic	User story
Authentication	**Authenticate**: As a registered user, I want to be able to log into my account securely using my email and password.
Agents management	**Create an agent**: As an administrator, I want to be able to create agents so that I can add them to the notification center.
	List agents: As an administrator, I want to be able to list agents so that I can view all registered agents.
	Edit an agent: As an administrator, I want to be able to edit agents so that I can modify their information.

Epic	User story
	Delete an agent: As an administrator, I want to be able to delete agents so that I can get rid of no longer needed agents.
Users management	**Create a notification user**: As an administrator/agent, I want to be able to create users so that I can send them notifications.
	List notification users: As an administrator/agent, I want to be able to list users so that I can view all created users.
	Edit a notification user: As an administrator/agent, I want to be able to edit users so that I can modify their information.
	Delete a notification user: As an administrator/agent, I want to be able to delete users so that I can get rid of no longer needed users.
Audience management	**Create an audience**: As an administrator/agent, I want to be able to create an audience so that I can target specific individuals based on a criteria.
	List audiences: As an administrator/agent, I want to be able to list audiences so that I can view all created user segments.
	Edit an audience: As an administrator/agent, I want to be able to edit an audience so that I can change selection criteria and segment configurations.
	Delete an audience: As an administrator/agent, I want to be able to delete a group of users so that I can get rid of no longer needed groups..
Notification channels management	**Create a channel**: As an administrator/agent, I want to be able to create notification channels so that I can send notifications through these channels.
	List channels: As an administrator/agent, I want to be able to list notification channels so that I can view all created channels.
	Edit a channel: As an administrator, I want to be able to edit notification channels so that I can update their configurations.
	Delete a channel: As an administrator/agent, I want to be able to delete notification channels so that I can get rid of no longer used channels.
Notification templates management	**Create a template**: As an agent, I want to be able to add notification templates so that I can send notifications based on that template.
	List templates: As an agent, I want to be able to list notification templates so that I can view all created templates.
	Edit a template: As an agent, I want to be able to edit notification templates so that I can keep them up to date.

Epic	User story
	Delete a template: As an agent, I want to be able to delete templates so that I can get rid of no longer used templates.
Notification preferences management	**Set notification preferences**: As a user, I want to be able to set my notification preferences, so that I can receive notifications from the channels I want.
Notification triggers management	**Create a trigger**: As an agent, I want to be able to create triggers for notifications so that I can schedule notifications to be sent automatically.
	List triggers: As an agent, I want to be able to list triggers for notifications so that I can view all created triggers.
	Edit a trigger: As an agent, I want to be able to edit notifications triggers so that I can modify or update its configurations.
	Delete a trigger: As an agent, I want to be able to delete notification triggers so that I can get rid of outdated and no longer used triggers.
Notification history	**Get notification history**: As a user, I want to be able to list notification history so that I can review my received notifications whenever I want.
	List sending logs: As an administrator/agent, I want to be able to list sent notification logs so that I can review all sent notifications.
Dashboard	**View metrics**: As an administrator/agent I want to be able to view metrics on my dashboard so that I can get an overview on important statistics related to notification activities.

II.2.3   Releases and sprints planning

A release refers to the deployment of a specific version or update of a software product. It involves meticulous planning and coordination to ensure successful delivery.

In the following table II.2, we present the detailed planning of the releases, outlining the sprints included in each release, and the epics included in each sprint.

II.3   Development infrastructure

In this section, we will discuss the development infrastructure established for the successful execution of the project. Our primary focus will be on defining the non-functional factors that are crucial for our product, then based on these factors, we'll proceed to discuss the software architecture, the database choice, and the curated set of tools and frameworks that were employed to support and streamline the development workflow.

Table II.2: Releases and sprints planning

Release	Sprints	Epics
Release 1 User, segmentation & channels	Sprint 1	Authentication
		Agents management
		Users management
	Sprint 2	Audiences management
		Notification channels management
Release 2 Sending notifications & dashboards	Sprint 3	Notification templates management
		Notification preferences management
	Sprint 4	Notification triggers management
		Notification history
		Dashboard

II.3.1   Non-Functional requirements

When designing a notification system, various technical requirements need to be considered to ensure the solution's effectiveness, reliability, and scalability. Here are the key factors that we should focus on while architecturing our system:

- **Security:** The system shall enforce secure communication protocols, such as HTTPS, to protect sensitive data during transmissionn, also data and preferences stored in the system shall be securely encrypted to prevent unauthorized access or data breaches.

- **Real-time:** The system shall deliver notifications in real-time or near real-time to ensure timely communication, messages and notifications should be delivered with minimal delay for high priority topics.

- **Scalability:** The system should be designed to handle a high volume of concurrent users and notifications without compromising performance and the system architecture should be scalable, allowing for horizontal scaling by adding more servers or utilizing cloud-based infrastructure as the user base grows.

- **Customizability:** The system should provide flexibility and customizability to meet the specific branding and user experience requirements of different organizations. Also the system should allow customization of user preferences to provide a personalized experience.

II.3.2   Software architecture

To implement our notification center, we will base our work on a client-server architecture. In this approach, the client, representing the user interface, interacts with the server, which houses the application's core logic and data. This separation of concerns allows for centralized management of data and business logic while catering to various clients.

One of the extensions to the mentioned architecture, and the one we will adopt is the **three-tier monolithic architecture**, where all the application modules and components are tightly integrated into a single codebase. This decision was based on several factors such as the project's scope, timeline, and the team's familiarity with monolithic architectures. Furthermore, for the envisioned requirements, a monolithic design offered simplicity and ease of development and deployment to quickly get the application started.

The figure II.2 illustrates the separation of the three tiers in the monolithic architecture :
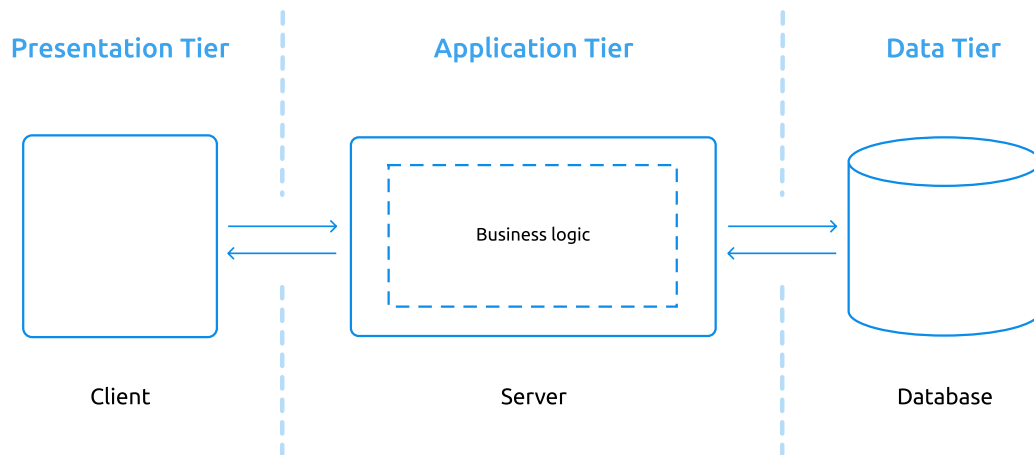

Figure II.2: Monolithic three-tier architecture

Presentation tier: The presentation tier is the user interface and communication layer of the application. Its main purpose is to display information to users and gather information from them. This top-level tier can run on a web browser, as desktop application, or a graphical user interface.

Application tier: In this tier, the data collected in the presentation tier is processed, sometimes against other information in the data tier, using business logic, a specific set of business rules. The application tier can also add, delete or modify data in the data tier.

Data tier: The data tier, sometimes called database tier, data access tier or back-end, is where the information processed by the application is stored and managed. This can be a relational database management system or a NoSQL Database server.

The user interface, business logic, and the database all reside on different machines and, thus, have different tiers. They are physically separated.

II.3.3 Technology stack and framework selection

In this part, we will delve into the technology stack and frameworks adopted for our project. This includes considerations for the back end, front end, and database components. By specifying these choices, we gain a clear understanding of the tools driving the development of our project.

II.3.3.1 JHipster Platform

In our quest of identifying efficient tools to expedite the development of an enterprise-level solution, we conducted research with a key focus on meeting the technical factors we envisioned. Our goal was to pinpoint a solution that not only accelerates project initiation but also empowers our team to concentrate on the core application logic while satisfying these crucial requirements.

We have reached a decision to adopt **JHipster** as the foundation for our project development. JHipster is a development platform to quickly generate, develop, and deploy modern web applications and microservice architectures [8].

The value proposition that JHipster provides is reflected in:

- The creation of fully configured applications for both the front and the back ends. In addition to the generation of code according to best practices and coding standards, ensuring a high level of code quality and consistency throughout the project.

- JHipster's integrated DevOps and CI/CD capabilities provide a seamless pathway to automate testing, deployment, and delivery processes.

JHipster provides flexibility and options, for the back end, it primarily supports Spring Boot, a popular Java-based framework which we discuss in detail in the following section II.3.3.2. On the front end side, JHipster offers a variety of choices including Angular, React and Vue, we elaborate on our choice for the front end technology in section II.3.3.3.

In conjunction with JHipster, we employed various complementary software tools to enhance the development process. (see Appendix A)

II.3.3.2 Server-side technology

This section delves into the technology considerations for the backend part of our solution and the employed internal architectural pattern.

II.3.3.2.1 Spring Boot framework

Spring Boot emerged as the backend technology for our project. Spring Boot is built upon the robust Spring framework, makes developing production-grade Spring based web application and microservices faster and easier through three core capabilities:

- **Auto Configuration:** Spring Boot comes with built-in autoconfiguration capabilities, it automatically configures both the underlying Spring Framework and third-party packages based on the provided settings.

- **Opinionated approach:** Spring Boot chooses which packages to install and which default values to use, rather than requiring the developer to make all those decisions on their own and set up everything manually.

- **Standalone applications:** Spring Boot helps developers create standalone that run on their own, without relying on an external web server, by embedding a web server such as Tomcat into the application during the initialization process.

II.3.3.2.2 Spring boot flow architecture

Spring Boot uses a hierarchical architecture in which each layer communicates with the layer immediately below or above it as shown in the figure II.3.
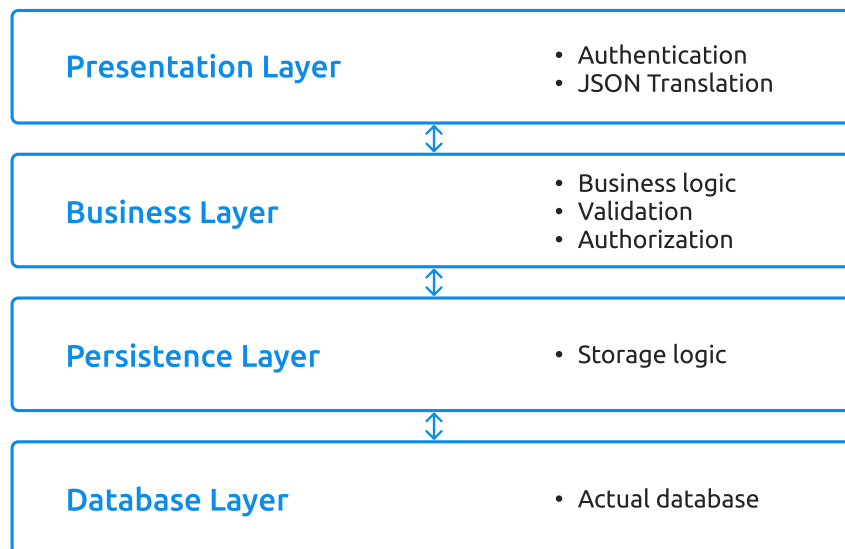

Figure II.3: Spring Boot application layers

1. **Presentation Layer**: The top layer of the spring boot architecture. It consists of views. It handles the HTTP requests and performs authentication. It is responsible for converting the JavaScript Object Notation (JSON) field's parameter to java objects and vice-versa. Once it performs the authentication of the request it passes it to the next layer.

21

2. **Business Layer**: Contains all the business logic. It consists of multiple service classes and is responsible for validation and authorization.

3. **Persistence Layer**: Implements all the database storage logic and it is responsible for converting business objects to the database records and vice-versa.

4. **Database Layer**: This layer provides integration with multiple databases. It is responsible for performing the CRUD operations.

Spring Boot integrates almost all of the features and modules of the Spring framework: like Spring MVC, Spring Core, Spring Data & JPA, etc. It follows nearly the same architectural pattern as Spring MVC, except for one thing: there is no need for Data Access Object (DAO) and DAO Implementation classes in Spring boot because its architecture has a data access layer and performs CRUD operations.
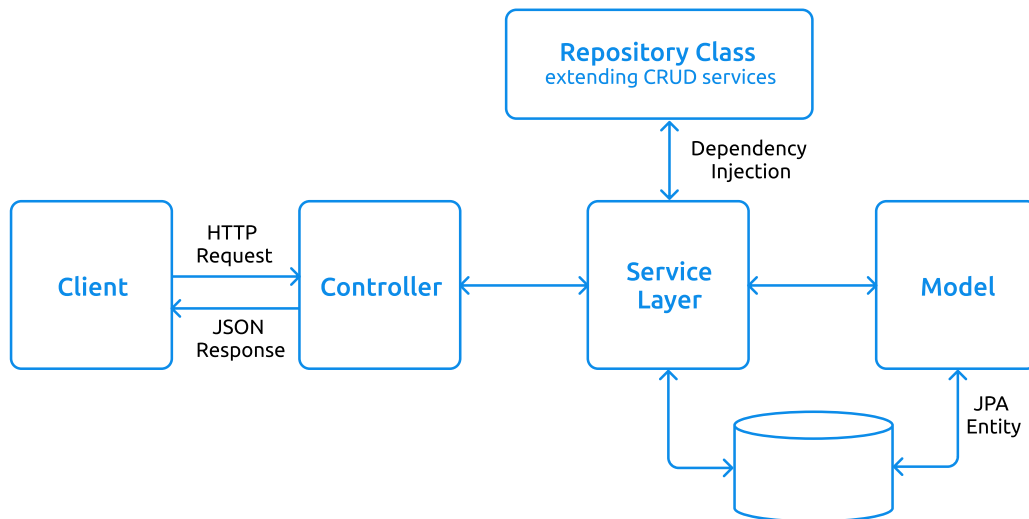

Figure II.4: Spring Boot flow architecture

Controllers are the major components in Spring's MVC framework. Their primary job is to handle HTTP requests and either hand off a request to a view to render HTML or like in our case write data directly to the body of the response as JSON [6].

In order to perform business logic on data, controllers call service classes from the service layer, which in turn inject repository classes to manipulate data in the database, and all entities are mapped to model classes through the Java Persistence API (JPA)

II.3.3.3 Client-side technology

This section illustrates the chosen technology for the client side of our solution and the adopted internal architectural pattern.

II.3.3.3.1 Angular framework

For the user facing part of our solution, we will select the **Angular** frontend framework. Angular, developed and maintained by Google, is a development platform, built on TypeScript and includes:

- A component-based framework for building scalable web applications.

- A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, etc.

- A suite of developer tools to help develop, build, test, and update code [7].

II.3.3.3.2 Angular architectural pattern

Angular follows a component-based architectural pattern that aligns closely with the Model-View-ViewModel (MVVM) pattern, a design paradigm that separates concerns in frontend development. While Angular's terminologies don't exactly match MVVM, the concepts map as follows:
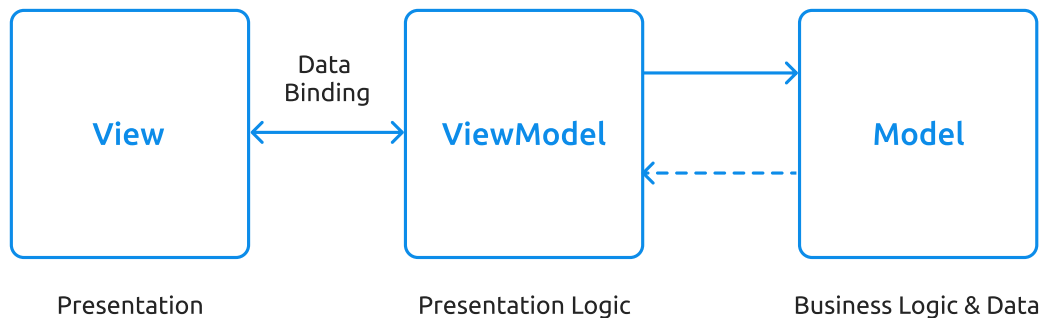

Figure II.5: MVVM architectural pattern

Model: Corresponds to business logic encapsulated within services and components. Services encapsulate data retrieval and manipulation, and act as the bridge between the application and external data sources.

View: Corresponds to the HTML templates in Angular. Templates define the structure and appearance of the user interface. Angular's binding mechanisms (property binding, event binding, and two-way binding) facilitate the synchronization between the model (data) and the view (UI).

ViewModel: Is represented by the component class. The component class contains the presentation logic, handling user interactions, processing data, and managing the application's state. It orchestrates the communication between the model and the view, ensuring that changes in one are reflected appropriately in the other.

II.3.3.4 Database

Selecting the appropriate database is a pivotal decision in our system design. After careful consideration, we've chosen to employ a relational database, specifically **PostgreSQL**. This choice is driven by the following criteria:

- **Structured Data:** We need to store notification channels and templates, user segments, and scheduling details in structured tables, these entities share relationships that interconnect them.

- **ACID Compliance:** We need to ensure data integrity and consistency, as it's critical for reliable notification delivery and management.

- **Query Flexibility:** We need to leverage SQL queries to effectively retrieve and manipulate Data, enabling comprehensive reporting and analysis.

PostgreSQL is an open source object-relational database system that has a strong reputation for its reliability, flexibility, and support of open technical standards. PostgreSQL comes with many features aimed to help developers build applications and help managing higher data loads.

Summary

In this chapter we laid the groundwork for the project by comprehensively gathering and analyzing the requirements. By identifying end-users, defining functional and non-functional requirements, and creating a clear specification through use case diagrams, elaborating the product backlog, planning sprints, specifying the application's architecture and the requisite development infrastructure, we have set the stage for the subsequent phases of development.

The insights gained from this chapter will prove invaluable in ensuring the successful implementation of the project and the satisfaction of the end-users. In the following chapter, we will start executing our project providing the practical realization of the initial release.

Chapter III

Release 1: Users, segmentation & channels

Introduction

In this chapter, we focus on the first release of our product, this release aims to create and enhance the user experience, implement userbase segmentation strategies, and manage notification channels. We will delve into the sprints backlog, design considerations, and the implementation process undertaken to bring these features to fruition.

III.1   Sprints backlog

During the Sprint Planning event, we estimated the effort required for each item in the first and second sprints backlog based on the number of working hours.

The priority of backlog items is reflected by their relative order in the table. Items positioned higher in the table indicate a higher priority.

Table III.1: Backlog of Sprint 1 & 2

Sprint	Epic	User story	Estimation (hours)
1	Authentication	Authenticate	16
		Create an agent	16
	Agents management	List agents	16
		Edit an agent	8
		Delete an agent	8

Sprint	Epic	User story	Estimation (hours)
	Users management	Create a notification user	16
		List notification users	16
		Edit a notification user	8
		Delete a notification user	8
2	Audience management	Create an audience	24
		List audiences	16
		Edit an audience	8
		Delete an audience	8
	Notification channels management	Create a channel	32
		List channels	16
		Edit a channel	8
		Delete a channel	8

III.2 Specification

In this section we are going to provide a detailed breakdown of some user stories. Each user story will be presented in a structured table format, ensuring ease of reference for the development team and that each user story is comprehensively documented. This table will include four key components:

- **Precondition:** Highlights any necessary conditions or prerequisites that must be in place before the user story can be initiated.

- **Business rules:** Business rules are critical as they define the constraints, policies, and conditions that govern how the software should behave in response to user interactions.

- **Technical specification:** This part specifies the technical aspects of the user story. It outlines the specific technical requirements, technologies, and architecture considerations that will be essential for the development team to realize the user story.

- **Acceptance criteria:** This section will provide a clear list of conditions that must be met for the user story to be considered complete and ready for release.

Table III.2: User stories specification

Epic	User story
Authentication	**Authenticate** As a registered user, I want to be able to log into my account securely using my email and password. **Precondition** • User is already registered and has a valid account. **Business rules** • Users must provide an email and password to authenticate. • Users should not be able to authenticate if credentials are incorrect. • Users must be redirected to the page they were trying to access before being prompted to log in. • Users must be able to reset their passwords if they forget them. **Technical specification** • The email field should accept a valid email address. • The login form should include a "Forgot Password" link that allows the user to reset their password. **Acceptance criteria** • User email and password are both required for log in. • Users should be able to log in using their registered email and password. • Invalid login attempts should result in an error message.

Epic	User story
Notification users management	**Create a notification user** As an administrator/agent, I want to be able to create users so that I can send them notifications. **Precondition** • Administrator/Agent authenticated and has a valid account. • Users do not exist already **Business rules** • Users should include a first and last name, email, a phone number and a country. • Administrators/Agents shouldn't be able to add a user with an already existing email or phone number. • Administrators/Agents should be able to import a list of users from a file. **Technical specification** • The email field should accept a valid email address. • The login form should include a "Forgot Password" link that allows the user to reset their password. **Acceptance criteria** • All required fields are provided: first and last name, email or phone number, and the country. • Created or imported users are saved to the database. • A message indicating the success or failure of the operation is displayed.

Epic	User story
Notification channels management	**Create a notification channel** As an administrator, I want to be able to create notification channels so that agents can use them to send notifications. **Precondition** • Administrator is authenticated and has a valid account. **Business rules** • Each notification channel must have a unique name. • The administrator should choose the type of notification channel, such as email, SMS, or push notifications. • The administrator should be able to configure the desired provider for the chosen channel type. **Technical specification** • A form should be provided for the administrator to enter the channel's name, type, the provider and its configuration. • The system should be able to handle multiple types of service providers, each with their own configurations. • The service provider credentials should be stored securely in the database. • The system should be able to integrate with external service providers using APIs. **Acceptance criteria** • The administrator can successfully create a new notification channel. • The administrator can conveniently configure the settings for each selected provider, such as API keys and credentials. • The system is able to deliver the notifications through the created channel according to its configurations. • A message indicating the success or failure of the operation is displayed.

Epic	User story
Audience management	**Create an audience** As an administrator/agent, I want to be able to create a segment of user subscriptions based on some criteria, so that I can target these subscriptions with notifications. **Precondition** • Administrator/Agent is authenticated and has a valid account. • Existing notification users and subscriptions. **Business rules** • Administrators/Agents should be able to create a segment based on one or more criteria including subscription channel, language, country, age, or other user or subscription attributes. • Administrators/Agents should be able to create as many audiences as they need. • Administrators/Agents should be able to store and retrieve audience data from the database. • Administrators/Agents should be able to get an estimation of the audience size while creation. **Technical specification** • A user subscription can be part of multiple audiences at the same time. • An audience shouldn't be empty, and should contain at least one notification subscription. • Administrators/Agents can create multiple audiences with different criteria. **Acceptance criteria** • The audience name should be unique. • Administrators/Agents can create an audience based on at least one criteria. • The group should only contain users satisfying the specified selection criteria. • A message indicating the success or failure of the operation is displayed.

III.3 Design

The design phase of software development is a critical step in the software development life cycle. It is the process of transforming the customer requirements into a detailed plan for how the software will be implemented. In this section we are going to focus on the design aspects related to the software and database design.

III.3.1 Class diagram

The class diagram provides a high-level overview of the system's structure, relationships, and interactions between different classes. It represents the static structure of the system, including classes, attributes, methods, and associations.

The following table provides a description of each class being part of the first release design.

Table III.3: Classes description

Class name	Description
User	This class represents the user entity in our solution
Authority	This class represents roles which are going to be assigned to users
NotificationUser	This class represents the target customers for receiving notifications
Channel	This class represents the notification channels that are going to send notifications through to our target users
ServiceProvider	This class models the credentials of the third party service to integrate with the channel
NotificationSubscription	This class models a user notification subscription to a specific notification channel
Audience	This class models a segment of notification user subscriptions selected based on a criteria
Criteria	This class models a selection criteria for an audience, it represents a filter, an operator, and a value

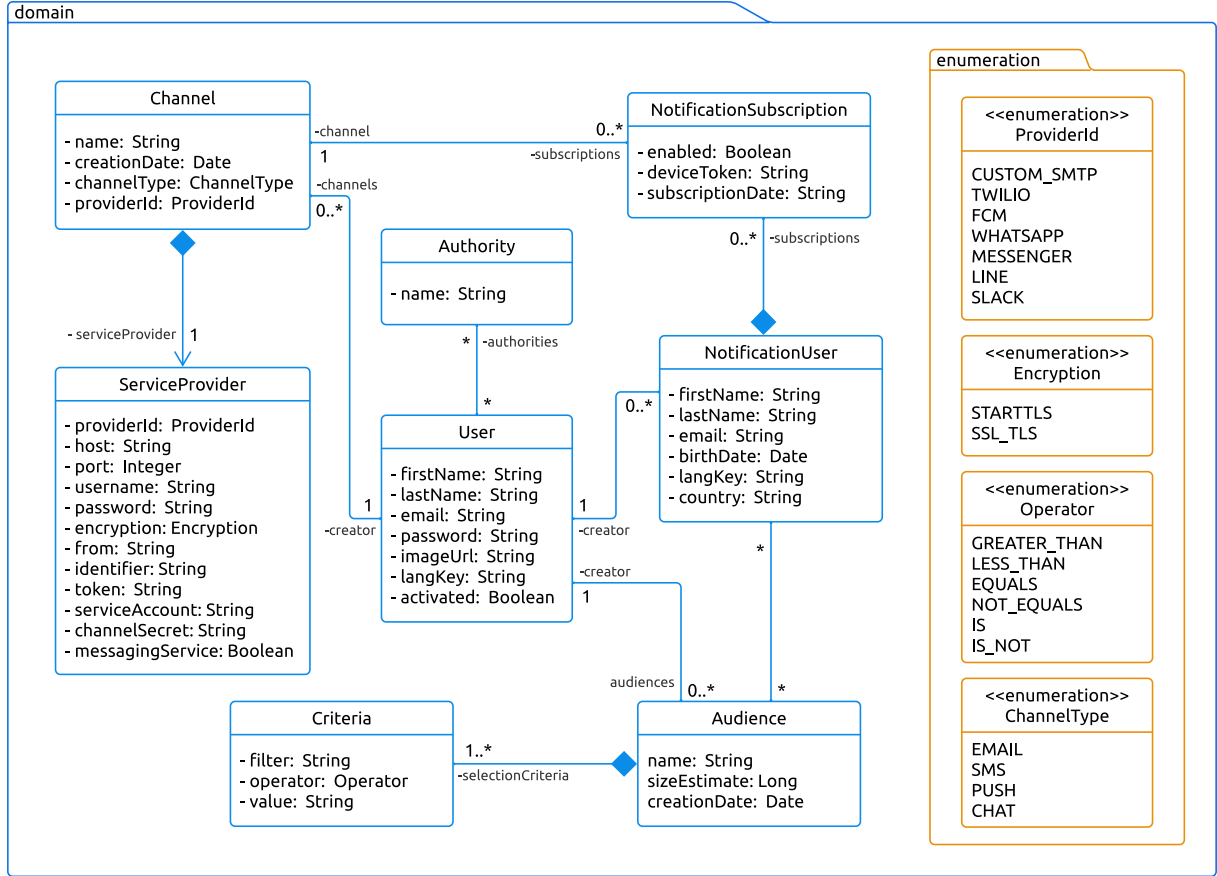The figure III.1 illustrates the class diagram for our first release.

Figure III.1: The class diagram of the first release

III.3.2 Database design

In this section, We will discuss the structure and organization of our database, defining tables and relationships between entities for efficient data management within our solution.

III.3.2.1 Data dictionary

First, we will define a data dictionary that describes the fields associated with each data element, indicating their types and constraints as shown in the table III.4.

Table III.4: Data dictionary of the first release

Entity	Field	Description	Type	Constraints
User	`id`	User's unique identifier	bigint	Primary key Not null
	`first_name`	User's first name	varchar(50)	
	`last_name`	User's last name	varchar(50)	
	`email`	User's email address	varchar(256)	Unique, Not null
	`password_hash`	User's encrypted password string	varchar(60)	Not null
	`image`	User's profile image url	varchar(256)	
	`lang_key`	User's prefered language key	varchar(10)	
	`avtivated`	User's account activation status	boolean	Not null
Authority	`name`	User authority (role)	varchar(50)	Primary key
NotificationUser	`id`	Notification user unique identifier	bigint	Primary key Not null
	`first_name`	Notification user's first name	varchar(50)	Not null
	`last_name`	Notification user's last name	varchar(50)	Not null
	`email`	Notification user's email address	varchar(256)	Not null
	`phone_number`	Notification user's phone number	varchar(50)	
	`lang_key`	Notification user's prefered language key	varchar(10)	Not null
	`country`	Notification user's country location	varchar(256)	Not null
	`birth_date`	Notification user's birth date	date	

Entity	Field	Description	Type	Constraints
Channel	`id`	Notification channel's unique identifier	bigint	Primary key Not null
	`name`	Notification channel's name	varchar(100)	Unique, Not null
	`channel_type`	Notification channel's type (EMAIL, SMS, PUSH, or CHAT)	varchar(10)	Not null
	`creation_date`	Notification channel's creation date	datetime	
	`service_provider_id`	Notification channel's specific service provider identifier	bigint	Unique, Not null
	`creator_id`	Identifier of the channel creator	bigint	Unique, Not null
ServiceProvider	`id`	Notification sending service provider's unique identifier	bigint	Primary key Not null
	`provider_id`	Service provider name identifier	varchar(50)	Not null
	`host`	Host address for the SMTP mail server	varchar(256)	
	`port`	Server port for the SMTP mail server	integer	
	`encryption`	Encryption type for the SMTP mail server	varchar(10)	
	`username`	Username for authenticating to the SMTP mail server	varchar(256)	
	`password`	Password for authenticating to the SMTP mail server	varchar(60)	
	`from`	The sender credential for the SMTP mail server or Twilio SMS provider	varchar(256)	
	`identifier`	Generic identifier for the service provider	varchar(256)	

Entity	Field	Description	Type	Constraints
	token	Access token for the service provider	varchar(60)	
	messaging_service	Whether or not to use a messaging service for Twilio provider	boolean	
	channel_secret	Secret code for the Line chat provider channel	varchar(60)	
	service_account	The whole configuration file content of the Firebase service account	text	
Notification Subscription	id	Notification user subscription unique identifier	bigint	Primary key Not null
	enabled	Whether the subscription is enabled or disabled	boolean	Not null
	device_token	The user token used to send notifications through, could be an email address, phone number, social media identifier	varchar(256)	Not null
	notification_user_id	Subscription's user identifier	bigint	Not null
	channel_id	Subscription's notification channel identifier	bigint	Not null
Audience	id	Audience's unique identifier	bigint	Primary key Not null
	name	Audience name	varchar(100)	Unique, Not null
	creation_date	Audience's creation date	datetime	Not null
	size_estimate	Audience's size estimation	bigint	Not null
	creator_id	Audience's creator identifier	bigint	Not null
Criteria	id	Criteria's unique identifier	bigint	Primary key Not null

Entity	Field	Description	Type	Constraints
	`filter`	The attribute used to filter out notification sub-scriptions	boolean	Not null
	`operator`	The operator used to create the comparison for the filter	varchar(20)	Not null
	`value`	The value to filter subscriptions by	varchar(50)	Not null
	`audience_id`	The unique identifier of the audience to which this criteria belongs	bigint	Not null

III.3.2.2 Logical data model

A logical data model is a model that is not specific to a database that describes things about which an organization wants to collect data, and the relationships among these things [3]. It contains representations of entities and attributes, relationships, unique identifiers, and constraints between relationships. The goal of creating a logical data model is to develop a highly technical map of underlying rules and data structures.

The figure III.2 illustrates the entity-relationship diagram for our first release.
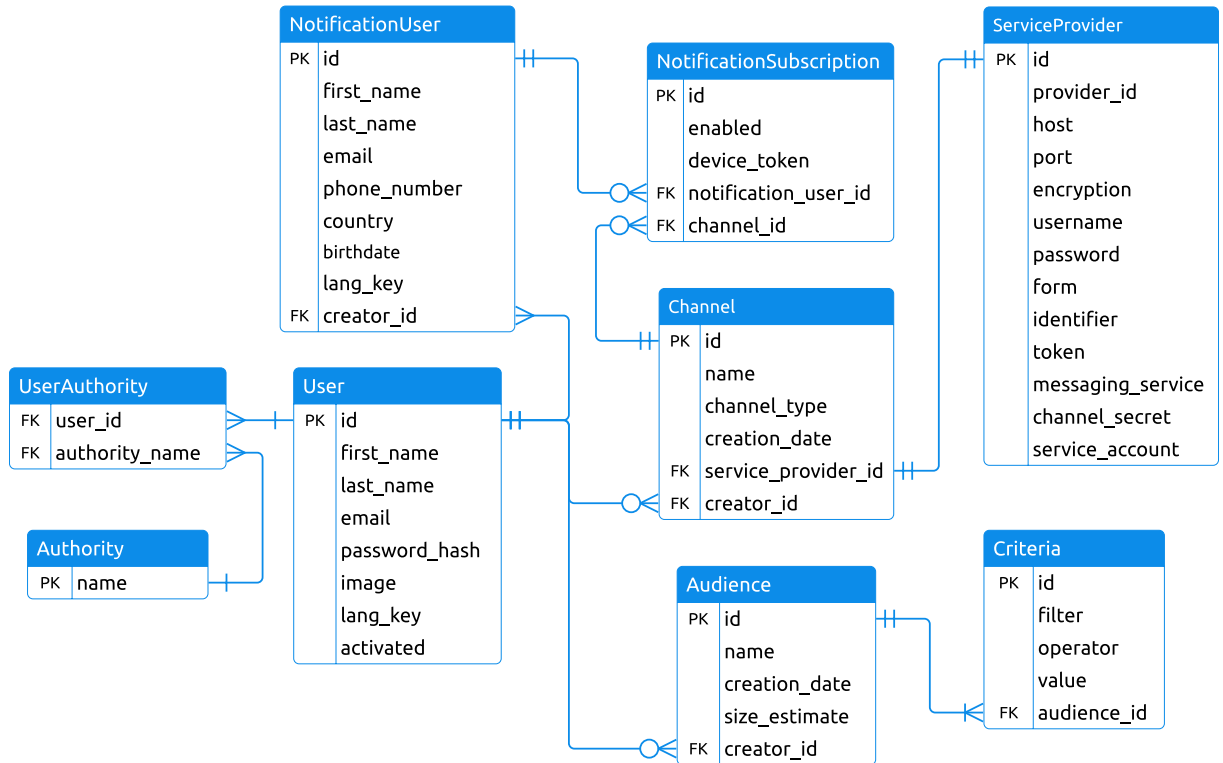
Figure III.2: The Entity-Relationship diagram of the first release

III.3.3 Sequence diagrams

In this section, we delve into the dynamic aspects of our software solution by employing UML sequence diagrams. These diagrams provide a visual representation of the interactions and communication patterns among various components and actors within our system. As we explore these sequences, we aim to shed light on the flow of information, the order of operations, and the coordination between different elements in our software architecture.

III.3.3.1 Sequence diagram for Creating a channel

First, we outline the process of creating notification channels, a critical functionality designed to integrate external notification delivery services within our solution.

As depicted in the user story, only administrators can create and manage channels. After filling the channel credentials the client validates the input data at the component level and hands the request to the server. In order to verify channel credentials, our server needs to communicate with external servers (Facebook, Twilio, Slack, Line). The reponse then tells if the channel is valid and could be saved in the database, or the credentials are invalid which cancels the request and asks the administrator to verify their input.

The following figure III.3 illustrates the sequence diagram for creating a channel.

III.3.3.2 Sequence diagram for creating an audience

For the user segmentation feature, both administrators and agents have the ability to create segments based on specific criteria. This essential functionality enables them to precisely target user subscriptions with relevant notifications.

In order to create an audience, users (administrators or agents) need to give their new audiences a name, then they can add one or more selection criteria while specifying for each criteria: a filter, a comparison operator, and a filtering value. The service layer of the client intercepts the changes while users are adding these criteria to make audience size estimation requests to the server, updating this information every short interval of time to keep users informed about the segments they are creating.

The following figure III.4 illustrates the sequence diagram for creating an audience.
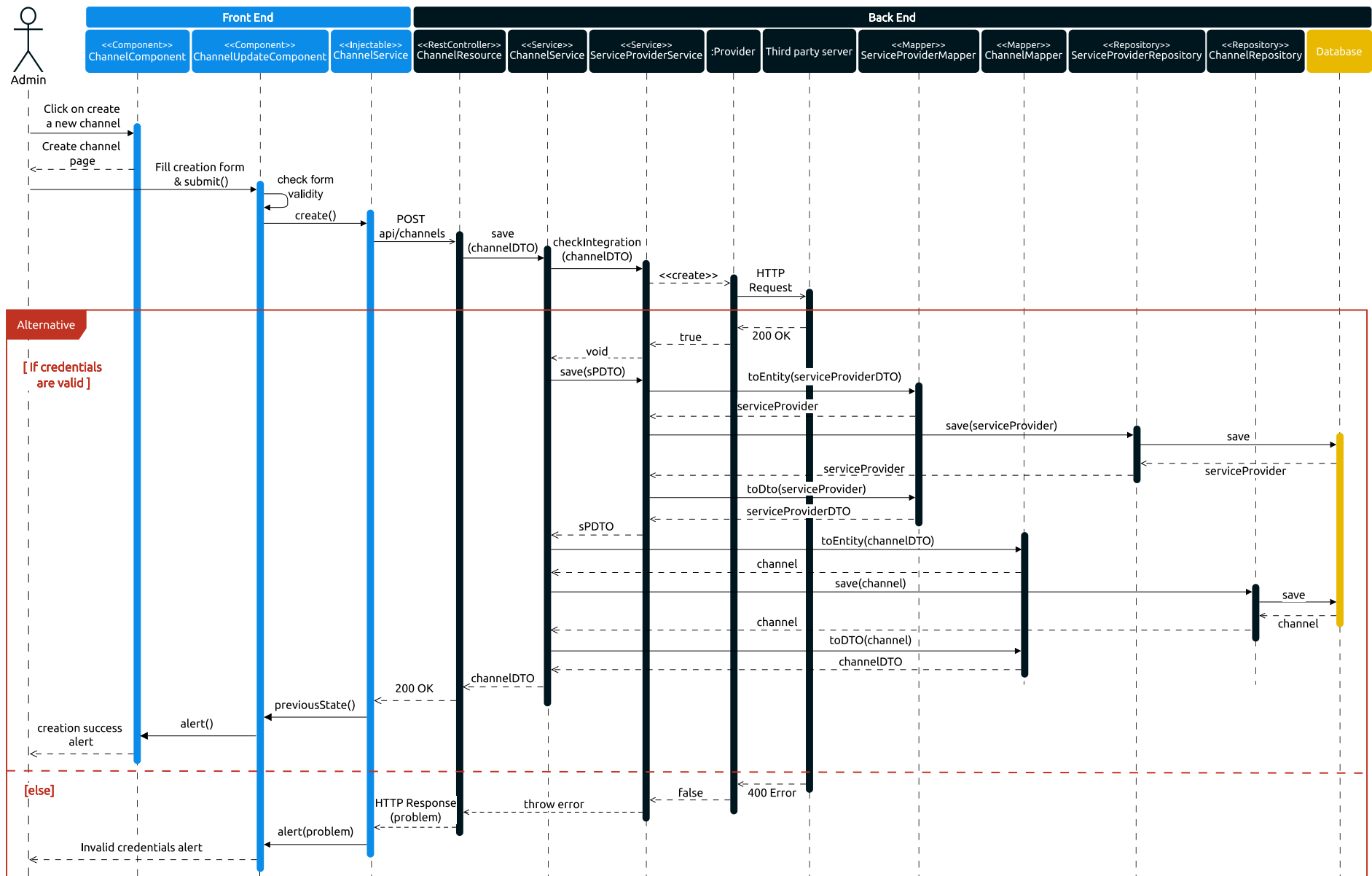
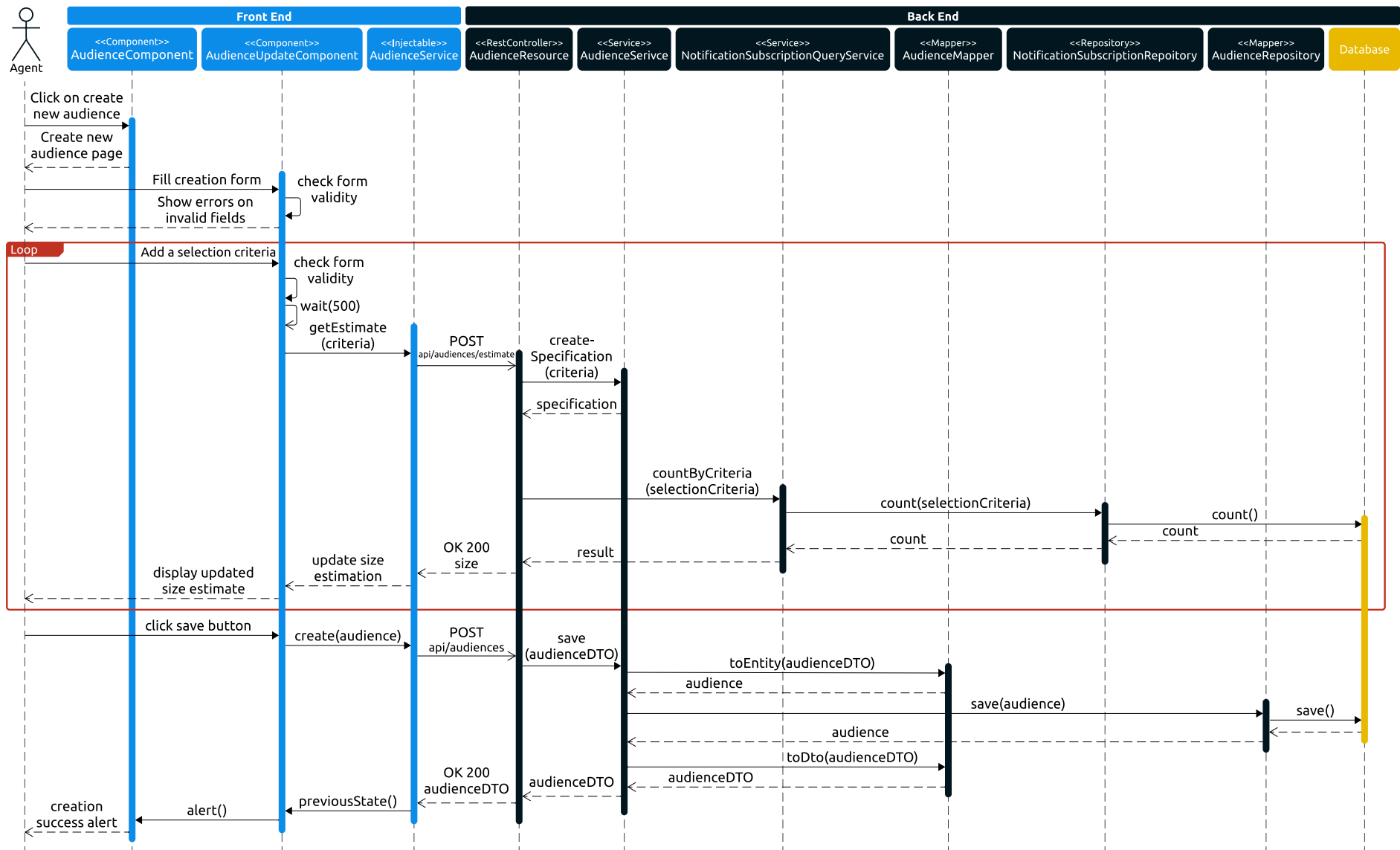Figure III.3: Sequence diagram for Creating a Channel

Figure III.4: Sequence diagram for creating an audience

III.4   Implementation

In the following section, we embark on a visual journey through a series of captured screenshots to provide an in-depth glimpse into the user interface and functionality of our application. the implementation of our application.

III.4.1   Authentication

The figure III.5 showcases the live implementation of the sign-in page, where users can provide their credentials (email address and password) in order to authenticate. The "Remember Me" checkbox allows users to stay signed in on future visits without the need to re-enter their credentials. The "Forget your password?" link provides a way for users to reset their password if they have forgotten it, and if there's an authentication error, a red banner with the error message is displayed above the input fields.
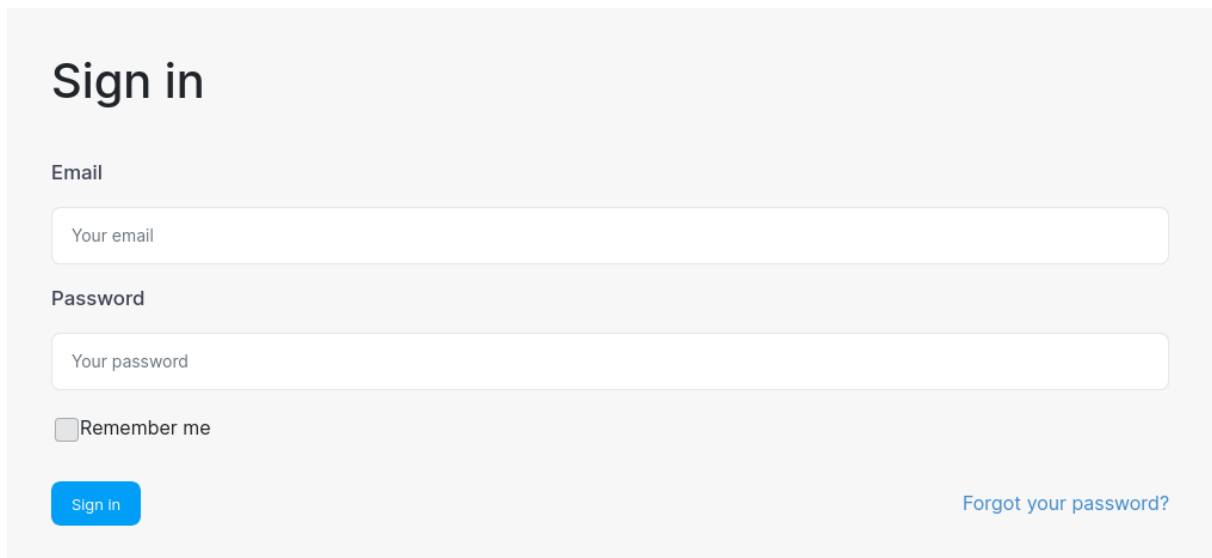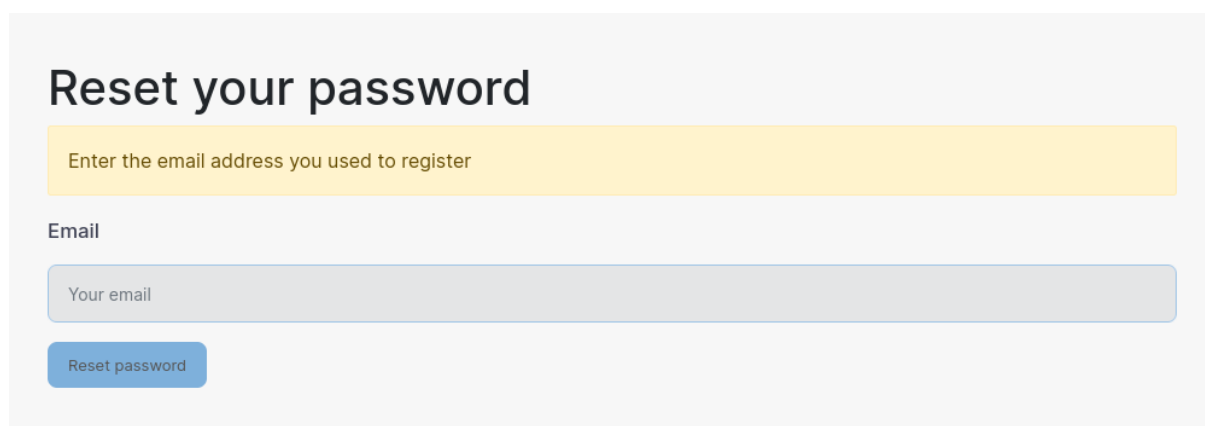
Figure III.5: Authentication page

The figure III.6 illustrates the implementation of the reset password page. A user is redirected to this interface when they click on the "Forgot your password?" link from the previous "Sign In" page, they should provide their email address and click on the reset button. A banner indicating that an email to recover their password is sent to their email address shows above the email input field.

III.4.2   Agents management

The figure III.7 illustrates the implementation of the agents management page. The table lists all the users (agents and administrators) for the logged in administrator, showing an overview of their information. For each user, an administrator can edit, reset a password, or delete an agent using the action buttons in the most right column, or else they can create a new agent or administrator using the "Create a new agent" button.
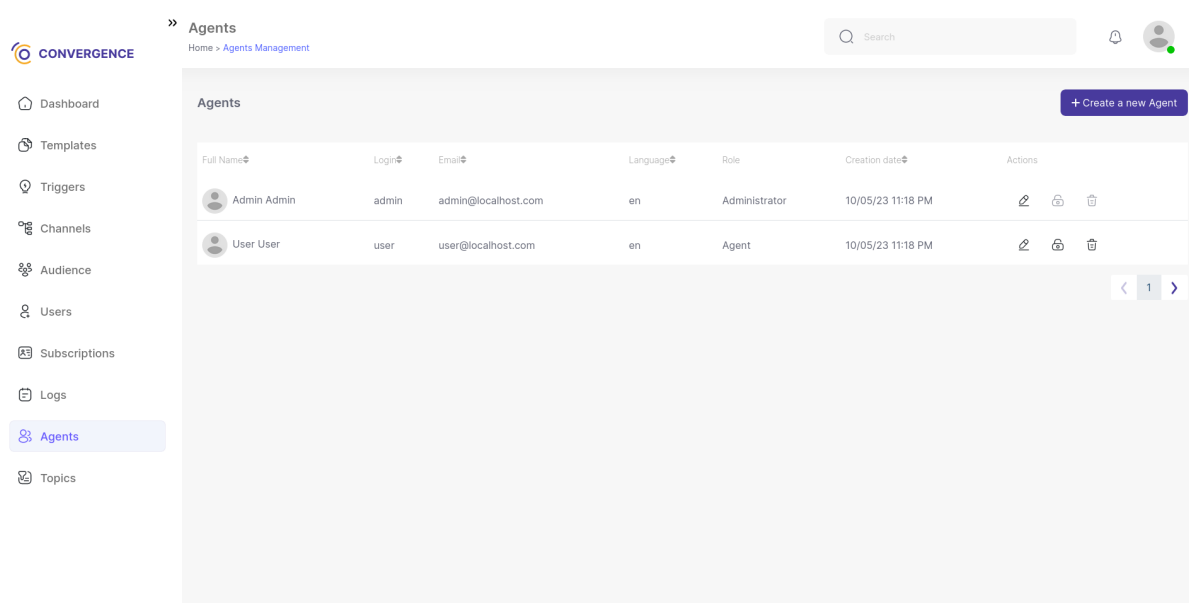
Figure III.6: Password reset page

Figure III.7: Agents management page

III.4.3 Channels management

The figure III.8 illustrates the final result of the implementation of channels management. The table lists information about created channels for every service provider that can integrate with our solution. The administrator can edit or delete these channels using the action buttons.

III.4.4 Audience management

The figure III.9 illustrates the final result of the implementation of the audience creation page. We created some selection criteria: the subsciption channel should be of email type, users age should be greater than 18 years, and subscribers are located in Tunisia. The size estimation for such audience is displayed in the section above.
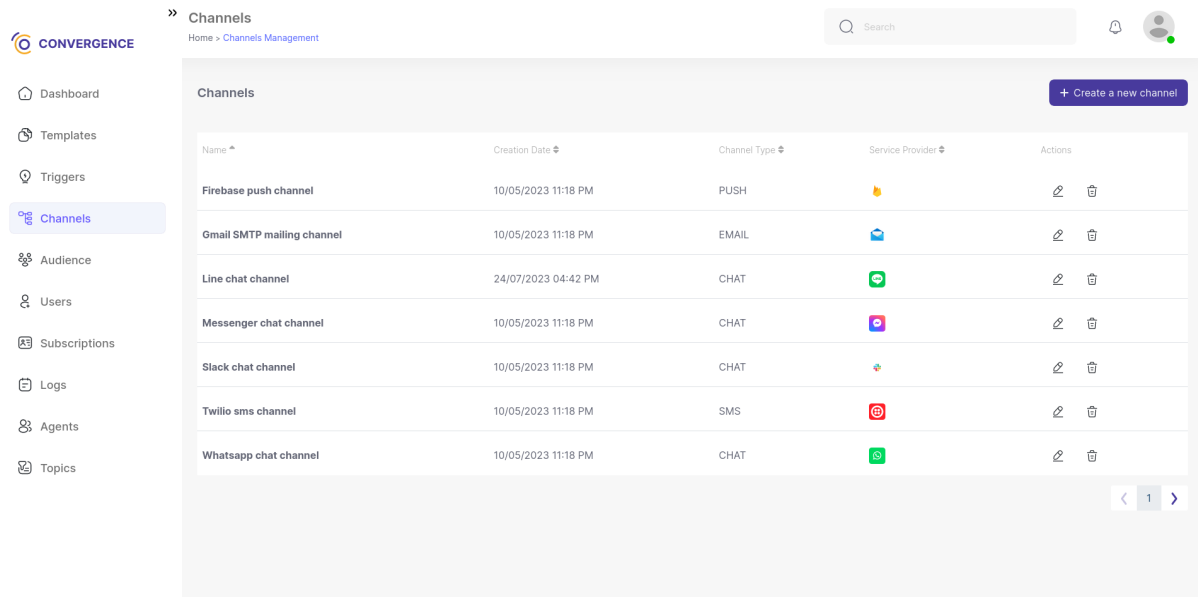
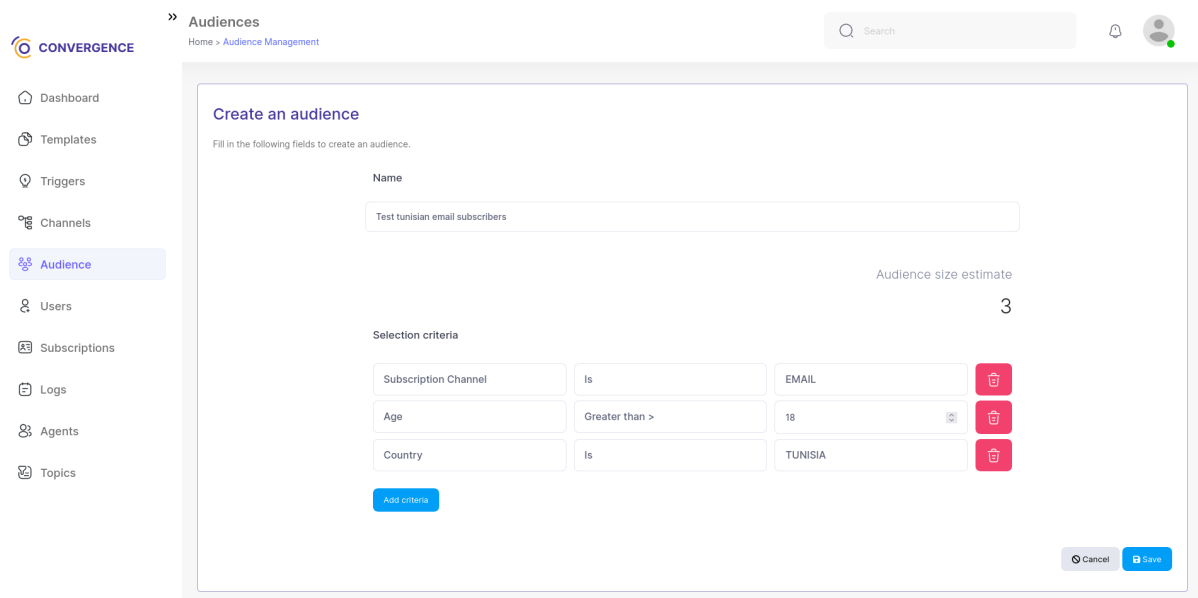Figure III.8: Channels management page

Figure III.9: Audience creation page

Summary

In this initial release, we defined user stories and related constraints in the specification phase. During the design phase we created a detailed plan that outlines how different components will interact to ensure that the software is well-organized, efficient, and capable of addressing the intended needs and goals. Finally, we showcased the result of implementing the essential components in our first release.

In the upcoming chapter, we will focus on the development of our second product release.

Chapter IV

Release 2: Sending notifications & dashboards

Introduction

Throughout this chapter, we will delve into the planning, design considerations, and implementation of our second product release. Building upon the foundation laid in our initial release, our focus now shifts towards fine-tuning the intricacies of notifications. This release encompasses a comprehensive overhaul of the notification system, spanning key areas such as notification templates, user preferences, delivery triggers, notification history, and the development of a powerful dashboard to provide insights into the delivery of notifications.

IV.1   Sprint backlog

During the second sprint planning event, we estimated the effort required for each item in the third and fourth sprints backlog based on the number of working hours. The priority of backlog items is reflected by their relative order in the table, same as the first release, items positioned higher in the table indicate a higher priority.

Table IV.1: Backlog of Sprint 3 & 4

Sprint	Epic	User story	Estimation (hours)
3	Notification templates management	Create a template	24
		List templates	16
		Edit a template	8
		Delete a template	8

Sprint	Epic	User story	Estimation (hours)
	Notification preferences management	Set notification preferences	16
4		Create a trigger	48
	Notification triggers management	List triggers	16
		Edit a trigger	8
		Delete a trigger	8
	Notification history	Get notification history	24
		List sending logs	16
	Dashboard	View metrics	32

IV.2 Specification

Similar to our approach in the first release, we will be detailing some selected user stories that we find most significant to drive the enhancements planned for this release.

Table IV.2: User stories specification for the second release

Epic	User story
Notification templates management	**Create a template** As an agent, I want to be able to add notification templates so that I can send notifications based on that template. **Precondition** • Agent is authenticated and has a valid account. • Existing notification topic. **Business rules** • Notification templates should include a name, a topic, a channel type, and the notification content. • Agents should be able to add placeholders for dynamic data in the template body. • Agents should be able to preview and test notification templates before saving. **Technical specification** • The system should persist notification templates in the database unpopulated with data. • Agents should be able to preview the notification template with test data without actually sending the notification to ensure that the message is rendered correctly. **Acceptance criteria** • Notification templates must have a name, a selected channel type, and non-empty body. • Agents can to add placeholders for dynamic data in the notification body. • Agents can preview the template with actual data and verify that the message is generated correctly. • A message indicating the success or failure of the saving operation is displayed.

Epic	User story
Notification triggers management	**Create a trigger** As an agent, I want to be able to create triggers for notifications so that I can schedule notifications to be sent automatically. **Precondition** • Agent authenticated and has a valid account. • Existing notification templates. • Existing audiences. **Business rules** • Agents should be able to choose a specific notification template. • Agents should be able to choose a specific audience to target. • Agents should be able to choose a delivery type for the trigger, either one time or recurring delivery. • Agents should be able to choose a frequency of sending notification in the recurring delivery type. • Agents should be able to specify a starting and ending date for the recurring delivery type. • Agents could create multiple triggers for the same notification template or audience. **Technical specification** • The system should provide a form where agents can select a notification template, an audience, set the trigger's schedule, and save the trigger. • The system should use a cron job or similar scheduling mechanism to send notifications according to the trigger's schedule. • The system should store the trigger and its associated scheduling data in the database. **Acceptance criteria** • Agents can sucessfully schedule the two types of notification delivery. • The notification system sends the notification according to the trigger's schedule. • The notification system stores the trigger and associated metadata in a database. • A message indicating the success or failure of the scheduling operation is displayed.

Epic	User story
Dashboard	**View metrics** As an agent I want to be able to view metrics on my dashboard so that I can get an overview on important statistics related to notification activities. **Precondition** • Agent is authenticated and has a valid account. **Business rules** • Administrators should be able to get a high-level metrics including: total number of sent notifications, subscriptions, delivery rates, open rates, and click-through rates. • The agent should be able to get an overview of notification activity specific to his account. • The administrator/agent should be able to get an overview on sent notifications by each of the channels to understand which ones are the most effective. • The administrator/agent should be able to get an overview on sent notifications metrics by each of the created segments of recipients. • The administrator/agent should be able to get an overview on sent notifications metrics by each of the created templates. **Technical specification** • Overviews should be displayed in the form of charts providing the metric for the current day and previous days. • Overviews by channel, audience or template should display the number of sent notifications per each category out of total. • Old metrics should be cached so that the system does not recalculate them with every request. . • Actual metrics should be calculated in real-time with a defined interval of update. **Acceptance criteria** • Overviews are displaying accurate metrics. • Update is happening after every defined interval of time. • Agents are getting metrics only for the notifications they sent. • Agents are getting metrics only for the templates and segments they created.

IV.3   Design

Building upon the insights gained from the initial design, this phase focuses on refining and expanding our software's architecture. In this section, we will explore the key considerations, decisions, and advancements made during this second release.

IV.3.1   Class diagrams

We have expanded our design to include five pivotal classes. These classes play a fundamental role in implementing the user stories related to notification management. They encompass critical aspects such as notification topics, templates, triggers, subscribers' notification preferences, and the core functionality of notifications—both for sending and maintaining a historical record. This section delves into the structure and relationships of these classes.

Table IV.3: Classes description

Class name	Description
Topic	This class models a notification topic which is used to categorize and prioritize notification delivery
Template	This class models a notification template which is used to create dynamic notifications content and configurations
NotificationTrigger	This class models a delivery trigger for sending notifications at a specific date and time
NotificationPreference	This class models a notification user preference for receiving notifications from the center
Notification	This class models each single notification that is going to be sent out to target users

In the following figure IV.1 we illustrate the extention of our class diagram, we highlighted newly added classes among others for a better readability.
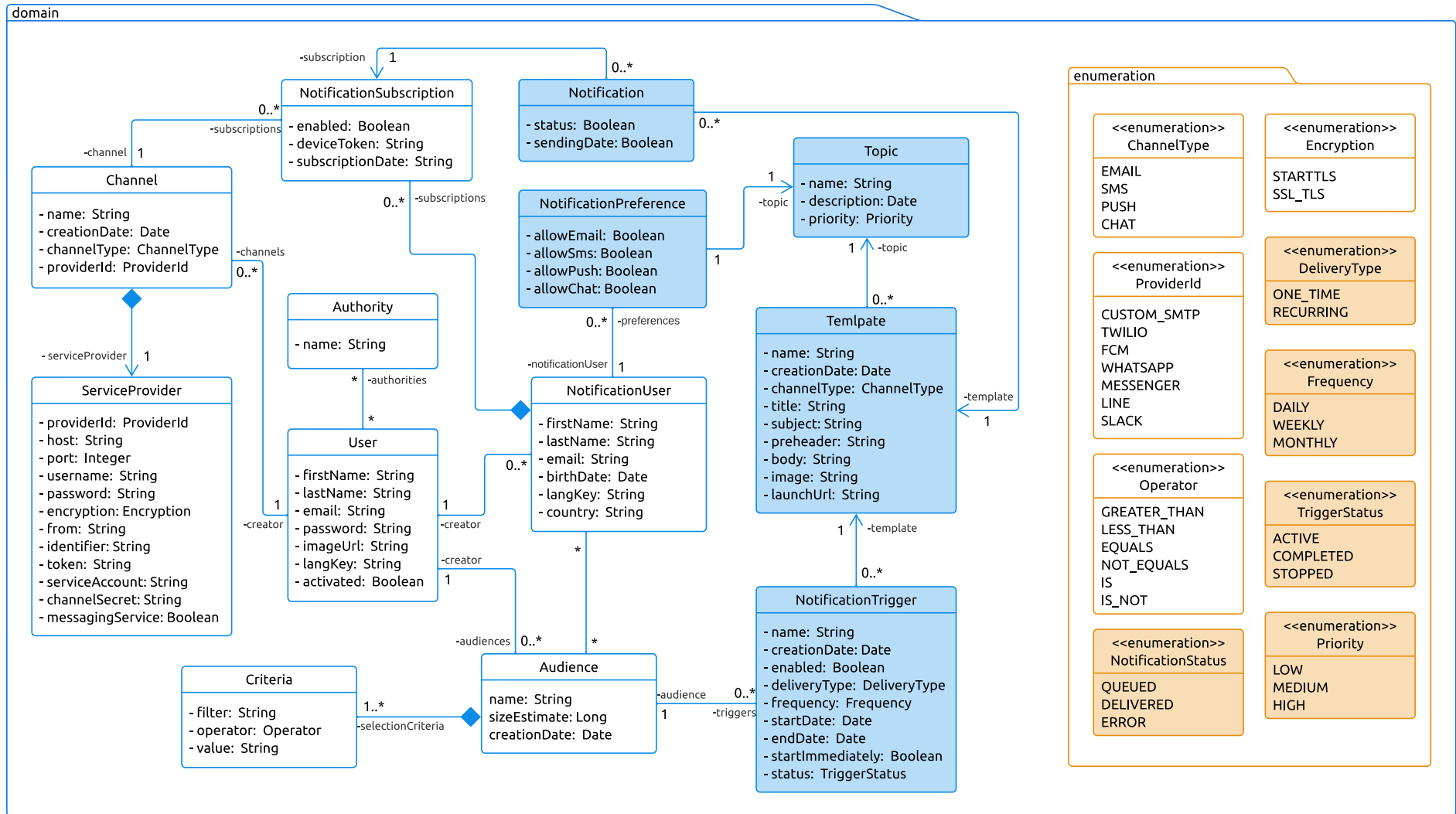
Figure IV.1: Class diagram of the second release

IV.3.2   Database design

In this section, We will elaborate on the structure and organization of our database from the first release, defining new tables and relationships for the implementation of this second release of our product.

IV.3.2.1   Data dictionary

In the following table IV.4, we provide a comprehensive overview of the essential data elements that will define our entities and relationships for the second release.

IV.3.2.2   Logical data model

Building upon the foundation laid in our initial logical data model, we enter a phase of refinement and expansion. In this section, we present the second iteration of our logical data model, which encapsulates the newly added entities during this second release. This updated data model will offer deeper insights into the structure and relationships of our data entities for the whole solution.

The figure IV.2 illustrates the updated entity-relationship diagram for this second release.

Table IV.4: Data dictionary of the second release

Entity	Field	Description	Type	Constraints
Topic	`id`	Topic's unique identifier	bigint	Primary key Not null
	`name`	Topic's name	varchar(100)	Unique, Not null
	`description`	Topic's description	varchar(256)	
	`priority`	Topic's notification sending priority	varchar(10)	Not null
	`creator_id`	Topic's creator unique identifier	bigint	Not null
Template	`id`	Notification template's unique identifier	bigint	Primary key Not null
	`name`	Notification template's unique name	varchar(100)	Not null
	`channel_type`	Notification template's channel type	varchar(10)	Not null
	`title`	Notification title for push notification	varchar(256)	Not null
	`preheader`	The preheader for email notifications	varchar(256)	
	`subject`	The subject for email notifications	varchar(256)	Not null
	`body`	The notification body content	text	Not null
	`image`	An image path that can be added for push and chat notifications	varchar(256)	
	`topic_id`	The unique identifier for the topic of the notification template.	bigint	Not null
	`launch_url`	The url to redirect to when clicking on push notifications	varchar(256)	

Entity	Field	Description	Type	Constraints
	creation_date	The notification template's creation date	date	Not null
	creator_id	The notification template's creator unique identifier	bigint	Not null
Notification Trigger	id	Notification trigger's unique identifier	bigint	Primary key Not null
	name	Notification trigger's unique name	varchar(100)	Unique, Not null
	status	Notification trigger status	varchar(20)	Not null
	audience_id	The unique identifier of the audience to target with notifications	bigint	Not null
	template_id	The unique identifier of the notification template	bigint	Not null
	delivery_type	The delivery type to be used for the trigger	varchar(20)	Not null
	frequency	The frequency of sending for the recurring delivery option	varchar(10)	
	start_time	The sending start date and time	date	
	end_time	The sending end date and time	date	
	creation_date	The trigger's creation date	date	Not null
	creator_id	The trigger's creator unique identifier	bigint	Not null
Notification Preference	id	Notification preference's unique identifier	bigint	Primary key Not null
	allow_email	Whether or not to receive emails for the specified topic	boolean	Not null

Entity	Field	Description	Type	Constraints
	`allow_push`	Whether or not to receive push notifications for the specified topic	boolean	Not null
	`allow_chat`	Whether or not to receive chat notifications for the specified topic	boolean	Not null
	`allow_sms`	Whether or not to receive SMS notifications for the specified topic	boolean	Not null
	`topic_id`	The notification topic's unique identifier	bigint	Not null
	`notification_user_id`	The notification user's unique identifier	bigint	Not null
Notification	`id`	Notification's unique identifier	bigint	Primary key Not null
	`status`	The status of the notification	varchar(20)	Not null
	`sending_date`	The notification sending date	date	Not null
	`subscription_id`	The targeted user subscription's unique identifier	bigint	Not null
	`template_id`	The unique identifier of the template used to send the notification	bigint	Not null
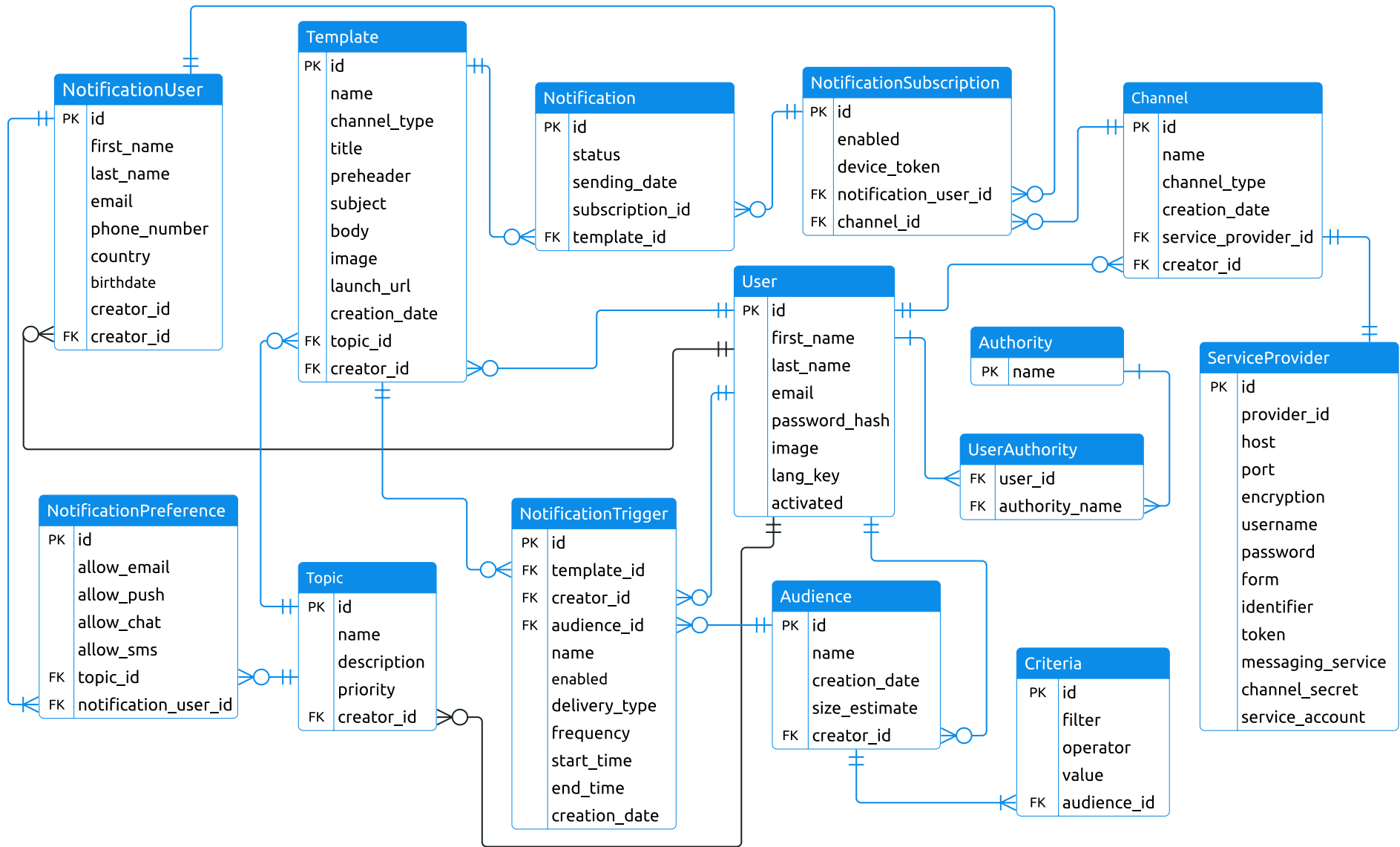
Figure IV.2: Entity-Relationship diagram of the second release

IV.3.3 Sequence diagrams

In the subsequent section, we revisit the dynamic facets of our software solution through further representation of the interactions and communication behaviors of components and actors within our system.

For this release we are going to focus on describing the process of creating notification templates and triggers, as we found those two functionalities to be the most important steps in order to be able to send notifications through our platform.

IV.3.3.1 Sequence diagram for creating a template

Agents can create templates by selecting the notification type and filling the required information. Agents can test templates before saving, in this case a REST request is made to the server which calls the service layer to configure and send the test notification.

When the agent proceeds to saving the template, a different REST request is sent to the server. The controller validates the request and calls the template service to map the data received to a template entity which is later saved by the repository in the database. The conroller finally sends a response back to the client indicating the success or failure of the operation.

The figure IV.3 illustrates the sequence diagram for creating a notification template.

IV.3.3.2 Sequence diagram for creating a trigger

In order to create a trigger, an administrator/agent selects a specific notification template, chooses a target audience, specifies a delivery type (one-time or recurring), sets the frequency, and defines a start and end date for recurring deliveries. Then the update component processes the user's input and formulates, through its service layer, a request to the backend for creating a notification trigger.

The controller receives the REST request and processes the incoming data. It validates the input and interacts with the service layer. The service layer ensures that the chosen notification template and audience exist, the dates and delivery type are valid, then it schedules an internal trigger based on those configurations. The reference for the internal trigger is then linked with the notification trigger we are creaating.

After applying this business logic, the service proceeds to save the notification trigger by calling the repository which in turn interacts with the database. Upon successful processing, the controller sends a response back to the cient indicating that the notification trigger has been created and scheduled successfully.

The figure IV.4 illustrates the sequence diagram for creating a notification trigger.
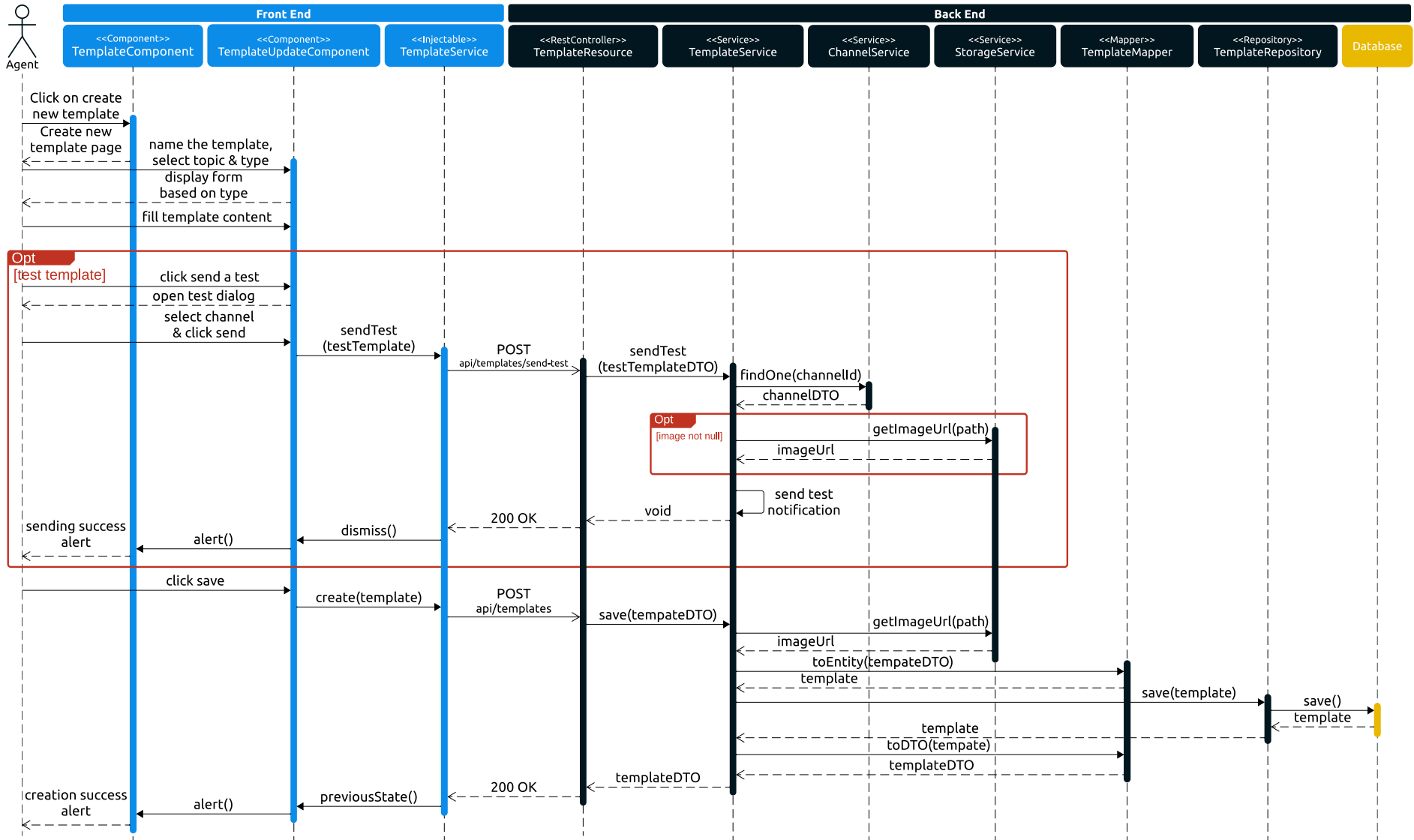
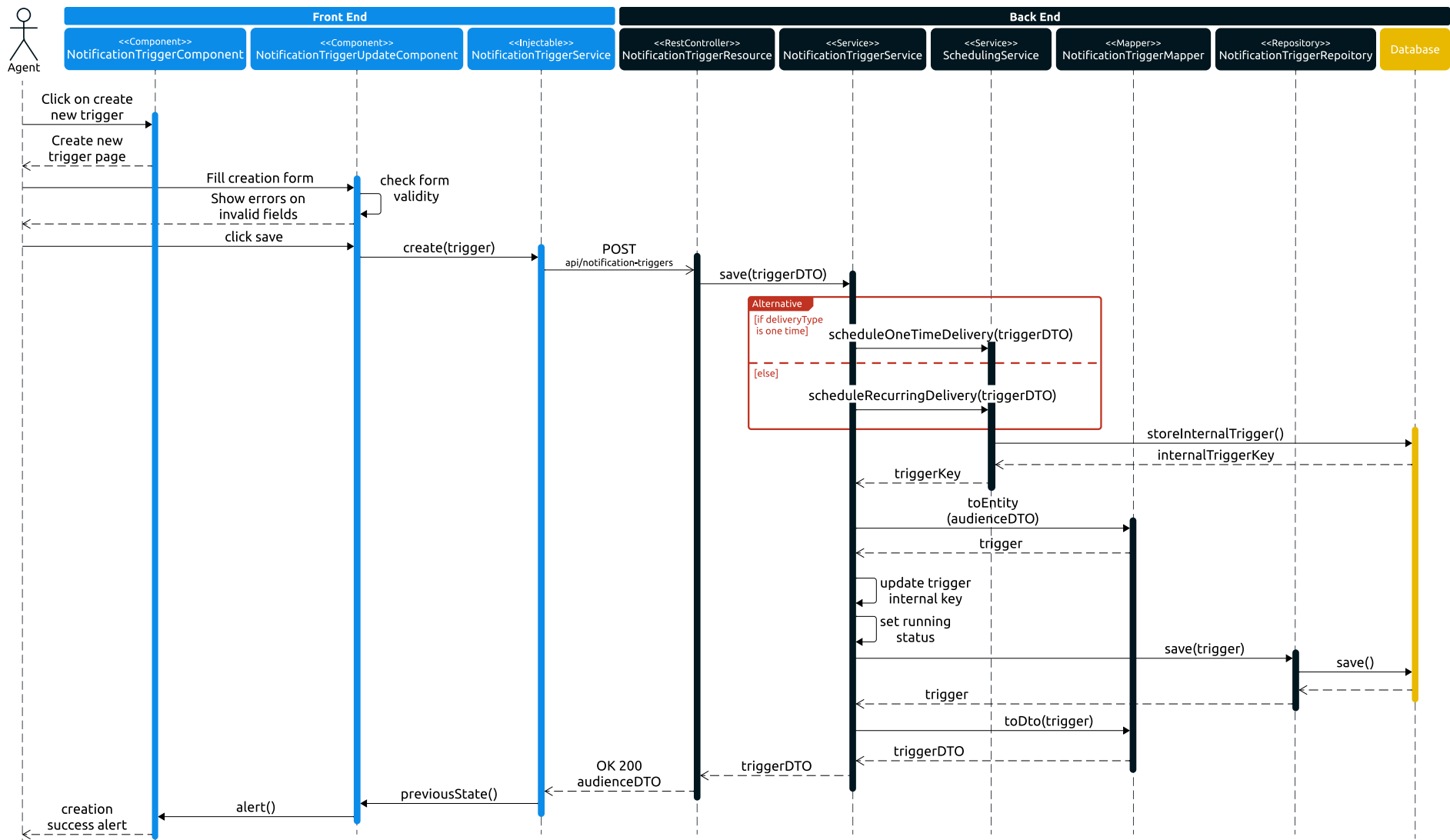Figure IV.3: Sequence diagram for creating a notification template

Figure IV.4: Sequence diagram for creating a trigger

IV.4   Implementation

Building upon the foundation laid in our first release, we showcase the evolution and refinement of our solution, highlighting the advancements made to meet our user requirements for this second release.

IV.4.1   Templates management

To be able to send notifications, first agents have to create a template by providing a name, selecting a specific topic for the template and choosing the template type (whether it's an email, sms, push or chat notification). Agents then should fill the template content, adding placeholders for dynamic data in the template body content, selecting and uploading an image if it is the case for the chosen type (as for the chat template), and if they want to test the template before saving they can do so by clicking the "Send test" button, else they can proceed to saving by clicking on "Save".

IV.4.2   Triggers management

Creating a notification trigger involves selecting a notification template and a target audience, choosing a delivery type (one-time or recurring), and setting the associated parameters, including frequency, start date, and end date.

The figure IV.6 shows the actual creation form where we selected an SMS test template, all SMS subscribers audience, the one time delivery type, and chose the option to send immediately. This option eliminates the need to specify a start date.

On saving success, the client navigates back to the notification triggers listing page, where we can see our created at the top of the list, showing its important details. Notice the status of the trigger is now set to "Running" while it is sending notifications. Upon acheiving, the status will change to "Complete" as shown for the older triggers in the figure IV.7.

Summary

In this chapter, we focused on the second release of our project, which centers around improving the notification sending processs. We started by specifying requirements related to this release, and delve into crucial design aspects to transform those requirements into actual implementations. Finally, we highlight our progress by showcasing visual representations of features covering templates and triggers management to enhance our notification system.

Figure IV.5: Template creation page

Figure IV.6: Trigger creation page

Figure IV.7: Triggers management page

Conclusion

In the fast-evolving digital landscape of today's world, the insurance, banking, and finance sectors face both unprecedented challenges and exciting opportunities. The need for seamless, personalized experiences has driven companies in these industries to embark on a journey of digital transformation. This transformation enhances customer engagement and elevates operational efficiency.

Our journey through the "Convergence" project at **Satoripop MEA** has been nothing short of transformative. Our mission was clear: to provide innovative solutions that facilitate the digital transformation process for organizations, enabling them to remain competitive, deliver exceptional customer experiences, and achieve sustainable growth in an ever-changing marketplace.

Within the "Convergence" ecosystem, we navigated various tools and sub-projects, each playing a vital role in our shared mission. Among these modules, we focused on the Notification Center, recognizing its pivotal role in enhancing communication by providing a centralized system for timely and relevant notifications. This project has allowed us to gain practical experience in designing and implementing complex systems, and most importantly developing proficiency in crucial technologies and frameworks.

As we bring this report to a close, we reflect on the significant milestones achieved during this internship project. We've took part in delivering a notification center that empowers organizations to communicate efficiently, engage customers effectively, and ensure the timely delivery of critical information.

However, our journey does not end here. The digital landscape will continue to evolve, and new challenges will emerge. Looking ahead, we must remain vigilant in adapting our notification center to meet the ever-changing needs of our clients and end-users. We envision future enhancements, such as campaing journeys, advanced analytics, AI-driven personalization, and even more secure data handling, to further solidify our position at the forefront of digital transformation.

In conclusion, the Convergence project has been a testament to our dedication to innovation and our commitment to solving real-world problems. We look forward to the next phase of this exciting journey.

Appendix A

Tools

This section presents the set of software tools we used to collaborate and support the development process.

	Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers. We used docker to create containers for the database and other components we needed for development and also to fully dockerize the application and all the services that it depends on.
	Bitbucket is a Git based source code repository hosting and collaboration tool owned by Atlassian. We used this service to host the source code for our project.
	Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration to create APIs faster. We used postman to test and maintain our notification center API.
	IntelliJ IDEA is an integrated development environment for developing computer software written in Java, Kotlin, Groovy, and other JVM-based languages.
	Figma is a collaborative web application for user interface and user experience design, with an emphasis on real-time collaboration, utilising a variety of vector graphics editor and prototyping tools. We used Figma to create wireframes, UI designs and illustrtations for our solution.
	draw.io is a cross-platform graph drawing software that can be used to create diagrams such as flowcharts, wireframes, UML diagrams, etc. We used draw.io to create the uml diagrams during the software desgin process.

	Trello is a web-based visual work and tasks management tool. We used trello to keep track of and to collaborate on the development tasks during the execution of the project.

Table A.1: Software tools

Bibliography

[1] *Amazon Simple Notification Service.* URL: https://aws.amazon.com/sns (visited on 04/20/2023).

[2] *Leader In Realtime Technologies.* URL: https://pusher.com (visited on 04/20/2023).

[3] *Logical data models.* Mar. 2021. URL: https://www.ibm.com/docs/en/ida/9.1.2?topic=modeling-logical-data-models (visited on 09/18/2023).

[4] *Push Notification Software to Improve Customer Engagement.* URL: https://onesignal.com (visited on 04/20/2023).

[5] Ken Schwaber & Jeff Sutherland. *The Scrum Guide.* Nov. 2020. URL: https://scrumguides.org/scrum-guide.html (visited on 07/01/2023).

[6] Craig Walls. *Spring in action, 6th edition.* Manning Publications Co, 2022. ISBN: 9781617297571.

[7] *What is Angular?* Feb. 2022. URL: https://angular.io/guide/what-is-angular (visited on 08/20/2023).

[8] *What is JHipster?* URL: https://www.jhipster.tech (visited on 08/22/2023).