



Training Notebook

Presented by:
Mateusz Rysiewski

What the project is about:

The project is a web application that serves as a training notebook, providing support for training by allowing users to store and access their progress data anywhere through the internet, thereby encouraging them to continue their workout journey.

Target audience:

The target audience for this application is individuals who are engaged in training activities and require a portable solution for storing their secured training notes accessible through their created accounts.

Ideas for future development:

Initially, the product aims to provide the capability of storing data related to sports and workouts, accessible on multiple devices through a REST API. However, there is potential for expanding the API with various features, including:

- Calculators such as BMI, calorie burn estimators.*
- User comparisons against other users.*
- Incorporating a chat feature where users can schedule and participate in group sports or workouts.*




Furthermore, if the REST API proves successful, it is recommended to develop a mobile application to broaden the customer base.

Monetization of the REST API:

- It is suggested to introduce non-intrusive advertisements that do not disrupt user experience while also offering the option to purchase an ad-free period or a complete ad-free version.*
- Visual aspects such as colors, fonts, and effects can also be incorporated.*
- Ultimately, subscription-based models can be implemented, but it will require a significant user base to achieve a fluid service and gain market traction.*

1. Infrastructure

Availability and general information




			 Google Cloud
Market share in Q4 2022:	32%	23%	10%
Country of origin:	United States	United States	United States
Established in:	2006	2010	2008
Regions and zones	99 availability zones and within 31 geographical regions	60+ announced regions	200+ countries or territories with 176 network edge locations, 106 zones, and 35 regions
Compliance with EU regulations:	Yes	Yes	Yes
Monthly website visitors in Q4 2022:	66.15M	8.333M	34.22M
General characteristics:	Oldest and most mature provider	Has experienced fluctuations in service performance	Relatively young platform but leverages Google's experience

		but has improved in recent years	
Documentation	Best in class	High quality	High quality

Short summary

Comparing the cloud providers mentioned above, AWS should be the most trusted vendor. It holds a significant market share and has gained the trust of numerous companies. AWS is a pioneer in the field with the longest tenure. There is no information available regarding service disruptions or issues with their service delivery. Additionally, AWS offers worldwide server availability.

Functionality and possibilities

			 Google Cloud
Additional feature count:	200+	100+	60+
biggest strength	<ul style="list-style-type: none"> Scalability the largest number of possibilities, extensive reach 	<ul style="list-style-type: none"> Seamless integration with the Windows ecosystem (e.g., .NET code can be used in Azure) 	<ul style="list-style-type: none"> pricing The most user-friendly out of big three Less vendor lock-in
performance speed max.	Up to 10Gbps	Up to 100Gbps	Up to 100Gbps
Compute offering	EC2 is considered the best with high flexibility and configurability.	Comparable to EC2 in terms of functionality but offers slightly fewer related services and flexibility.	Offers high flexibility and configurability similar to EC2. It also has a wide range of computer-related services, including the popular Kubernetes Engine. It falls between EC2 and


			Virtual Machines in terms of flexibility.
Difficulty in interface handling:	High	Average	Average

Short summary

There is no clear winner in terms of functionality as each cloud provider excels in different areas. Here are the recommendations:

- AWS is the best choice when scalability and a wide range of features are needed, although it may have a steeper learning curve in terms of usability.
- Azure is attractive for users with a Windows ecosystem and those seeking seamless integration. It is generally easier to use compared to AWS.
- GCP is a good option for customers focusing on containers and seeking support in that area. They are strong competitors in the market, mainly due to aggressively competitive pricing and simplifying onboarding for new users, although they may have fewer functional capabilities. Additionally, Google Cloud has less vendor lock-in with services like BigQuery Omni, which can analyze data across different clouds, including AWS and Azure.

PRICING




			 Google Cloud
They have expense calculators	Yes	Yes	Yes
Ability to customize the offer for a specific customer	Yes	Yes	Yes

Average costs	AWS and Azure are similar	AWS and Azure are similar	Smaller than Azure and AWS
Complexity of the calculator	High	Medium	Medium
Billing method	Per minute	Per minute	Per second
Additional information			<ul style="list-style-type: none"> • At the start of usage, you receive \$300 credit for GCP services for a period of 12 months • The biggest discounts on the market

Short summary

- AWS and Azure have very similar pricing in the market, while GCP stands out in this aspect by offering the biggest competition. It provides many payment-related add-ons to attract new customers, although it has slightly fewer functionalities compared to its more expensive competitors.

SECURITY, STORAGE, MIGRATION

			 Google Cloud
Default security settings:	Securely configured	Less restrictive default settings	Typically, secure default settings
Data backup capability	Yes	Yes	Yes
encryption for all files	256-bit AES	256-bit AES	256-bit AES

Data transmission encryption methods	SSL/TLS	SSL/TLS	SSL/TLS
Protection	Shield	DDoS	Google Cloud Armor
VPN GATEWAY	<ul style="list-style-type: none"> • Point to site • Site to site • Limit of 10 site to site connections per VPN gateway 	<ul style="list-style-type: none"> • Point to site • Site to site • Limit of 30 site to site connections per VPN gateway 	<ul style="list-style-type: none"> • Only site to site
Authentication and Authorization	IAM	Azure AD	Oauth 2.0
Containerization support	Yes	Yes	Yes
NoSQL database security support	Yes	Yes	Yes
SQL database security support	Yes	Yes	Yes
Database migration	Offers services and tool	Only service	Only service
Migration		supports a hybrid model, enabling enterprises to switch the solution gradually	

Summary:

- AWS: Amazon Web Services is widely used across various industries and offers a wide range of cloud services. It provides strong security and securely configured default settings. It offers data backup support, various data encryption methods, and robust network security such as AWS Shield. AWS also has rich database migration tools.

- Azure: Microsoft Azure is popular among companies already using Microsoft solutions such as Windows, Office, or SQL Server. The platform offers hybrid migration capabilities, allowing gradual transition to the cloud. It has strong security features, although default settings are less restrictive. Azure supports data backup, provides data encryption methods, and offers VPN solutions.
- GCP: Google Cloud Platform focuses on developers and offers a rich set of serverless services. It specializes in areas such as data analytics, big data processing, and machine learning. GCP provides a high level of security, with typically secure default settings. It offers data backup support, data encryption methods, and database security solutions.

In general, each provider offers high-level security services. However, to choose a specific provider, it is important to examine the final requirements of our security and whether the service we use supports those requirements. It can be assumed that AWS will provide more capabilities, even for less common security needs, due to its extensive range of features, which Azure is also striving to match.

OVERALL SUMMARY

WHEN TO CHOOSE AWS:

- Targeting larger companies
- Not concerned about costs
- Large-scale project with extensive functionality
- Need for a highly reliable provider
- Need for the widest reach

WHEN TO CHOOSE AZURE:

- Strong reliance on Microsoft technologies in the project
- Need to gradually transition from another service provider
- Not concerned about costs

- Large-scale project with extensive functionality

WHEN TO CHOOSE GCP:

- Consideration of costs is important
- Seeking a provider with a low entry barrier
- Concern about vendor lock-in and desire for flexibility
- Utilizing specific elements offered by Google

In my opinion, Google Cloud Platform (GCP) would be the best choice for the project at hand. GCP offers a wide range with enough of functionalities and supports the tools we intend to use, such as a SQL database and containerization. It is known for its ease of use, which will accelerate the implementation process. Additionally, GCP provides sufficient capabilities to meet the project's initial requirements.

2. Architecture

Technical aspects:

The "Training Notebook" project consists of an interactive frontend written in React (JS/CSS/HTML), which communicates with a Spring Cloud API Gateway in the cloud. The API Gateway acts as a mediator, enabling communication between the frontend and microservices.

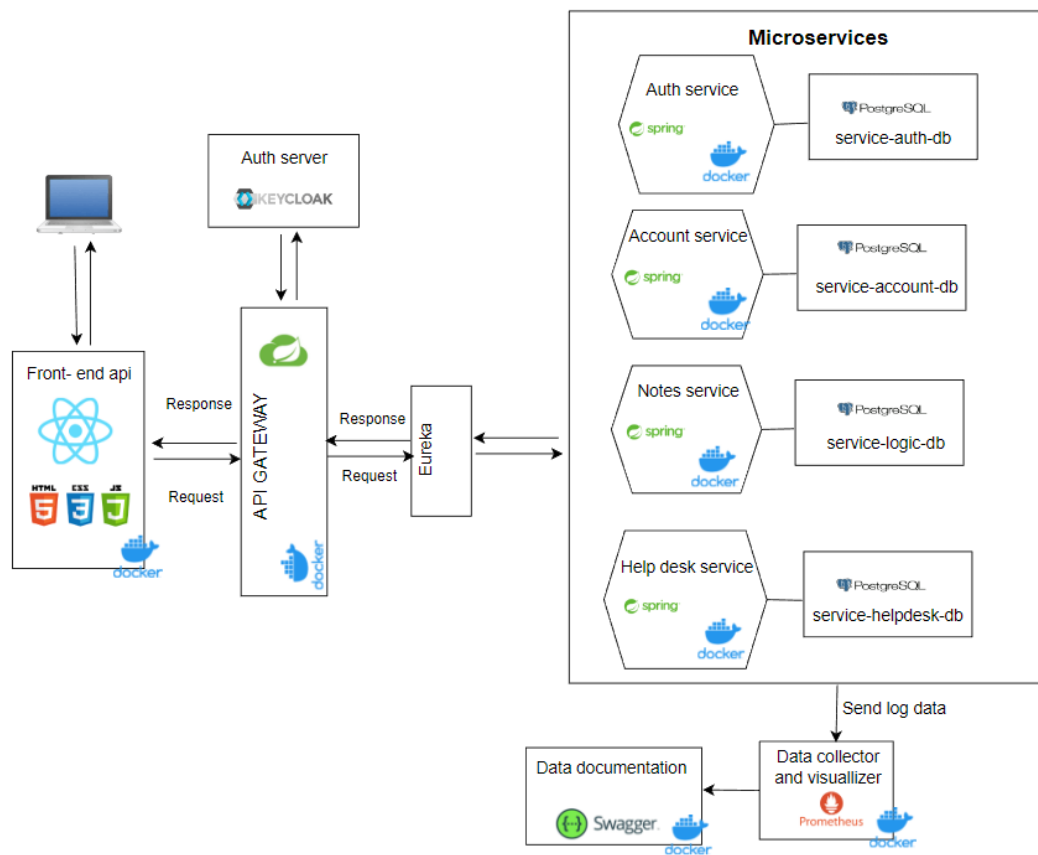
The project also utilizes the Keycloak authentication server, which handles user authentication and access control to resources.

Communication between the microservices is facilitated by Eureka, which serves as a service registry, allowing for dynamic service discovery and communication between services.

There are three independent microservices in the project, each with its own PostgreSQL database. The microservices are responsible for different functionalities within the project.

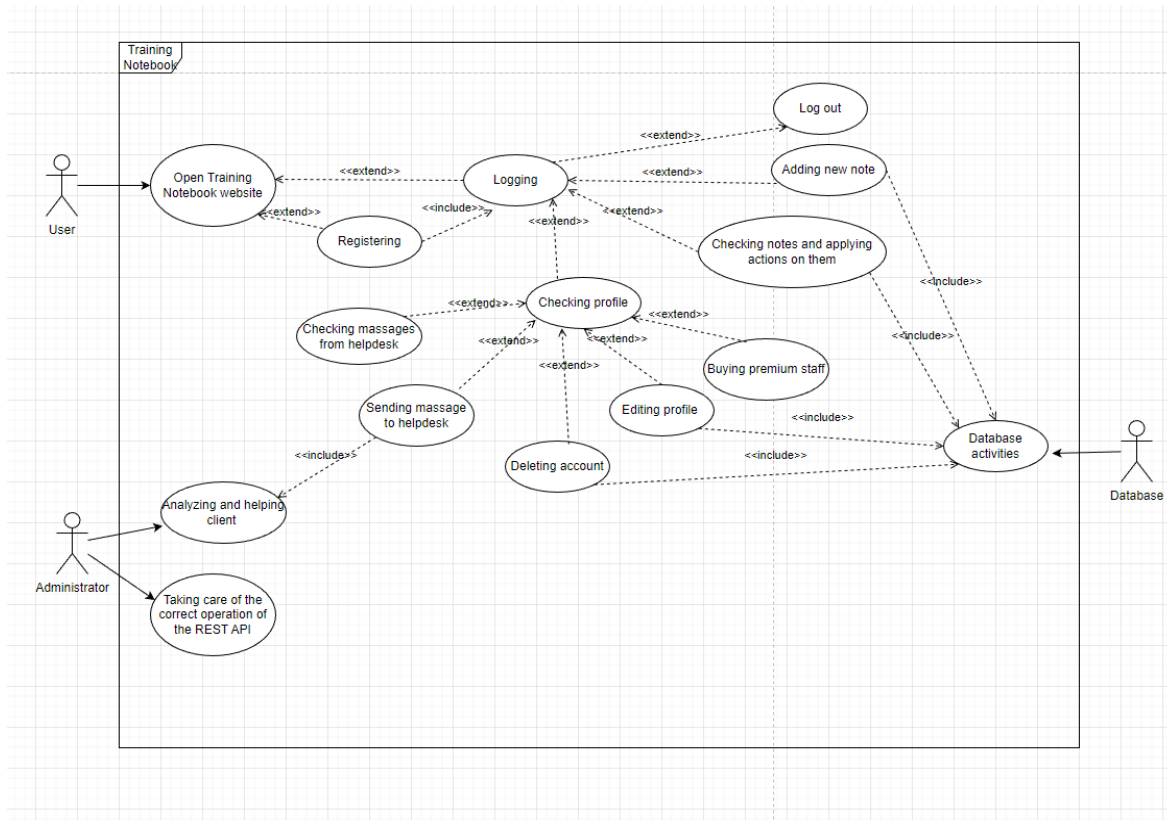
Additionally, log data is sent from the microservices and collected by the Prometheus Data Collector, which visualizes the data in the Prometheus Visualizer. Prometheus data is also used for generating documentation in Swagger.

This architecture diagram of the "Training Notebook" project illustrates the interactions between the various components and tools used in the project.



1) Use Case Diagram

It is a graphical representation of the interactions between users (actors) and the system, which allows for depicting how the system will respond to various user actions.



2) Use Case Descriptions

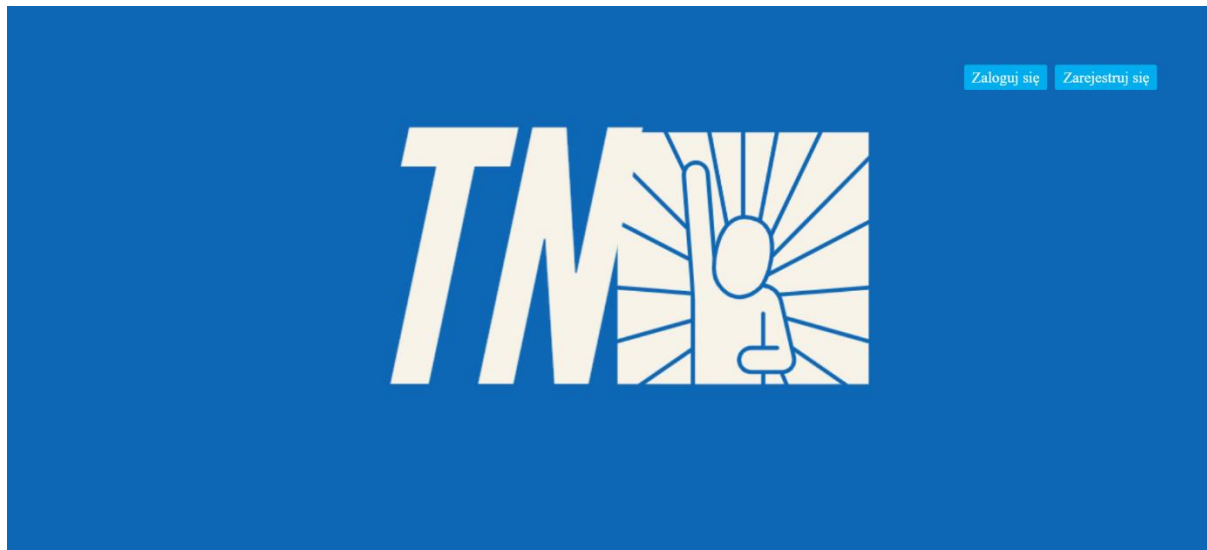
A detailed description of each use case depicted on the previously mentioned diagram, providing a comprehensive explanation of the actions taking place.

Name:	Open Training Notebook website
NUMBER:	1
ACTORS:	User
SHORT DESCRIPTION:	The user navigates to the Rest API website and is redirected to the homepage. They are faced with a choice to either create an account or log in.
PRECONDITIONS:	Desire to use the website:
POSTCONDITIONS:	The user is aware of the website they are on and proceeds to either log in or register.
MAIN EVENTS:	1.The user navigates to the homepage. 2a. The user proceeds to the login page by clicking the button labeled "log in." 2b. The user proceeds to the registration page by clicking the button labeled "register."
ALTERNATIVE FLOWS:	2.The user does not select any button and does not interact with the notepad.

**SPECIAL
REQUIREMENTS:**

2.The user directly accesses the login or registration page through a URL link instead of the homepage.
*Functional database and servers.

Start page prototype:



Name:	Registering
NUMBER:	2
ACTORS:	User
SHORT DESCRIPTION:	The user goes through the registration process by filling in the required information and is redirected to the login page.
PRECONDITIONS:	Need for account registration:
POSTCONDITIONS:	The user already has a registered account.
MAIN EVENTS:	<ol style="list-style-type: none">1.The user is on the registration page.2a. They choose the standard method and enter the required information to create an account.2b. They choose the registration method using email (OAuth2).3.The account is successfully created.4.The user is redirected to the login page.
ALTERNATIVE FLOWS:	<p>* If needed, the user can proceed to the login page using the button below, without going through the registration process.</p> <p>3.The account is not created due to invalid data or an existing user. An appropriate message is returned, and the user returns to step 2.</p>

	4. There is an error with the redirection, and the user needs to find an alternative way to access the login page.
SPECIAL REQUIREMENTS:	*Functional database and servers.

Register page prototype:

Name:	Logging
NUMBER:	3
ACTORS:	User
SHORT DESCRIPTION:	The user logs into their account.
PRECONDITIONS:	The user must have an account.
POSTCONDITIONS:	Authentication is performed, and access to the user's account and notes is provided.
MAIN EVENTS:	<ol style="list-style-type: none"> 1. The user enters their login credentials. 2. The credentials are verified, and access is granted. 3. The user is redirected to the notes page, where they can navigate further within their account.
ALTERNATIVE FLOWS:	2. The user does not have an account and is directed to the registration page
SPECIAL REQUIREMENTS:	*Functional database and servers.

Login page prototype:



Name:	Log out
NUMBER:	4
ACTORS:	User
SHORT DESCRIPTION:	The user logs out.
PRECONDITIONS:	The user must be logged in.
POSTCONDITIONS:	The user is logged out and redirected to the login page, losing access.
MAIN EVENTS:	1.The user clicks the "log out" button.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Checking profile
NUMBER:	5
ACTORS:	User
SHORT DESCRIPTION:	The user navigates to the account profile.

PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user is on the page displaying information about the logged-in person and can perform actions related to the profile.
MAIN EVENTS:	1.The user goes to the account profile page. 2a. They check the information from the helpdesk. 2b. They send a message to the helpdesk. 2c. They delete their account. 2d. They edit their account information. 2e. They purchase premium staff.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Adding new note
NUMBER:	6
ACTORS:	User
SHORT DESCRIPTION:	The user adds a new note.
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user adds a new note to their account.
MAIN EVENTS:	1.The user enters the note details (must enter at least the note title). 2.The user clicks the add button and adds the new note.
ALTERNATIVE FLOWS:	2.The user did not enter at least the note title, so they need to go back to step 1.
SPECIAL REQUIREMENTS:	*Functional database and servers.

Adding notes page prototype

Trenings: [Nowe notatki](#) [Profil](#) [Wyloguj się](#)

Nowa notatka treningowa

Title:
Running with friends

Duration:
Running

Start Date: 12.04.2023 17:16
End Date: 12.04.2023 19:16

Description:

kwiecień 2023

pon	uto	śro	czw	pią	sob	nie
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
1	2	3	4	5	6	7

Wycofaj Dodał

Add Note

Name:	Checking notes and applying actions on them
NUMBER:	7
ACTORS:	User
SHORT DESCRIPTION:	The user checks the notes they have added.
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user reviews the notes and can use various sorting options, as well as have the ability to edit or delete them.
MAIN EVENTS:	1.The user goes to the notes page and browses through them. 2a. If they are not looking for the latest notes, they sort them to find the desired ones. 2b. The user deletes a selected note. 2c. The user edits a selected note.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Notes page prototype:



Name:	Checking massages from helpdesk
NUMBER:	8
ACTORS:	User
SHORT DESCRIPTION:	The user navigates to the account profile.
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user reviews the current answers from the helpdesk.
MAIN EVENTS:	<ol style="list-style-type: none"> 1.While in the user profile, they click the "Check Answers from Helpdesk" button to go to the information page. 2.They review the answers and can sort them as needed.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Buying premium staff
NUMBER:	9
ACTORS:	User
SHORT DESCRIPTION:	The user makes a purchase of additional items on the website.
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user proceeds to purchase the add-ons.
MAIN EVENTS:	<ol style="list-style-type: none"> 1.The user is redirected to a page that offers them enhancements requiring payment (e.g., ad-free experience). 2.They choose the service, click on "Make Payment," and are redirected to the payment processing page. 3.After receiving confirmation of successful payment, the user gains access to the purchased add-ons.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Editing profile
NUMBER:	10
ACTORS:	User
SHORT DESCRIPTION:	The user edits their profile information or customizes the page according to their visual preferences.
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user makes changes to their account or visually customizes the page.
MAIN EVENTS:	<ol style="list-style-type: none"> 1.The user clicks on the "Edit" button next to the element in their profile that they are interested in. 2.They enter or select new data. 3.The data is updated.
ALTERNATIVE FLOWS:	3.The data may require confirmation, such as entering a password, before being updated.
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Deleting account
NUMBER:	11
ACTORS:	User
SHORT DESCRIPTION:	The user deletes their account.
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user deletes the account they are logged into.
MAIN EVENTS:	<ol style="list-style-type: none"> 1.The user clicks on the "Delete Account" button. 2.They enter their password and confirm it again. 3.The account is deleted. 4.The user is redirected to the homepage
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Sending messages to helpdesk
NUMBER:	12
ACTORS:	User
SHORT DESCRIPTION:	Sending a support request to the helpdesk
PRECONDITIONS:	The user must be logged in and be on the appropriate URL.
POSTCONDITIONS:	The user sends a support request.
MAIN EVENTS:	<ol style="list-style-type: none"> 1.In their profile, the user clicks on the "Send Message to Helpdesk" button and is redirected to a page requiring them to fill in the information. 2.They fill in the required details and send the message.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Analyzing and helping client
NUMBER:	13

ACTORS:	Administrator
SHORT DESCRIPTION:	Analysis of the message and providing a response
PRECONDITIONS:	The user must send a message and have an account.
POSTCONDITIONS:	The user's issue has been resolved.
MAIN EVENTS:	<ol style="list-style-type: none"> 1.The administrator receives the information in their inbox. 2.They analyze the problem. 3.They make an appropriate decision regarding the user's message.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	*Functional database and servers.

Name:	Taking care of the correct operation of the REST API
NUMBER:	14
ACTORS:	Administrator
SHORT DESCRIPTION:	Monitoring the proper functioning of the REST API
PRECONDITIONS:	The administrator must have access to the REST API and related components such as the database and server.
POSTCONDITIONS:	Monitoring and ensuring proper functioning.
MAIN EVENTS:	<ol style="list-style-type: none"> 1. The administrator monitors and ensures the proper functioning of the services.
ALTERNATIVE FLOWS:	
SPECIAL REQUIREMENTS:	

Name:	Database actions
NUMBER:	15
ACTORS:	Database
SHORT DESCRIPTION:	Performing actions by the database based on client requests
PRECONDITIONS:	Using a service that requires a database
POSTCONDITIONS:	Ensuring and saving the desired outcome for the client in the database based on the service
MAIN EVENTS:	1.Receiving a request to perform a service, such as saving a new profile name in the database. 2.Checking access rights. 3.Performing the service. 4.Returning information about successful execution.
ALTERNATIVE FLOWS:	3.Failure to perform the service due to lack of access. 4.Returning information with an error in execution.
SPECIAL REQUIREMENTS:	*Working database and servers.



3) Description of the actors

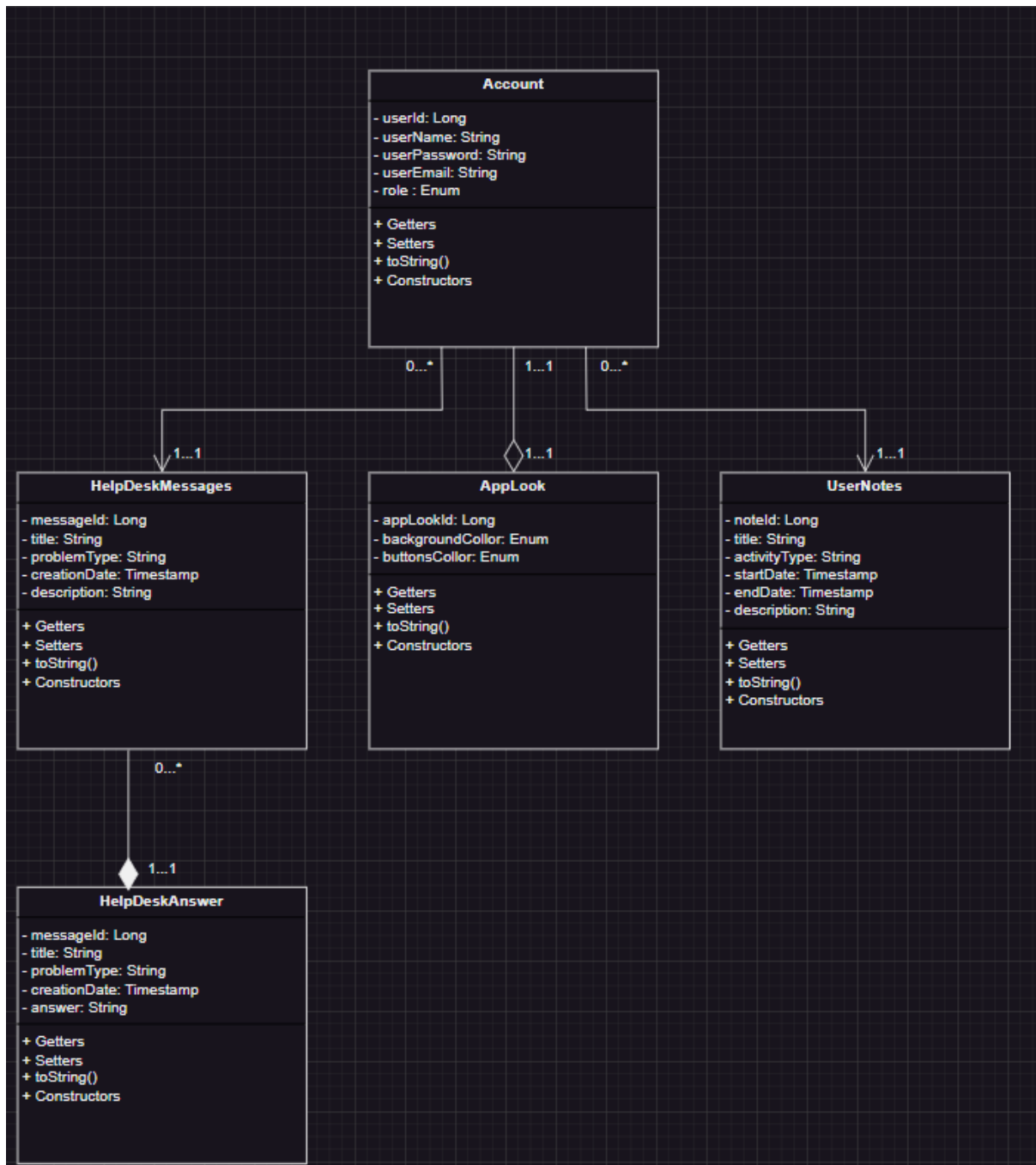
- a. User -> A user who utilizes the services offered by the website.
- b. Administrator -> An individual responsible for overseeing and maintaining the proper functioning and development of the project, as well as being in communication with the user.
- c. Database -> The system's database that is utilized for operations involving data manipulation.

4) Class diagram for entities in the project:

This diagram presents the connections between entities in the REST API.

- a. Account is the base class from which connections with other classes originate.
- b. AppLook is a class representing the API's appearance.
 - It must exist when creating an account because the website needs to have a design, and only one design is assigned to each account. It can be later edited or have premium features added.
 - This class is dependent on the Account class.
- c. UserNotes is a class representing user notes.
 - An account doesn't have to have notes, but it can have multiple notes.
 - Notes must be associated with an account initially, and the API focuses on managing notes for individual users, so there is a maximum of one account associated.
- d. HelpDeskMessages is a class representing messages sent by users.
 - An account can have sent messages, but it's not mandatory, and there can be multiple messages.
- e. HelpDeskAnswer is a class accessible only for accounts with the ADMIN role, containing answers to inquiries.
 - An inquiry can have an answer, but it's not mandatory, and there can be multiple answers.
 - This class is dependent on sending messages.

A visual representation of a class diagram



5) Design patterns to be used in the project:

- **Microservices Architecture Pattern**: Division into two microservices - backend and frontend. This pattern helps in scalability and maintainability as each microservice can be developed, deployed, and scaled independently.

- Repository Pattern: Using repositories in the backend to separate the database access logic from other layers. Repositories provide a unified interface for communication with the database and facilitate testing by abstracting data access.
- Session Pattern: Storing authentication and authorization tokens in the frontend API session. This ensures secure management of user sessions and access to protected resources.
- Endpoint Pattern: Creating appropriate endpoints in the backend that expose CRUD (Create, Read, Update, Delete) operations on data. Endpoints define the communication interface between the frontend and backend.
- Dependency Injection Pattern: Utilizing dependency injection (e.g., annotations like `@Autowired`) to manage dependencies between classes. This avoids strong coupling between classes and facilitates testing and project development.
- Serialization and Deserialization Pattern: Using this pattern to convert object data into a textual format (e.g., JSON) during communication between the frontend and backend.

3. Project planning

Sprints

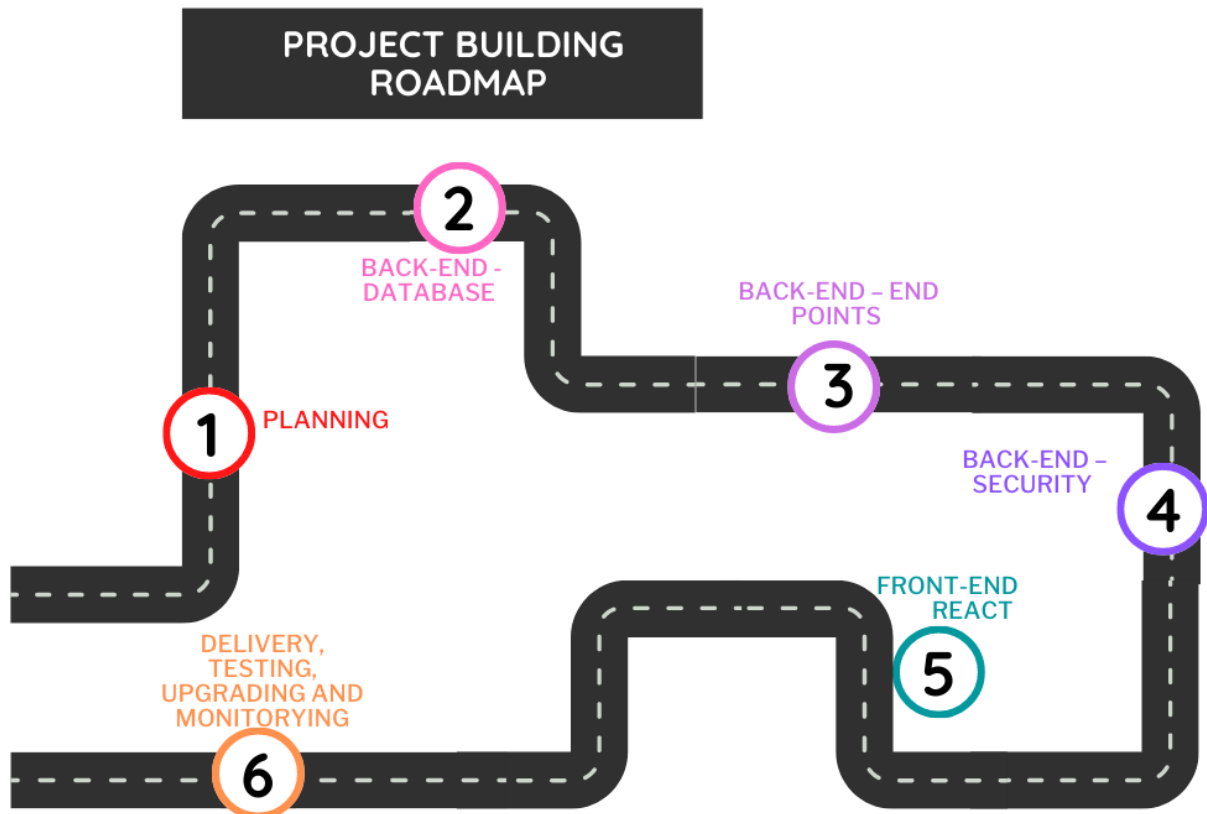
During the project implementation, we will utilize the concept of SCRUM, specifically sprints, which I have designed below. They will allow us to minimize wasted time and ensure greater productivity. Throughout the 1-2 week project duration, each day will consist of executing a pre-planned sprint.

How sprint will look like



- 1) Notate analysis and Scrum Task Board short term brainstorming.
- 2) Building a product, code writing segment
- 3) Testing the built part of the software
- 4) Sharing the produced code if it passed the tests correctly to e.g. GitHub
- 5) Making notes for the next sprint and completing the Scrum Task Board

Project building roadmap



PLANNING

- 1) Plan the entire system architecture - analyse documentation
- 2) Gather the necessary tools and resources in the workplace to perform the task
- 3) Reanalyse the architecture if there are any issues with using any of the tools and make necessary adjustments

BACK-END - DATABASE

- 4) Begin implementing the backend:
 - a. Create a new project
 - b. Provide all necessary dependencies
 - c. Establish and test the database connection
 - d. Create directories for classes to enhance future readability and separation of logic, such as controllers, services, entities, repositories

5) Entities:

- a. Create classes that serve as entities using JPA (getters, setters, constructors, toString())
- b. Define relationships between tables (e.g., OneToMany)

6) Repositories:

- a. Create interfaces that serve as repositories for the created entities. This stage allows for testing the database's correct creation.
- b. Verify elements such as:
 - i. Desired table relationships
 - ii. Correct variable types
 - iii. Proper naming according to logic and best practices
- c. Check the correctness of annotations in entity classes and repositories

7) Service Interfaces:

- a. Create interfaces containing method declarations necessary for database communication. These methods will be further implemented and filled with logic. Examples include getting single or multiple elements, setting elements, deleting elements, etc.

8) Service Implementation Classes:

- a. Create classes that implement the methods from service interfaces
- b. Inject dependencies from repositories
- c. Provide logic in methods and conduct tests on them

BACK-END - END POINTS

9) Controllers:

- a. Create classes that serve as controllers for REST API endpoints
- b. Inject the implemented service class with methods that provide services in the database
- c. Create necessary methods in the API that offer services such as DELETE, POST, GET, PUT. Ensure services are provided under specific URLs.
- d. Connection services between e.g notes service with account service

BACK-END – SECURITY

10) Security Configuration:

- a. Configuration of Keycloak with gateway
- b. Create a SecurityConfig class that configures the entire security. It should include the securityFilterChain method, which establishes various security aspects, including:
 1. Access roles for specific endpoints
 2. Filters and their actions
 3. Overall security behaviour
- c. Implement additional security elements, such as:
 1. JWT tokens
 2. OAuth2
 3. Retrieving and authenticating users from the database
- d. Conduct final tests for the entire backend, including adding SLF4J logging to unit and integration tests to facilitate tracking application behaviour and any potential issues during testing.

FRONT-END REACT

- 11) Visualization:
 - a. Connection with back-end microservice
 - b. Building visual aspects for back-end
- 12) Security
 - a. Connection auth-server Keycloak security with front-end

DELIVERY, TESTING, UPGRADING, AND MONITORING

Metrics and Logging:

- Use Jenkins to automatically collect metrics and logs from the application.
- Use Prometheus for generating reports.

DOCKER:

- Package the APIs in containers.
- Test the correct packaging of the project.

CLOUD SERVICE:

- Make the project publicly available by using an intermediary company that provides cloud hosting for full-time usage.

DOCUMENTATION:

- Documentation is maintained throughout the project, but in this stage, a thorough analysis is conducted, corrections are made, and the project is summarized. The project concludes with maintenance, care, and utilization of services, as well as revisiting previous stages if necessary.

4. Tools overview



- a. IntelliJ IDEA: A programming environment that provides efficient and advanced tools for Java programming. With intelligent code suggestions, automatic code completion, refactoring, and integration with the Spring Boot framework, IntelliJ IDEA speeds up the process of creating, testing, and debugging your REST API.



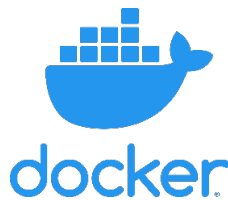
- b. PostgreSQL: A database that allows for storing and managing project data. PostgreSQL offers advanced relational features such as JOIN queries and views, making it easier to work with application data. Choosing PostgreSQL as the database ensures data consistency, scalability, and performance for REST API.



- c. Jenkins: A tool for continuous integration and application delivery. It enables the automation of the software build, testing, and deployment process, speeding up the development cycle of your REST API project. With Jenkins, you can quickly and safely deliver changes to the production environment, eliminating the possibility of human errors and ensuring continuous quality of your API.



- d. Spring Boot: A framework for building Java applications, particularly web applications and REST APIs. Spring Boot provides many ready-to-use solutions and simplifies configuration, allowing you to focus on implementing business logic.



- e. Docker: An application containerization platform that allows for seamless application portability across different environments without dependency issues. With Docker, you can create containers that contain all the dependencies of your REST API, ensuring consistency between the development and production environments. Deploying applications using Docker containers is fast, seamless, and provides resource isolation, eliminating compatibility issues.



- f. Hibernate: An Object-Relational Mapping (ORM) library that simplifies the mapping of Java objects to relational databases. Hibernate allows for more object-oriented data management, eliminating the need for

manual SQL query management. This improves the readability and performance of the REST API code.



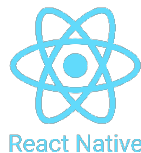
- g. Maven: A tool for dependency management and project building. Maven allows you to configure the project, manage libraries, and create a consistent and easily manageable development environment. It allows you to focus on API development rather than configuring and managing the environment.



- h. Postman: An API testing tool that facilitates the verification and testing of endpoints without the need to create additional API clients.



- i. Mockito: A library for creating unit tests and generating object mocks, enabling testing in isolation from other components. This makes it easier and more effective to test API behavior, resulting in higher quality and reliability of the project.



- j. React: JavaScript library for building user interfaces, widely used in front-end development. Offers a component-based architecture and virtual DOM for efficient rendering and responsive UI. Popular, with a large ecosystem and active community support. Ideal for creating dynamic and interactive user interfaces for REST APIs. React's modular and reusable component structure aligns well with the microservices architecture, allowing for independent development and deployment of

UI components that can be easily integrated with different microservices.

5. Authorization/authentication

To secure the project, I will integrate the services offered by Spring Security with the Keycloak tool, as Keycloak provides advanced authentication and authorization features, enabling effective access protection to the application.

Keycloak offers the following features:

- Single Sign-On and Single Logout (allows users to log in and log out once across multiple applications without having to repeatedly enter authentication credentials)
- Identity Brokering and Social Login (enables integration with external identity providers such as Google, Facebook, Twitter, allowing users to log in using their existing social accounts)
- User Federation (allows aggregation of users from different sources such as LDAP, Active Directory, databases)
- Fine-Grained Authorization Service (enables more precise management of user permissions and access to resources based on defined rules and policies)
- Centralized management and Admin Console (provides centralized management of users, clients, and system configuration)
- Client Adapters (provides adapters for various platforms and programming languages, facilitating integration with Keycloak)
- Standards-Based (built on security standards such as OpenID Connect, OAuth 2.0)

Keycloak advantages:

- Open-source (Open-source platform, providing transparency, community support, and the ability to modify and customize the source code as per specific requirements)

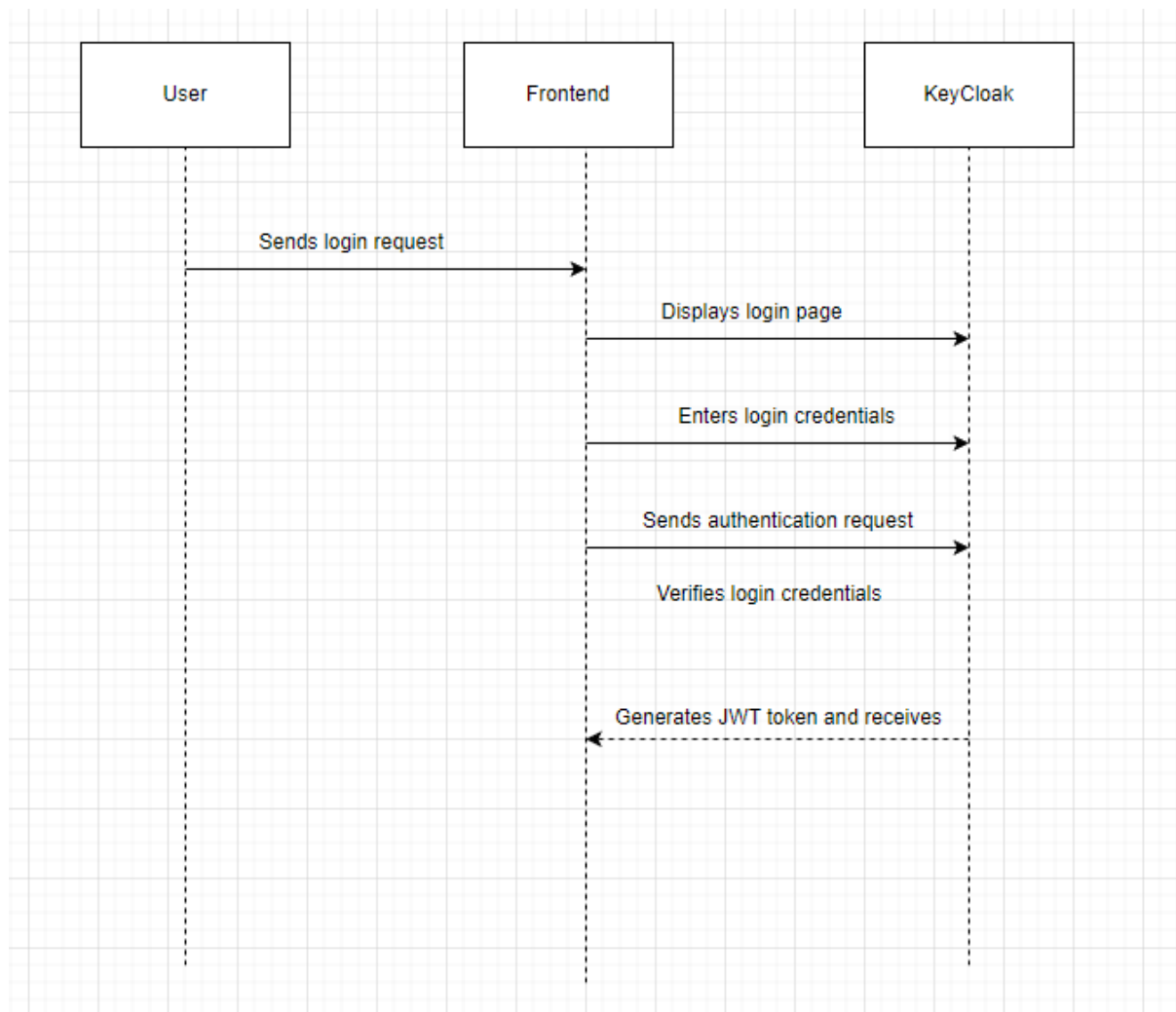
- Versatility (Offers a wide range of features and functionalities, making it suitable for various use cases and scenarios)
- Scalability (Designed to handle high loads and can scale horizontally to accommodate growing user bases and increased traffic)
- Security (Provides robust security measures, including authentication, authorization, and protection against common security vulnerabilities)
- Customizability (Allows customization and configuration of authentication and authorization flows, user management, and integration with other systems, providing flexibility to meet specific business needs)

How security will be implemented in the project:

This aspect can easily change because I never worked before with keycloak in microservices with React frontend

Authentication:

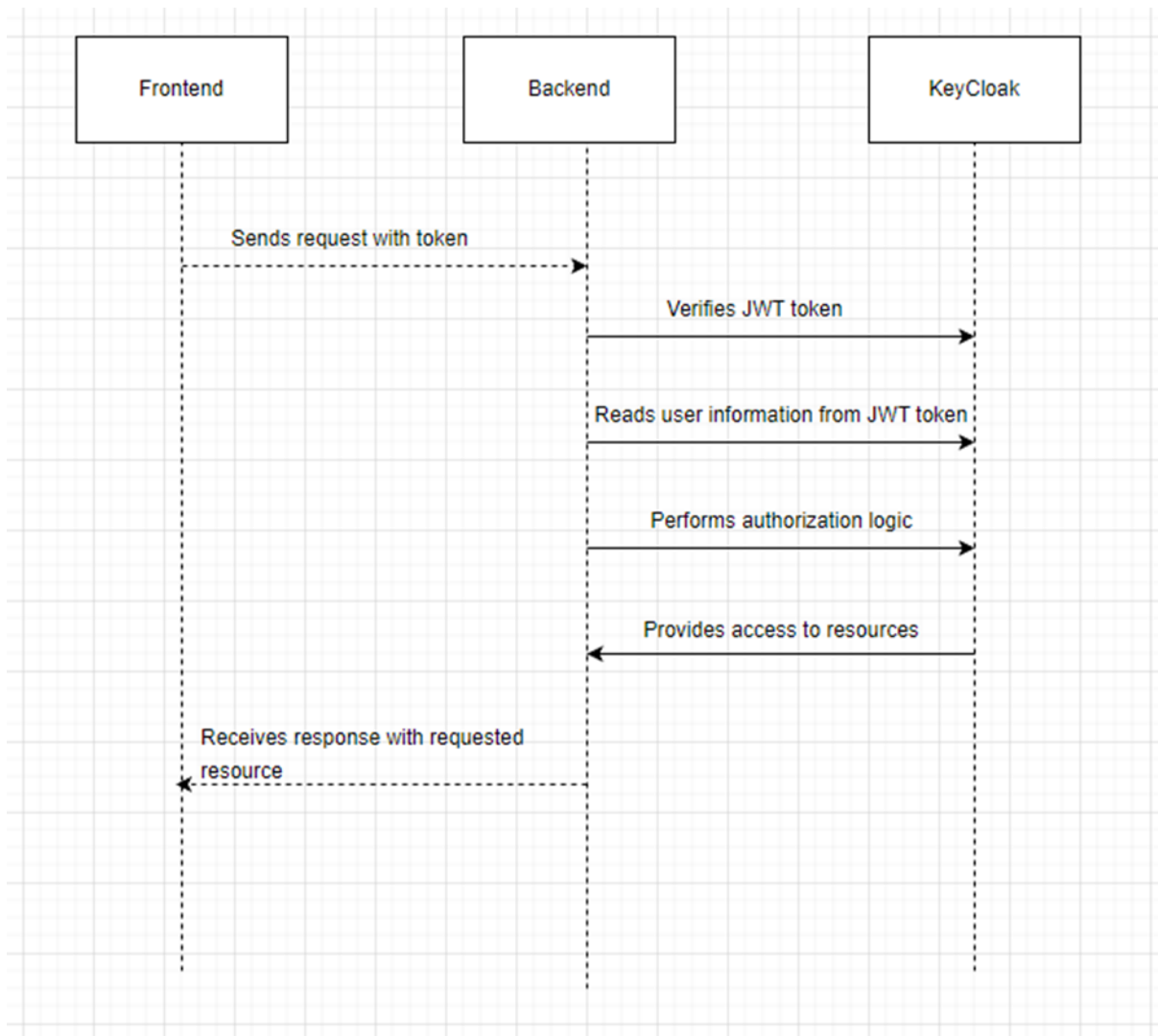
- The user logs in by Keycloak, and is redirected to the server where it authenticates (OAuth2),
- Keycloak verifies the login information and grants the user a JWT token that provides access.



Authorization:

- With each request to the backend, the frontend includes the JWT token in the header.
- The backend uses the public keys provided by Keycloak to verify the validity and signature compliance of the JWT token.
- If the token is valid, the backend reads user information (roles, permissions, etc.) from the JWT token.
- Based on the retrieved information, the backend performs authorization logic, checking if the user has the necessary permissions for the requested resource or belongs to a specific role.

- The backend only provides appropriate resources to authenticated and authorized users.



6. Security constraints

Potential threats and issues to consider during the project implementation and maintenance:

- A. Improper authentication and authorization: Improper implementation of authentication and authorization mechanisms can lead to vulnerabilities and security attacks. It is important to ensure that authentication methods are secure and that authorization mechanisms are properly configured.

- B. SQL Injection attacks: Insufficient filtering and validation of input data can lead to SQL Injection attacks, which can allow malicious execution of SQL code on the database. Always use parameterized SQL queries or ORM mechanisms to prevent such threats.
- C. Cross-Site Scripting (XSS): Inadequate filtering and validation of input data can enable Cross-Site Scripting attacks, which allow injecting malicious JavaScript code into web pages. Properly encode output data to prevent such attacks.
- D. Lack of protection against Brute Force attacks: If the authentication mechanism lacks proper protection against Brute Force attacks (e.g., account locking after multiple unsuccessful login attempts), there is a risk of user account compromise. This element will be addressed while developing the API.
- E. Disclosure of sensitive information: Improper management of sensitive information, such as user passwords or authentication keys, can lead to unauthorized disclosure. Take measures to properly secure such information, such as appropriate hashing and storage in a secure environment.
- F. Insufficient monitoring and logging: Lack of monitoring and logging of system activities can make it difficult to detect and trace improper actions. Ensure proper log configuration and implement a monitoring system that enables quick detection and response to potential threats.

7. Metrics and logging

Regarding logging, I have analyzed the market, and SLF4J seems to be the most suitable tool for my project. As for metrics, Prometheus is one of the most popular services in the market.



SLF4J allows for:

- Precise logging control, enabling the logging of various application activities to different sources such as files, console, or remote servers.
- Application monitoring.
- Easy troubleshooting and performance improvement by tracking and analysing actions.

Following are the reasons why SLF4J is preferred over Log4j (its biggest competitor):

- It acts as an abstraction layer.
- SLF4J is an open-source library that is independent of any specific logging implementation, eliminating the need to manage multiple logging configurations for different libraries.
- SLF4J provides placeholder-based logging, improving code readability by removing checks like `isInfoEnabled()`, `isDebugEnabled()`, etc.
- By using SLF4J's logging method, the cost of constructing logging messages (strings) is deferred until needed, resulting in CPU and memory efficiency.
- Since SLF4J uses fewer temporary strings, it reduces the workload for the garbage collector, resulting in better throughput and performance for the application.



Prometheus

Prometheus -> This component is responsible for scraping, storing and querying the metrics data. We will run the Prometheus server inside a new Docker container. When it comes to metrics data collection, Prometheus follows an HTTP

pull model or in other words, the Prometheus server periodically queries other applications for their metrics data. On a high level.

Why should we use Prometheus for monitoring REST APIs over other metrics solutions?

- Easy implementation: Prometheus offers easy deployment and configuration. It requires minimal code to start monitoring a REST API.
- Multidimensional data model: Prometheus's multidimensional data model allows you to label metrics with tags such as environment, API version, user, etc. This enables more detailed analysis and metric filtering.
- Active data collection: Prometheus independently collects metrics from REST APIs using a pull-based mechanism.
- Rich exporter library: Prometheus has a wide range of exporters that facilitate collecting metrics from various tools and frameworks.
- Powerful PromQL query language: PromQL is a query language specifically designed for Prometheus. It enables advanced metric analysis and aggregation, allowing for the generation of complex reports and charts.
- Rich monitoring features: Prometheus offers various monitoring features such as alerts, notifications, event logging, visualizations, and more.

How does Prometheus collaborate with other tools in the project?

Docker:

- Prometheus can monitor applications running in Docker containers. Exporters like Prometheus Docker Exporter can be configured to collect metrics from containers and provide them to Prometheus.

- Ready-made Docker images containing Prometheus and related tools like Grafana (for metric visualization) and Alertmanager (for managing alerts) can be used.

Jenkins:

- Jenkins can be configured to export metrics related to the CI/CD process, such as build time, success/failure count, duration, etc. Prometheus can collect these metrics and provide performance and stability data for the Jenkins process.
- Jenkins can also be configured to run performance or load tests as part of the CI/CD process, and Prometheus can monitor the results.

Kubernetes:

- Prometheus can be easily integrated with Kubernetes. There are tools for monitoring and collecting metrics from Kubernetes clusters, such as kube-state-metrics and node-exporter. These tools can be configured to export metrics to Prometheus.
- Ready-made solutions like Prometheus Operator simplify deploying Prometheus in a Kubernetes cluster and enable automatic scaling based on metrics.

Drawbacks:

- Data storage: Prometheus stores metric data in its own time-series database. This may require additional disk space management and data retention configuration, especially for projects generating large amounts of metrics.
- Metric export requirement: Prometheus requires applications to export metrics in the appropriate format (e.g., Prometheus exposition format) for collection and monitoring. This may require some code changes in the application or the use of additional metric exporting tools.

In summary, Prometheus is an excellent solution that aligns well with the project's tools and goals. I have chosen it particularly for its integrity and ease of collaboration.

8. Databases

Choosing SQL vs NoSQL

The main aspect of NoSQL is scalability, which can be useful. However, I have emphasized data consistency in the project, which will facilitate work on it. I value specific and rigid work rules, easier manipulation, and more precise data thanks to SQL queries.

Selected databases:

In the project, I am considering using one of the two largest databases on the market, namely PostgreSQL and Oracle Database. I have more experience with PostgreSQL, which can increase the speed of project implementation and its development in the future. Each database has certain arguments that distinguish it.



PostgresSql:

- ⊕ Value for Money: Oracle is a commercial solution that has steep pricing options, with additional payments required for extra features. PostgreSQL easily clinches this comparison since acquisition, installation and support is

completely free of charge. This factor alone can prove to be crucial when it comes to small and medium-sized organizations.

- ⊕ **Support:** Another win for the open-source solution. PostgreSQL has an extremely active community where patches, tweaks, updates and more can be found very easily. Even answers to questions that arise during installation or upgrades are easy to find with minimal delays. This is not the case with Oracle, where support costs money. Large organizations that choose to implement PostgreSQL can also opt for paid professional support, where services tend to be cheaper than their Oracle counterpart.
- ⊕ **Scalability:** While both solutions are quite capable in this category, PostgreSQL may have a slight advantage due to its open-source characteristics. Not only is it much lighter than Oracle, it doesn't require users to spend more money to expand infrastructure. PostgreSQL is capable of accommodating any volume of data.
- ⊕ **Compatibility:** Oracle has a robust language in PL/SQL, however PostgreSQL allows you to write language handlers in multiple languages (Python, R, etc.) directly in the database. PostgreSQL also clearly has the edge when it comes to compatibility with operating systems, which is extremely crucial in today's diverse and complex development environments. FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix and Windows are all compatible with PostgreSQL, which is a big advantage.



Oracle Database

- ⊕ **Scalability:** While both solutions are quite capable in this category, PostgreSQL may have a slight advantage due to its open-source characteristics. Not only is it much lighter than Oracle, it doesn't require users to spend more money to expand infrastructure. PostgreSQL is capable of accommodating any volume of data.

- ⊕ **Functionality** : Oracle Database has decades of experience and high levels of development expertise. It not only provides more transactions per second than PostgreSQL, but also arguably provides higher levels of security. However, it should be noted that many of Oracle's security features come at an added cost. Oracle is secure and ensures that user data is not tampered with through prompt updates. Its experience with various industries also gives it the upper hand. That doesn't mean, however, that PostgreSQL has poor functionality. It offers three levels of transaction isolation: Read Committed, Repeatable Read and Serializable. It is immune to dirty reads. Requesting a Read Uncommitted transaction isolation level provides read committed instead. PostgreSQL supports full serializability via serializable snapshot isolation (SSI).

In summary

Both technologies are well supported by the tools used in the project, such as Java, Spring, and Docker.

When to use PostgreSQL:

If cost savings, flexibility, and an active community are important to us.

When to use Oracle:

If we need high performance, advanced features, and security, and license costs are not a problem.

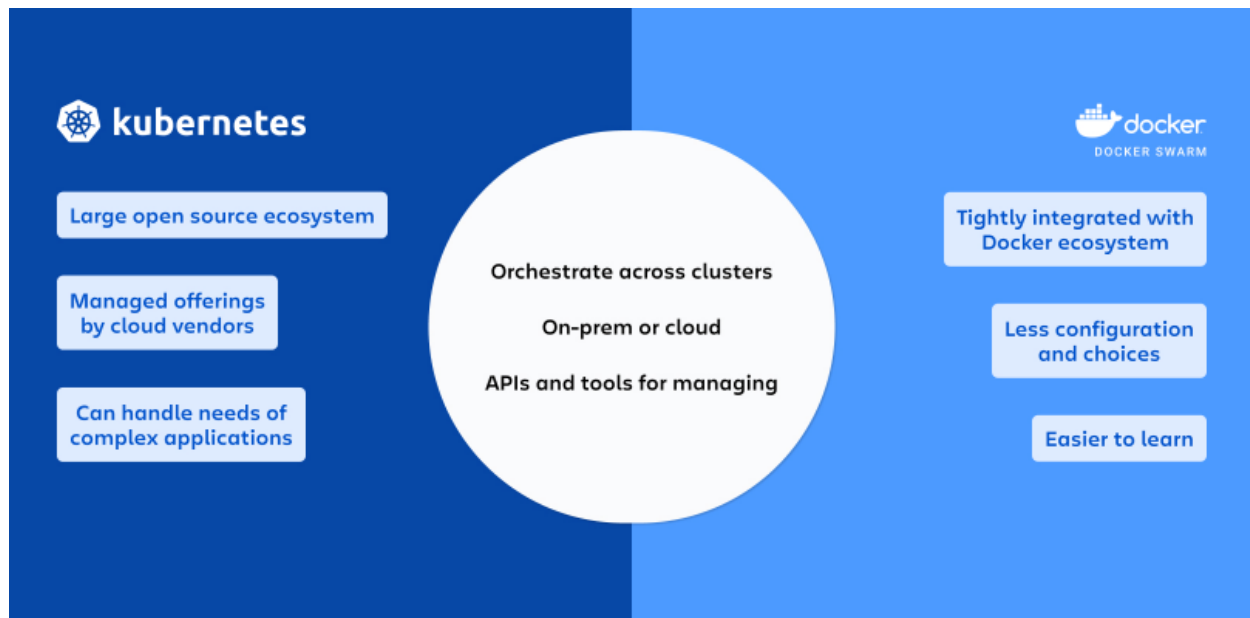
9. Containerization

Suggestions:

Docker -> Docker is a commercial containerization platform and runtime that helps developers build, deploy, and run containers. It utilizes a client-server architecture with simple commands and automation through a single API interface.

Kubernetes -> is a popular open-source platform for orchestrating containerized systems in a cluster of networked resources. Kubernetes can be used with or without the Docker platform.

Which one to choose:



Source(<https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker>)

Choice of Docker:

- Docker is a good choice when the REST API project is smaller, has a simple structure, and the priority is quick deployment and running of the application.
- Docker provides container isolation, allowing to avoid the "it works on my machine" problem.
- It is easier to use than Kubernetes and can be more easily managed by smaller teams.
- It may be suitable if there is no need for dynamic scaling of the application and the infrastructure is relatively simple.

Choice of Kubernetes:

- Kubernetes is a good choice when the REST API project is larger, more complex, and requires scalability and flexible application management.
- Kubernetes offers advanced orchestration features such as load balancing, automatic scaling, fault tolerance, and management of multiple containers across multiple hosts.
- It is more suitable if the application requires automatic management of multiple containers and dynamic scaling based on load.
- It can be beneficial if there are plans for cloud integration and higher service isolation.

How the choice will impact the REST API:

Choosing Docker can speed up the deployment and running process, especially for smaller and simpler projects. Docker provides container isolation, avoiding conflicts between applications and enabling faster deployment of changes.





Choosing Kubernetes can provide better scalability, flexibility, and automation in application management. This allows for easier handling of higher workloads and dynamically adjusting the number of containers based on actual application traffic.

Docker can be used as the runtime environment for containers in a Kubernetes cluster. This means that the benefits of both solutions can be leveraged by deploying the application in Docker containers and managing them with Kubernetes.

I believe that Docker fits well for a small project with a notebook and can be sufficient to handle the entire REST API, even considering future API feature development.

10. CI/CD

Comparing CI/CD Offerings by Top 3 Cloud Providers vs Jenkins:

	 Jenkins	 AWS CodePipeline	 Azure Pipelines	 Google Cloud Build
Tool Type	CI/CD	CI/CD	CI/CD	CI/CD
Free Version	YES	NO	YES	NO
Price	FREE	Pay as you go	Pay as you go	Pay as you go
Operating System	Windows, Linux, macOS	n/a	n/a	n/a
Open source	YES	NO	NO	NO
Difficulty Level	Medium	Medium	Medium	Medium
Plugins	5/5	3/5	4/5	3/5
Integrations	5/5	3/5	4/5	3/5
Platform	On-premise & cloud	Cloud	Cloud	Cloud
Kubernetes Support	YES	NO	YES	NO
External DB Server Required	NO	n/a	n/a	n/a
Built-in Git Repository	NO	YES	YES	YES
Version Control System Integration	GIT, Mercurial, TFS, SVN, Bazaar, CVS	GIT	GIT	GIT
Plugin Source	Internal store	Internal store	Internal store	GitHub
Authentication GitHub/Azure AD	GitHub/Azure AD	NO	Azure AD	NO
Does it support iOS/macOS?	YES	YES	YES	NO
Can it define Pipeline as code?	YES	YES	YES	YES
Does it support	YES	YES	YES	YES

Docker containers?				
--------------------	--	--	--	--



- 1) Jenkins is the most popular CI/CD tool currently available in the market. It offers the largest number of plugins and various integrations that allow performing all the required operations during the different steps of the pipeline. However, Jenkins is also one of the oldest tools, which means it falls behind other more modern tools.

Best for teams who want to use the most widely used solution in the market, providing the largest number of plugins and integrations, and who want to avoid being overly dependent on a cloud provider, as Jenkins is free to use.



AWS CodePipeline

- 2) AWS CodePipeline is a fully managed CI/CD service that helps automate release processes for fast and reliable updates to applications and infrastructure. It leverages the benefits of Amazon AWS cloud, such as integration with other internal services or the ability to pay for the service only as long as you need it.

Best for teams already using AWS cloud services.



Azure Pipelines

- 3) Azure Pipelines is a CI/CD service managed within the Microsoft Azure cloud. It enables creating CI processes in the cloud for Linux, macOS, and Windows systems. It also supports building web, desktop, and mobile applications.

Best for teams already using Microsoft Azure cloud services.



- 4) GCP Cloud Build is a CI/CD service managed within the Google Cloud Platform. It allows for quick software development in various programming languages, including Java, Go, Node.js, and others. Additionally, it supports deployment in multiple environments, such as virtual machines, Kubernetes, or Firebase.

Best for teams already using Google Cloud Platform services.

In summary

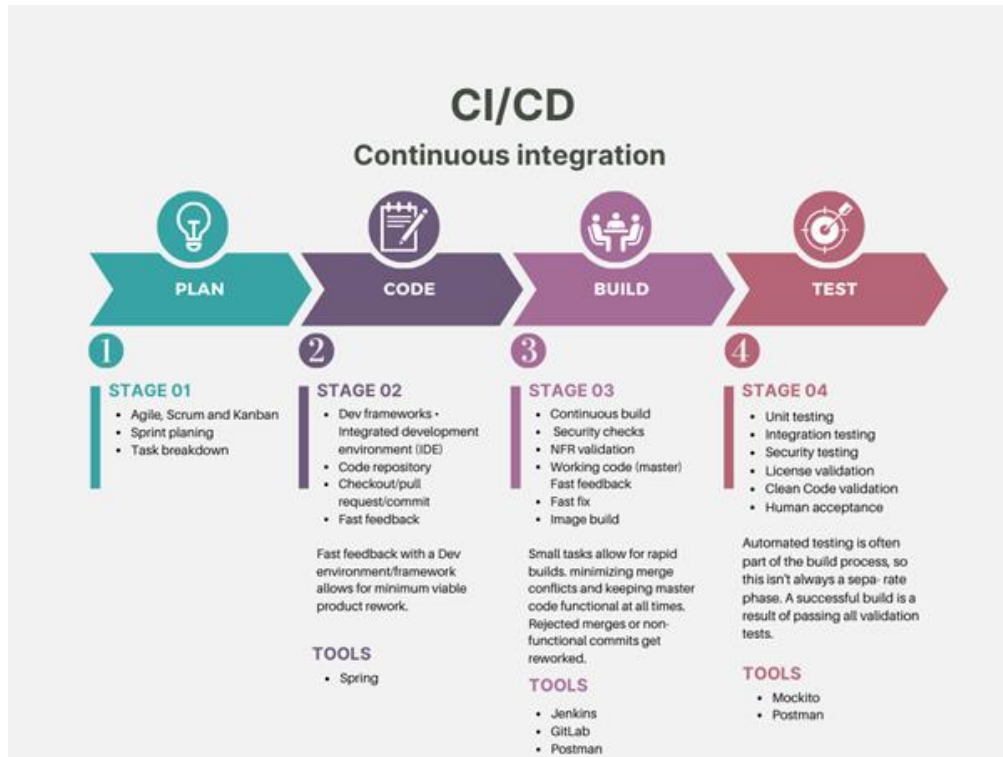
It is worth considering using a CI/CD solution provided by your cloud service provider. However, Jenkins is undeniably the best option in almost every aspect, which is why I would choose it for my project. It is also worth mentioning that although there are many CI/CD tools in the market, while many of them are generally inferior to Jenkins, they have some specialization that attracts specific customers. For example, some may provide a server that delivers software changes at high speed. If that particular factor is essential to us, it is worth considering that specific service provider.

Introduction to CI/CD with Jenkins for the project:

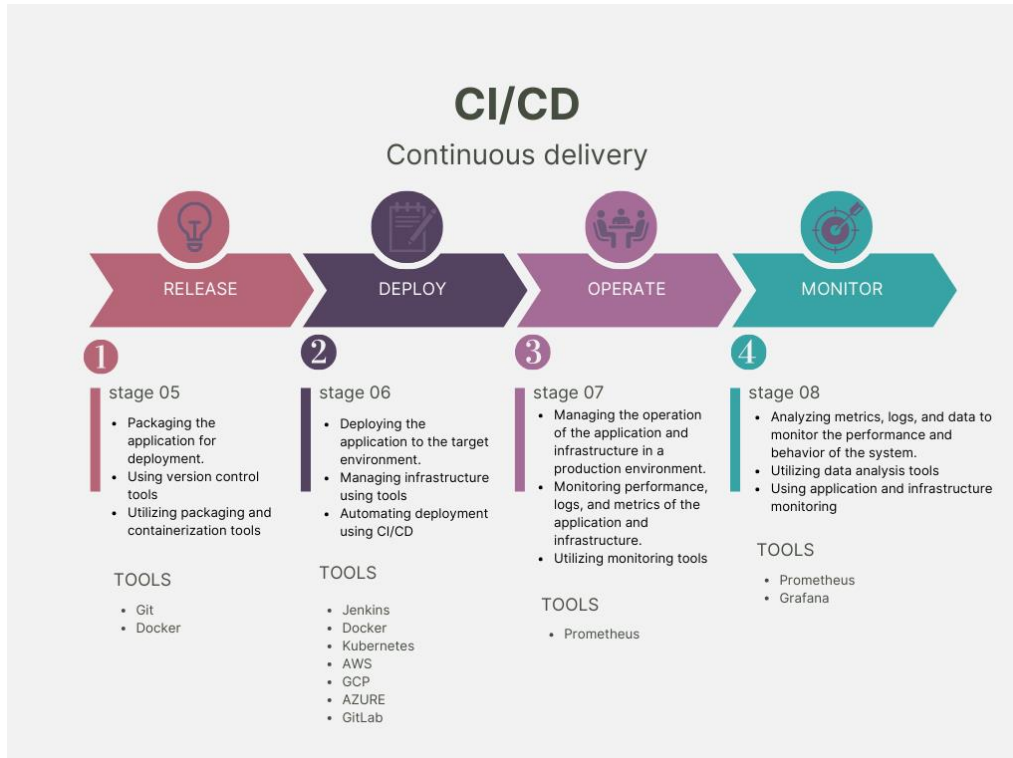
- 1) Configure Jenkins on your server or use a cloud service.
- 2) Create a new project in the Jenkins dashboard to represent your application project.
- 3) Configure the code repository, providing information about the code repository from which Jenkins will retrieve the application code. This can be a Git repository, SVN, or other repositories supported by Jenkins.

- 4) Create a CI task, configure the task responsible for building and testing the application. Specify the commands and scripts to be executed during the CI process, such as code compilation, running unit tests, generating documentation, etc.
- 5) Configure a CD task by adding another task responsible for deploying the application to the production environment. You can use tools like Docker, Kubernetes, or deployment scripts to automate this process.
- 6) Configure triggers to determine how Jenkins should respond to changes in the code repository. You can set up webhooks, schedules, or manual triggers to execute CI/CD tasks at the appropriate times.
- 7) Configure Jenkins to generate reports on build results, test results, and deployments. You can also integrate Jenkins with Prometheus for metric monitoring.

Plan for Continuous integration



Plan for Continuous delivery



11. Testing

Writing Back-end Code Process:

Unit Testing: You can use a testing framework like JUnit to create these tests. Unit tests allow you to verify the correctness of individual modules and components of the application.

Integration Testing: These tests check the interaction between different components of the system. You can write integration tests that simulate interactions with APIs, testing different scenarios, data flows, and integrations with other services.

Using Mocks: When testing APIs, especially in the early stages, you can use mocking tools to simulate the behavior of external dependencies or not yet properly implemented parts of the system. Mocks allow you to test different scenarios and cases without relying on real external services.

When developing a significant part of the back-end with controllers:

Postman: It enables sending HTTP requests and checking responses, making it a great choice if you don't have a front-end yet. You can use Postman to test functionality, data validation, and different resources and endpoints in the API. Additionally, it is worth mentioning its user-friendly interface.

Moreover, with Postman's integration with Jenkins, you can automate the API testing process. You can configure Jenkins to run Postman test collections on demand or at specific stages of the CI/CD process. This means you can automatically perform tests for functionality, data validation, and test different resources and endpoints in your API.

During the initial stages of setting up the front-end:

You can also leverage the tests mentioned in the early stages of back-end development.

During front-end finishing work when we primarily work on HTML/CSS/JS + controllers:

Conducting real browser testing is useful to test the website. Sometimes we may not receive any errors in the console, but through testing, we may encounter some hidden display issues. Additionally, examining the URL provides information if we encounter a "continue?" message after navigating to another page, indicating two redirections.

After project implementation for maintenance and improvement:

Performing tests using Prometheus metrics is useful for monitoring the performance and behaviour of the REST API system. It allows monitoring response times, analysing throughput, monitoring resource load, and setting alarms and notifications. Testing metrics in this regard provides better control over performance and optimization of the REST API system.

Testing in the cloud with selected providers:

AWS:

- AWS Device Farm: Used for testing mobile applications on various devices and platforms.
- AWS CodeBuild: Tool for automating the building, testing, and deployment of applications in the AWS cloud environment.
- AWS Load Testing Services: Offers services for testing application performance and load in the AWS environment.

Azure:

- Azure DevTest Labs: Enables the creation of test environments in the Azure cloud, allowing easy resource management and application testing.
- Azure DevOps: Application lifecycle management platform that offers testing and deployment features.

GCP:

- Cloud Test Lab: Tool for testing mobile applications on various devices in the Google Cloud environment.
- Cloud Deployment Manager: Allows the creation and management of test environments and automation of the application deployment process in GCP.

12. Bibliography

(Infrastructure)

<https://devcontentops.io/post/2021/05/azure-devops-vs-aws-devops-vs-gcp-devops>

<https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/?US>

<https://www.coursera.org/articles/aws-vs-azure-vs-google-cloud>

<https://www.pluralsight.com/resources/blog/cloud/compute-compared-aws-vs-azure-vs-gcp>

<https://cast.ai/blog/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud-platform/>

(Architecture)

<https://www.sztukakodu.pl/15-zasad-przy-budowie-rest-api-za-ktore-deweloperzy-cie-pokochaja/>

https://en.wikipedia.org/wiki/Sequence_diagram

(Authorization/authentication)

<https://www.baeldung.com/spring-boot-keycloak>

<https://www.youtube.com/watch?v=vmEWywGzWbA>

<https://javaleader.pl/2021/11/21/integracja-keycloak-z-spring-boot/>

(Logging and metrics)

<https://blog.teonisoftware.com/adding-logging-and-metrics-to-a-java-rest-api>

<https://prometheus.io/docs/introduction/overview/>

(Databases)

<https://hevodata.com/learn/postgresql-vs-oracle/>

<https://www.linkedin.com/pulse/difference-between-oracle-postgresql-ipspecialistofficial>

<https://www.datavail.com/blog/postgresql-vs-oracle-lets-compare/>

(Containerization)

<https://www.atlassian.com/microservices/microservices-architecture/kubernetes-vs-docker>

<https://www.dynatrace.com/news/blog/kubernetes-vs-docker/>

(CI/CD)

<https://www.czerwinski.it/pl/2022/03/27/znajdz-swoje-najlepsze-narzedzie-ci-cd/>

<https://dev.to/aws-builders/aws-azure-or-maybe-google-cloud-which-provider-has-the-best-cicd-system-866>

(Testing)

<https://www.geeksforgeeks.org/rest-api-testing-and-manual-test-cases/>