Explanatory note

on the topic: "IoT service of tracking and identification of objects"

Kyiv - 2024

CONTENT

LIST OF SYMBOLS AND ABBREVIATIONS

ACID - Atomicity, Consistency, Isolation, Durability

ADC - Analog-to-Digital Converter

API - Application Programming Interface

CRUD - Create, Read, Update, Delete

DTO - Data Transfer Object

GPIO - General Purpose Input/Output

GND - Ground

HTTPS - HyperText Transfer Protocol Secure

IDE - Integrated Development Environment

IoC - Inversion of Control

IoT - Internet of Things

I2C - Inter-Integrated Circuit

JAR - Java ARchive

JDBC - Java Database Connectivity

JMS - Java Message Service

JPA - Java Persistence API

JPQL - Java Persistence Query Language

JSON - JavaScript Object Notation

Junit - Java Unit Testing Framework

JVM - Java Virtual Machine

JWT - JSON Web Token

LoRaWAN - Long Range Wide Area Network

MOSI - Master Out Slave In

MISO - Master In Slave Out

OXM - Object XML Mapping

OLED - Organic Light Emitting Diode

ORM - Object-Relational Mapping

PWM - Pulse-Width Modulation

REST - Representational State Transfer

REST API - Representational State Transfer Application Programming Interface

RFID - Radio-Frequency Identification

RST - Reset

SCK - Serial Clock

SDA - Serial Data Line

SPI - Serial Peripheral Interface

SQL - Structured Query Language

STA / **AP** / **STA** + **AP** - Station/Access Point/Station + Access Point

TCP / **IP** - Transmission Control Protocol/Internet Protocol

UART - Universal Asynchronous Receiver-Transmitter

UML - Unified Modeling Language

URL - Uniform Resource Locator

Vin - Voltage In

VCC - Voltage at Common Collector

WAR - Web Application ARchive

Wi - **Fi** - Wireless Fidelity

XML - eXtensible Markup Language

DB - Database

IN STUP

In today's world, where technologies are at the forefront of development, the concept of the Internet of Things is becoming not only important, but also an integral part of many industries. The integration of physical devices into the network environment gives them the ability to collect, exchange and process data. Identification and tracking of objects is one of the key aspects of this concept, which has great potential in solving the problems of effective management and control of various objects in various fields of activity.

The object of research of this work is the system of identification and tracking of objects, which uses *IoT* technologies to ensure communication between devices and digital infrastructure. The subject of the study is the methods and tools of *IoT* service implementation to optimize identification and tracking processes.

The purpose of this work is to develop a universal *IoT* service that provides effective identification and tracking of objects using radio frequency identification integrated into a single network. The implementation of this service requires solving the following tasks: development of the system architecture, selection and adaptation of technological solutions, programming of the service's functionality and its testing.

The result of the development of this *IoT* service will open wide opportunities for its use in industry and other sectors, ensuring increased efficiency of object management, optimization of logistics flows and improvement of the overall security of the system.

1 ANALYSIS OF THE SUBJECT FIELD

The main goal of *IoT service development* is object tracking and identification. In this section, wireless technologies and methods of object identification, analogues of commercially available identification and tracking services, their capabilities and applications will be considered.

1.1 Technologies for identification

1.1.1 QR codes

QR codes are two-dimensional barcodes that can store much more information than traditional one-dimensional barcodes. They were created in 1994 by the Japanese company *Denso Wave* . Their main advantages are fast and convenient reading using a camera or a special scanner, as well as the ability to store various information, such as text, *URLs* , contact data, hyperlinks, etc.

Figure 1.1.1 – Laser *QR* code scanner

QR codes are widely used in various fields, such as marketing, advertising, logistics, inventory, medicine, education, etc. In advertising materials, *QR* codes allow users to quickly go to websites , receive additional information about goods

and services, and make purchases. In the field of logistics and inventory, it is used to track goods and quickly access data about them.

One of the most important advantages *of QR* codes is their wide application and ease of use. They allow you to effectively transfer information and provide a convenient way to access additional content. However, in order to read them, it should be borne in mind that the device has a camera and requires additional software. In addition, the text in *the QR* code can be damaged or erased, which will lead to the loss of information. [1]

1.1.2 Wi-Fi and Bluetooth

Wireless networks such as *Wi-Fi* and *Bluetooth* are key technologies for wireless communication between different devices. They allow you to transfer data directly between devices that are within a certain range. Such networks have a wide range of applications, including logistics and inventory.

Figure 1.1 .2 – *Wi-Fi* and *Bluetooth logos* [2]

In the field of logistics and inventory, *Wi - Fi* and *Bluetooth* can be used to track the movement of equipment and materials in the enterprise. For example, devices with *Wi-Fi* or *Bluetooth* can be attached to moving objects, such as pallets of goods or equipment. Thanks to this, the collection of data on the movement of goods becomes more automated and efficient.

In addition, these technologies can be used to automate inventory management processes. For example, using *Wi-Fi* or *Bluetooth,* you can monitor stock levels, automatically update stock information, and place replenishment orders when supplies are low or low.[3]

1.1.3 NFC

NFC is a wireless technology that allows the exchange of data over a short distance (usually up to 10 centimeters) between two devices, both of which are equipped with *NFC* . This technology is based on a radio frequency identification protocol, but it allows two-way data exchange between devices, rather than simply reading information from a tag.

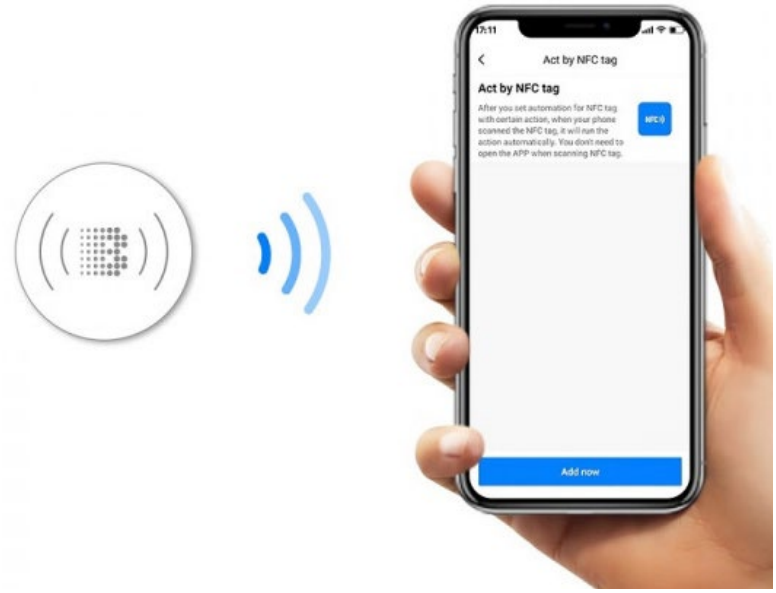


Figure 1.1.3 – Reading *an NFC* tag from a smartphone [4]

NFC tags are widely used in various fields, including payment systems, loyalty programs, contactless access, contact sharing, interactive advertising materials, transportation systems, and much more.

One of the main advantages *of NFC* is its convenience and ease of use. It allows you to quickly and easily exchange data between devices without the need for a physical connection or entering passwords.

However, the disadvantage *of NFC* is its limited range, which limits its use in some cases, such as large warehouses or transport networks. It is also worth noting

that *NFC* technology may require special reading equipment, which may be additional costs for businesses. [5]

1.1.4 RFID

RFID is a wireless identification technology that uses radio frequency signals to transmit data between *RFID* tags and readers. This technology allows non-contact identification and tracking of objects that have *RFID* tags, even at long distances and in conditions of limited visibility.

Figure 1.1.4 – *RFID* tag identification from the reader

The main components of *the RFID* system include tags are small devices that contain a microchip and an antenna for wireless communication. Tags are attached to objects to be identified or tracked.

Readers read and record data on *RFID* tags. Readers usually have an antenna to interact with the tags and a data processor to transmit information to a computer or other devices.

The system software manages the operation of *the RFID* system, data reading and processing, integration with other systems. [6]

So, for the further development of the Internet of Things service for tracking and identification of objects, the most appropriate technology is *RFID* . It allows contactless identification from a long distance, has high speed and is effective in various conditions.

1.2 IoT services

The Internet of Things is a concept that encompasses a wide network of physical objects that interact with each other and with the environment through the Internet. These objects can include various devices, sensors, software, and other technologies that have the ability to exchange data. The main goal *of IoT* is to create "smart" systems capable of automating processes, improving efficiency and providing new opportunities for interaction.

Figure 1.2.1 – Display of the *IoT* network concept [7]

Devices and sensors are physical objects that can collect data from the environment. They can be equipped with sensors for measuring temperature, humidity, pressure, light, etc.

The data collected by the devices is transmitted over the network to the processing center. This can be done using various communication protocols, such as *Wi-Fi, Bluetooth, Zigbee, LoRaWAN* , and others.

Processing and analysis of collected data can be stored and processed in cloud systems or local servers. Analysis of this data allows you to obtain useful information and make appropriate decisions.

Information and analytical results can be presented to users through various interfaces, such as mobile applications, web interfaces, control panels.

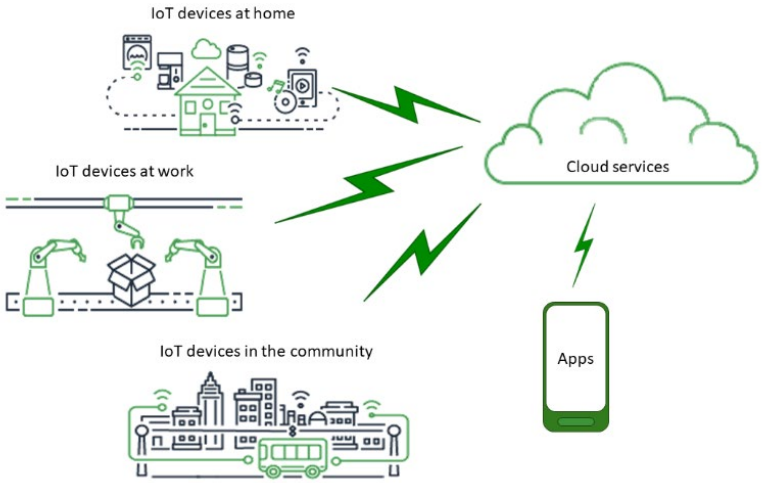


Figure 1.2.2 - Concepts of the "Internet of Things" [8]

The IoT service is a concrete implementation of the *IoT concept* , which ensures the integration of physical objects into a single system to achieve certain goals. Such services provide users with the ability to manage devices, monitor their status, receive data in real time, and perform analytics. [9]

1.3 Overview of existing solutions

There are several key players in the market that offer end-to-end *RFID* solutions, including *Zebra Technologies, Impinj, Avery Dennison, Honeywell* and *Alien Technology* . These companies offer a variety of hardware, including labels, scanners, printers, and specialized software , that provide high reliability and performance in a variety of applications. However, the high cost of solutions and the need to use specialized equipment are significant barriers to their widespread implementation.

1.3.1 Zebra Technologies

Zebra Technologies offers comprehensive *RFID* solutions that include printers, scanners and asset management software. They are used in logistics, retail, healthcare and manufacturing.

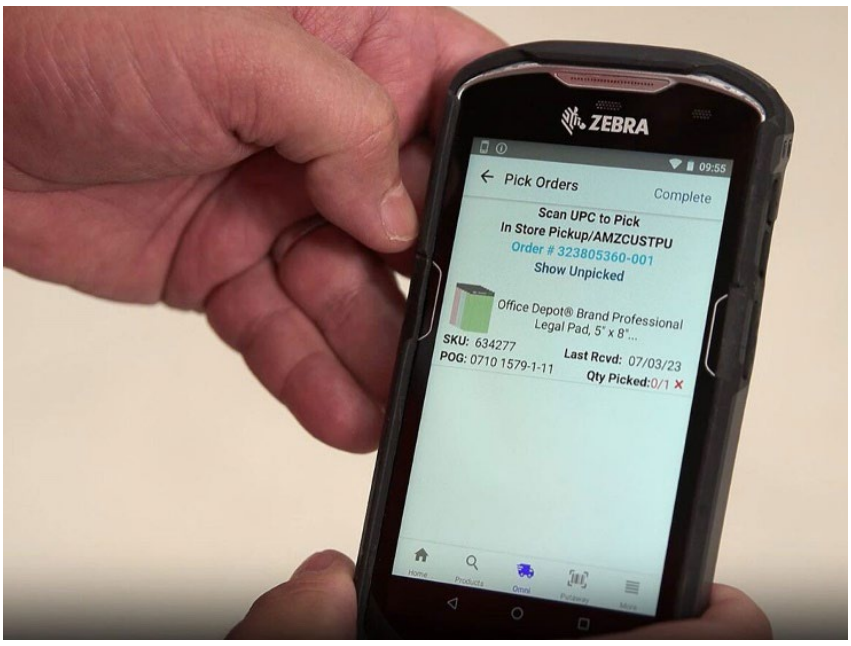


Figure 1.3.1 - *Zebra Technologies* service

Advantages:

- High accuracy inventory and real-time tracking.
- A wide range of equipment and software.

Disadvantages:

- High cost of equipment and implementation.
- Dependence on specific equipment models (*Euristiq*). [10]

1.3.2 impinj

Impinj offers complete *RFID* solutions including tags, readers and software. Their platform is often used in retail, logistics and manufacturing.

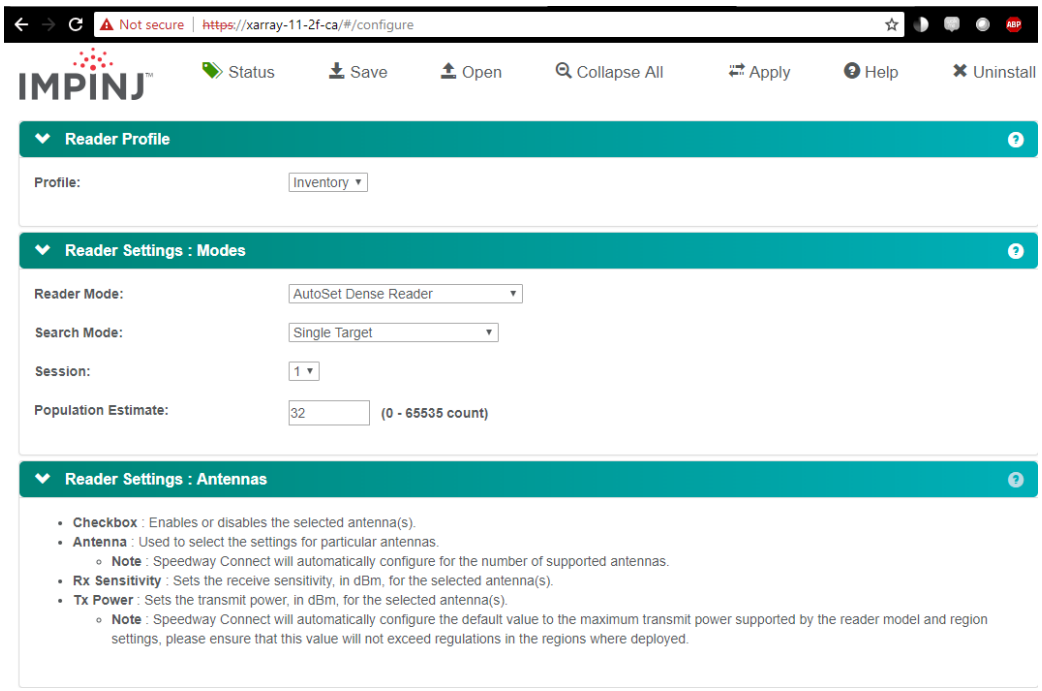


Figure 1.3.2 – *Impinj Speedway* service

Advantages:

- High reliability and accuracy of data.
- High performance in real conditions.

Disadvantages:

- The high cost of the solution.
- The need for specialized equipment (*DipoleRFID*).

1.3.3 Avery Dennison

Avery Dennison creates *RFID* tags and asset tracking solutions in various industries such as retail, healthcare and logistics.

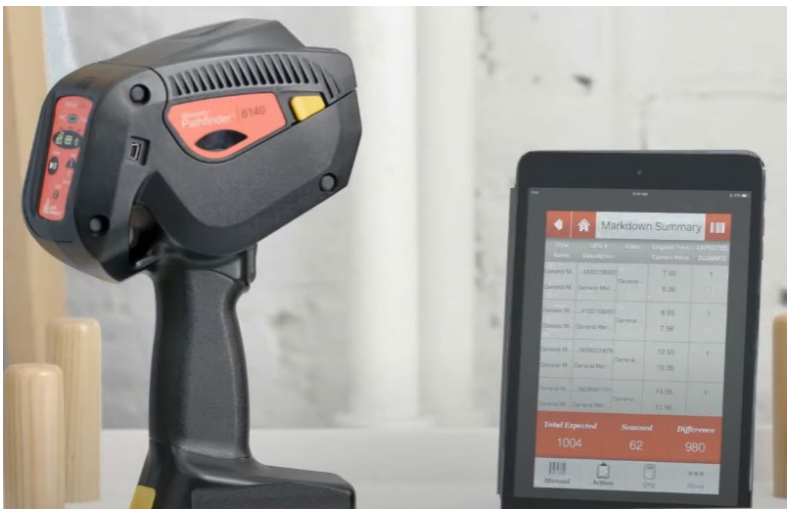


Figure 1.3.3 – *Avery Dennison* service [11]

Advantages:

- Increasing supply chain visibility.
- Reduction of product losses.

Disadvantages:

- High implementation costs.
- Dependence on certain equipment (*DipoleRFID*). [12]

1.3.4 Honeywell

Honeywell provides solutions for automatic identification and data collection, including *RFID* technologies. They are used in retail trade and logistics.

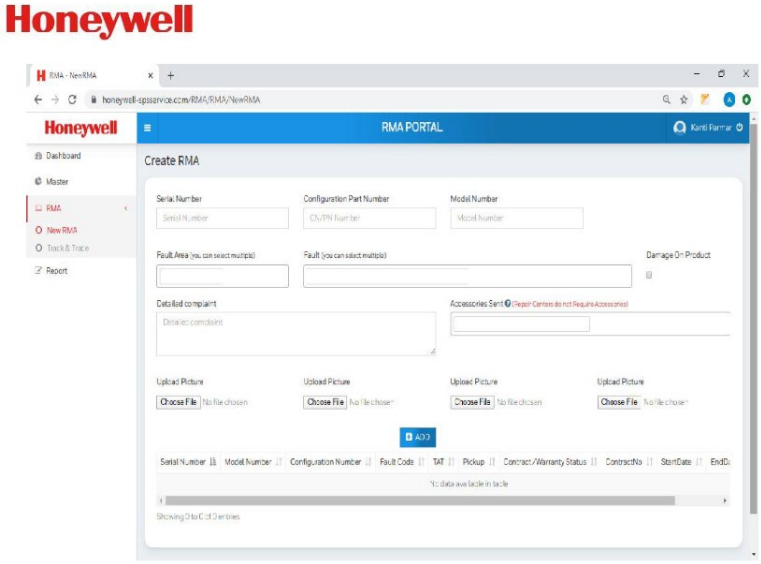


Figure 1.3. 4 – *Honeywell* service

Advantages

- Reliability and durability of solutions.
- A wide range of applications.

Disadvantages:

- High cost of implementation.
- The need to use specialized equipment . [13]

1.3.5 Alien Technology

Alien Technology offers a wide range of *RFID* products, including tags, readers and antennas. They provide high performance and reliability in areas such as manufacturing, warehouses, retail and logistics.

Figure 1.3. 5 – Alien Technology service

Pros:

- High performance and reliability.
- A wide selection of products.

Disadvantages:

- The high cost of the solution.
- Dependence on the equipment of a specific manufacturer [14]

Therefore, all these technologies have a high implementation cost and depend on the specific equipment manufacturer. This limits their availability and versatility for different users. The developed *IoT* service for tracking and identifying objects based on *RFID* tags will be more accessible to users, as it can be used by any equipment with access to the Internet. This will allow it to be used both in the private and corporate sectors, from small to large enterprises in various industries, without the need for specialized devices and binding to a specific manufacturer. The service will provide a comprehensive approach to data management, which allows you to receive accurate and up-to-date information about the movement and state of objects in real time.

2 CHOICE OF CURRENT TECHNOLOGIES FOR THE IMPLEMENTATION OF TRACKING AND IDENTIFICATION SERVICE

Choosing the right software, tools, and development environment is a critical step in the process of creating any software product. The right choice not only greatly increases development productivity, but also minimizes the risks associated with malfunctions and ensures greater stability of the product in the future.

Integrated development environments (*IDEs*) and other tools provide a large set of features that make it easier to write, test, and debug code. In addition, the use of modern frameworks and libraries can help in the effective integration of the latest technologies and development methods. The choice of the optimal set of tools also affects the project's scalability and its adaptation to the growing demands of the market.

2.1 Selection of programming language

the popular and stable programming languages for developing server applications is Java . A general-purpose object-oriented programming language developed by *Sun Microsystems* and released in 1995. *Java* is known for its platform independence due to its use of *the Java Virtual Machine* (*JVM*). Java is widely used in corporate applications, banking, and scientific research.

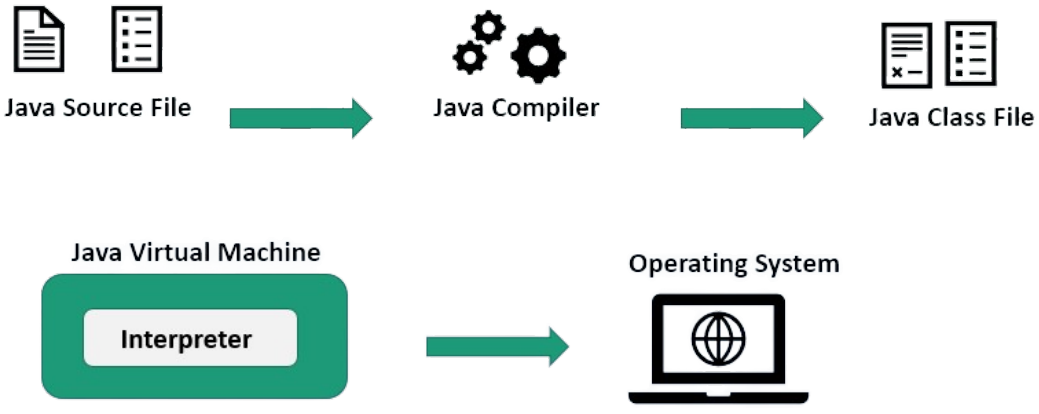


Figure 2.1.1 – Principle of operation *of J VM* [15]

Java is the optimal choice for developing the server part *of the IoT* tracking and identification service. Thanks to *the JVM , Java* applications can run on any operating system, which provides flexibility and ease of deployment in different

environments. The use of compilation allows you to achieve a high speed of program execution, which is especially important for processing a large number of requests in real time, typical of *IoT* services. This language is well-suited for the development of large distributed systems due to its multithreading and distributed processing capabilities, which allows services to be scaled efficiently to meet growing requirements.

Built-in security mechanisms protect applications from various types of attacks, which is critical for *IoT* services that work with a large number of sensors and other devices. A large number of libraries and frameworks, such as *Spring* , *Hibernate* , and many others, facilitate the development, testing and maintenance of applications, allowing to quickly implement new functions and ensure stable operation of the service. In addition, *Java* is a mature programming language with a long history of successful use in enterprise applications, which guarantees its reliability and stability. [16]

Considering all these advantages, *Java* is an excellent choice for developing the backend of an *IoT* service that requires high performance, security, scalability, and reliability.

2.2 Selection of the development environment

Choosing a development environment is critical to effective software development. Integrated development environments (*IDE*) provide a wide set of tools and functions that significantly speed up the process of writing, testing and debugging code.

Using a modern *IDE* allows you to focus on the logic of the program, and not on routine tasks, which reduces the likelihood of errors and increases productivity. Modern *IDEs* support integration with version control systems, automatic code completion, and also provide built-in tools for analysis and debugging, which makes them indispensable for writing code.

2.2.1 IntelliJ IDEA

IntelliJ IDEA is an integrated development environment designed for programming in *Java* , as well as other languages such as *Kotlin* , *Scala* , *Groovy* and others.

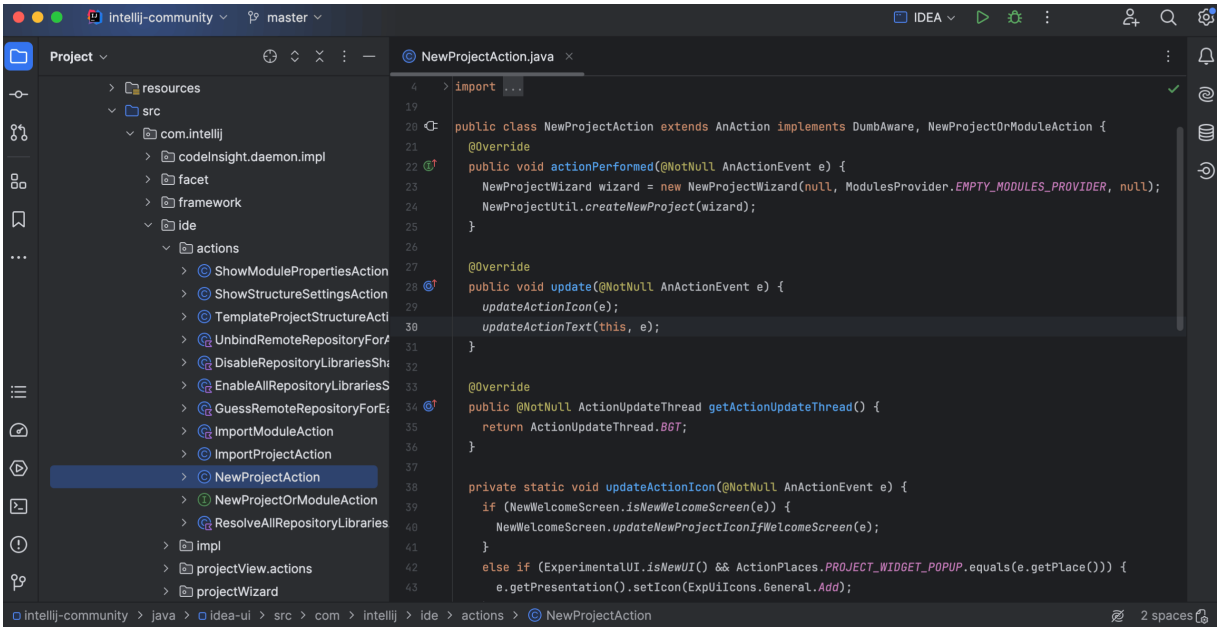


Figure 2.2.1 - *IntelliJ IDEA development environment*

It was developed by *JetBrains and is considered one of the most popular tools for Java* software development . *IntelliJ IDEA* provides developers with a wide range of tools for convenient work with code, including automatic code completion, error correction hints, a built-in debugger, a version control system, support for various technologies, and much more. It has excellent performance, speed and high level of support. *IntelliJ IDEA* has an active user community and many extensions (plugins) that allow you to adapt the development environment to the specific needs of a developer or project. [17]

2.2.2 Eclipse

Eclipse is an integrated development environment used for programming in various programming languages, including *Java* . Originally developed by *IBM* , then open sourced by the *Eclipse Foundation* , it became known for its flexibility, extensibility, and wide range of capabilities.

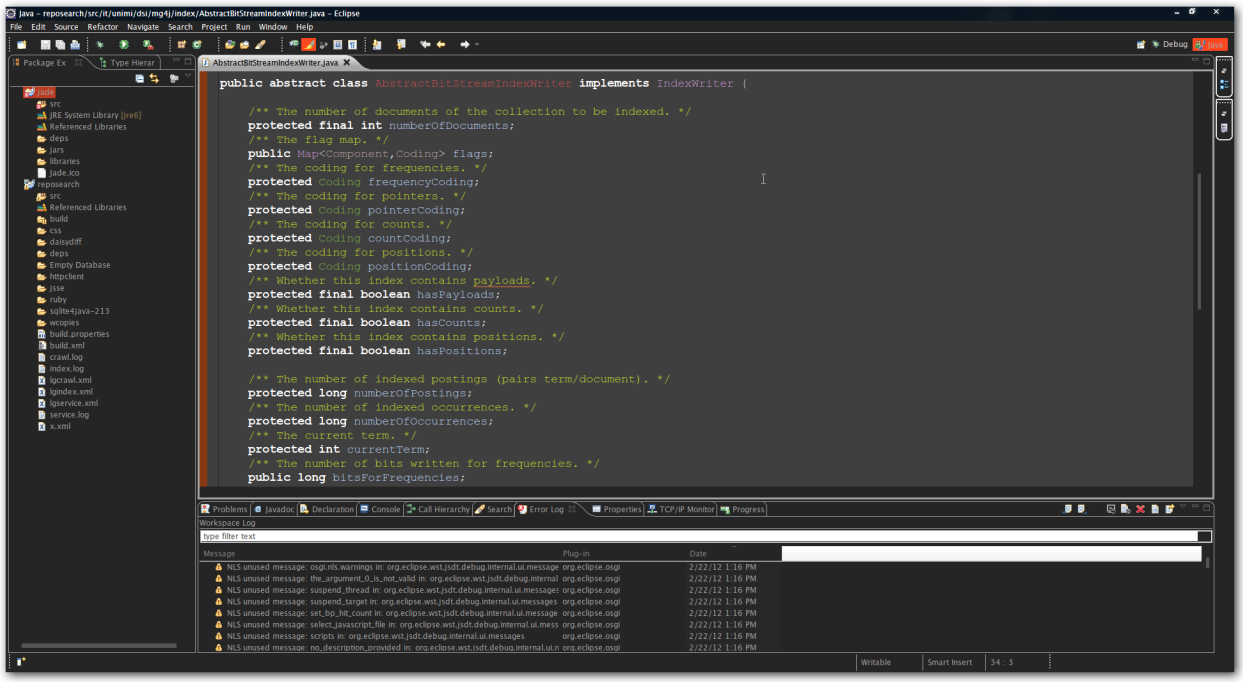


Figure 2.2.2 - *Eclipse development environment*

Eclipse provides a wide range of features, as well as the ability to expand with plugins. This means that you can customize the development environment to suit your needs and project requirements. Although *Eclipse is known for its Java* support , it can also be used to develop in other programming languages such as *C/C++, Python, PHP* , and many others. Over the years, *Eclipse* has developed a large and active community of users and developers, allowing you to find support, advice and solutions to problems. [18]

2.2.3 NetBeans

NetBeans is a development environment developed by *Oracle* and designed for programming in *Java* and other programming languages such as *PHP, JavaScript, HTML* and others. *NetBeans* has an intuitive and simple user interface that helps developers quickly get up and running with *the IDE* . *NetBeans* has a large number of built-in tools, such as a debugger, a version control system, built-in support for *Maven* and *Gradle* , a web interface editor, and much more.

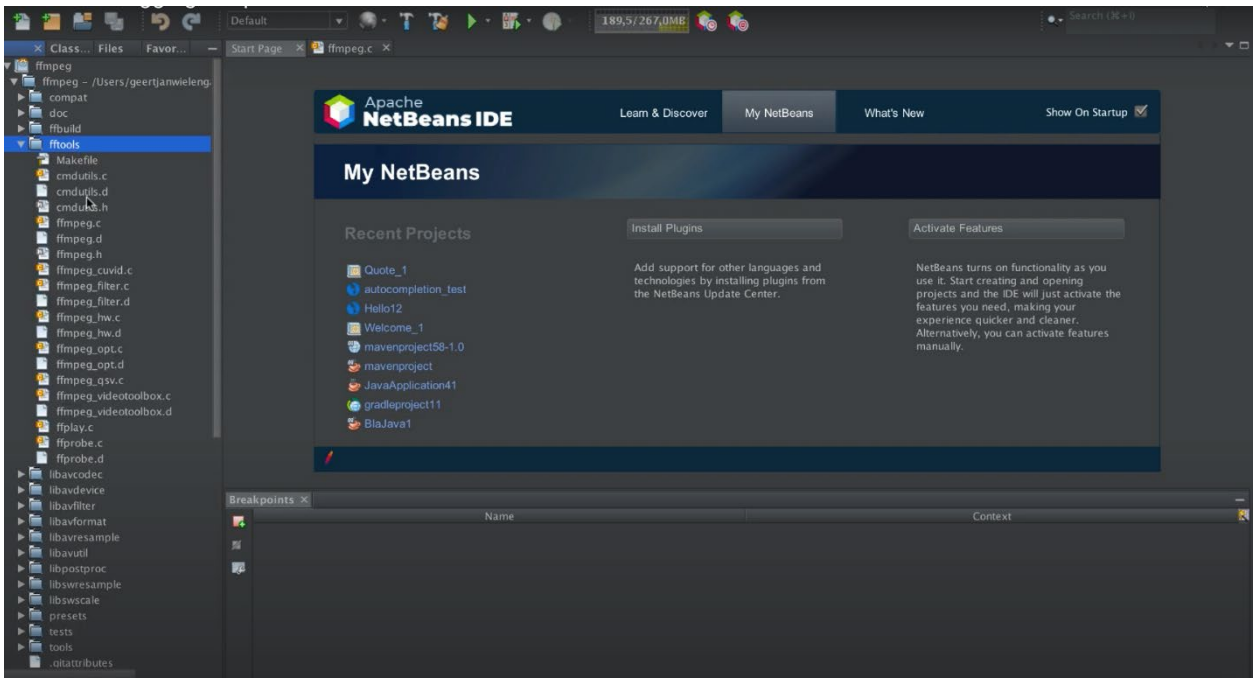


Figure 2.2.3 - *NetBeans development environment*

NetBeans provides tools for developing Java EE -based web applications , including support for application servers such as *GlassFish* and *Apache Tomcat* . With plugins and extensions, *NetBeans* can be extended to support different technologies and tools, making it more flexible for different projects and developer needs. [19]

IntelliJ IDEA was chosen to develop the service . Because, it is an excellent choice for development due to its powerful features that make it easier to write code and reduce the number of errors. Built-in support for version control systems, integration with *Maven* and *Gradle* , and a wide selection of plugins allow you to customize the environment for specific project needs.

2.3 Selection of tools for the implementation of the server part

The Spring Framework is a framework designed for the *Java platform* that provides a variety of capabilities for efficient software development.

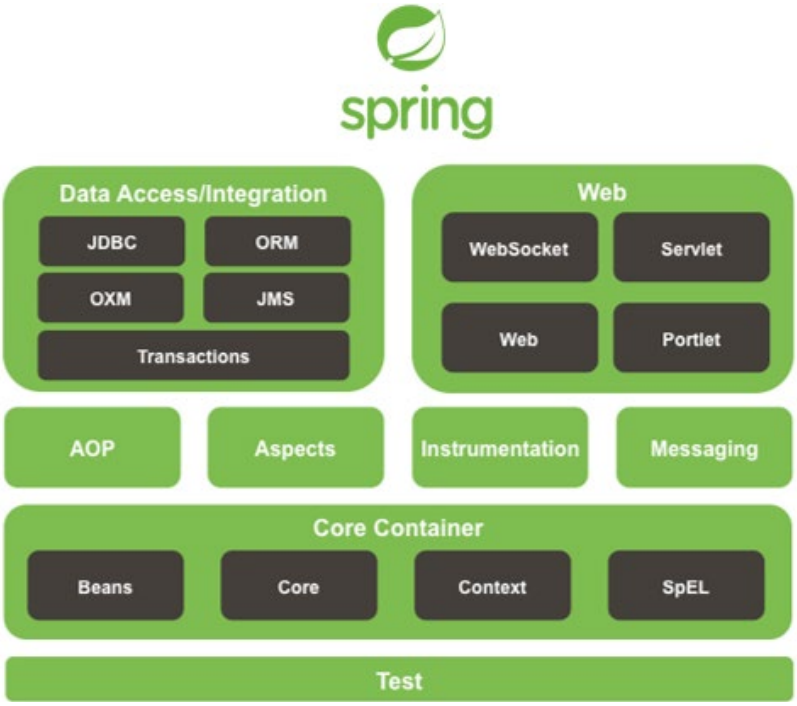


Figure 2.3.1 - *Spring Framework architecture*

Figure 2.3.1 shows the architecture of *Spring modules Framework Runtime* . It consists of several components that perform different functions. The main container includes the *Beans module* . It provides lifecycle management and component configuration. The core contains core functions such as *IoC* . The context allows access to configuration components and their implementation. *SpEL* is an expression language that allows you to manipulate objects in the *Spring context* .

The data access and integration module supports various methods of data access and interaction with other systems. *JDBC* allows you to connect to relational databases through *Java Database Connectivity* . *ORM* integrates with various *ORM* frameworks such as *Hibernate* . *OXM* supports conversion between *Java* and *XML objects* . *JMS* helps to integrate with messaging systems. The *Transactions* module manages transactions to ensure data integrity.

The Web module includes *a WebSocket* for two-way communication between clients and the server. *Servlet* supports traditional *Java* servlets. General web application support includes *Spring MVC* . *Portlet* allows you to develop applications that use portlet containers.

The AOP module provides support for aspect-oriented programming. It allows you to separate cross-functional functions, such as logging and transaction management, from the core application logic. The *Aspects* module includes libraries for creating and using aspects in *Spring* applications. *Instrumentation* supports working with bytecode and optimizes the performance of *Java* applications. *Messaging* provides messaging, including asynchronous methods and integration with various messaging systems. *Test* includes support for testing *Spring* applications, including unit tests, integration tests, and configuration tests. [20]

The Spring Framework Runtime consists of many modules, each of which provides specific functionality for creating complex, scalable, and flexible *Java* applications. Each of the components is closely integrated with the others, which allows you to effectively use the framework's capabilities.

2.3.1 Spring Boot

Spring Boot is an innovative and powerful part of *the Spring Framework* , which

Spring- based applications .

Spring Boot provides standard default configuration options that allow developers to get started with an application with minimal configuration. No need for large *XML* configuration files or complex *Java* configurations reduces deployment time and makes working with the framework easier.

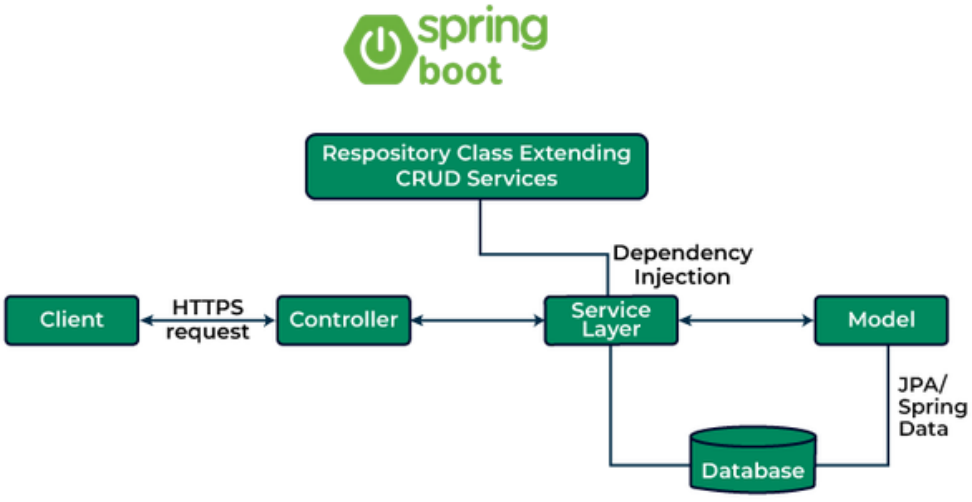


Figure 2.3.2 - *Spring Boot* execution flow architecture

In fig. 2.3.2 shows the architecture of the *Spring execution flow Boot* . It consists of several important components that work together to process requests and manage data. The client sends *HTTP* requests to the server. The controller receives these requests, processes them, and sends responses back to the client. It calls methods of the service layer.

The service layer contains the business logic of the application. It handles requests from the controller and accesses the data layer. Repositories in this layer use *JPA* or *Spring Data* for interaction with the database. They provide *CRUD* operations. The model represents data to the application and passes it between different components. The database stores all application data. *Spring Boot* uses *JPA* or *Spring Data* to access this data.

The work process looks like this. The client sends a request to the controller. The controller passes the request to the service layer. The service layer executes the business logic and accesses the repositories. Repositories work with a database. After performing operations, the result is returned to the controller. The controller sends the response back to the client.

Spring Boot implements a multi-layered architecture in which each layer simultaneously interacts with the other in a hierarchical structure.

In *Spring Boot* has four main layers (Fig. 2.3.3): *Presentation Layer* , *Business Layer* , *Persistence Layer* and *Database* .

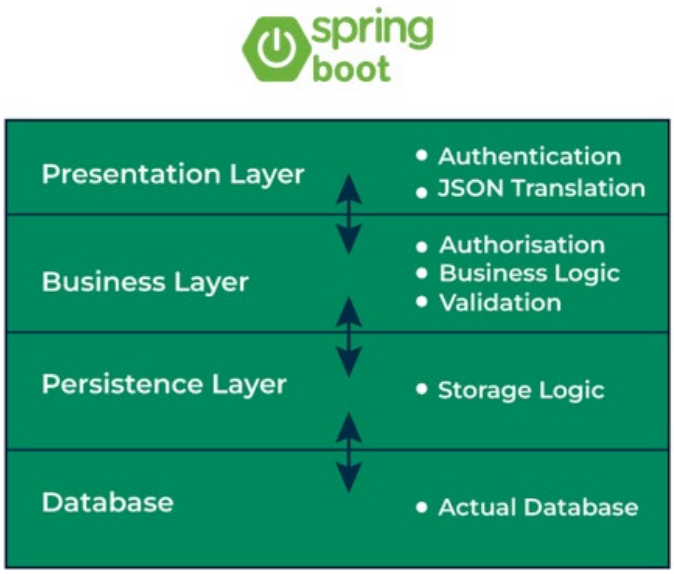


Figure 2.3.3 - Layered architecture *of Spring Boot*

The Presentation Layer handles all *HTTP* requests from clients. It translates *JSON parameters* into objects, authenticates requests, and passes them to the business layer. Basically, this layer consists of the front-end part or, in other words, the presentation.

The Business Layer is also known as the Service Layer *and* corresponds to the entire business logic of the application. It consists of service classes that perform services provided by the data access layer. without it, it performs authorization and validation.

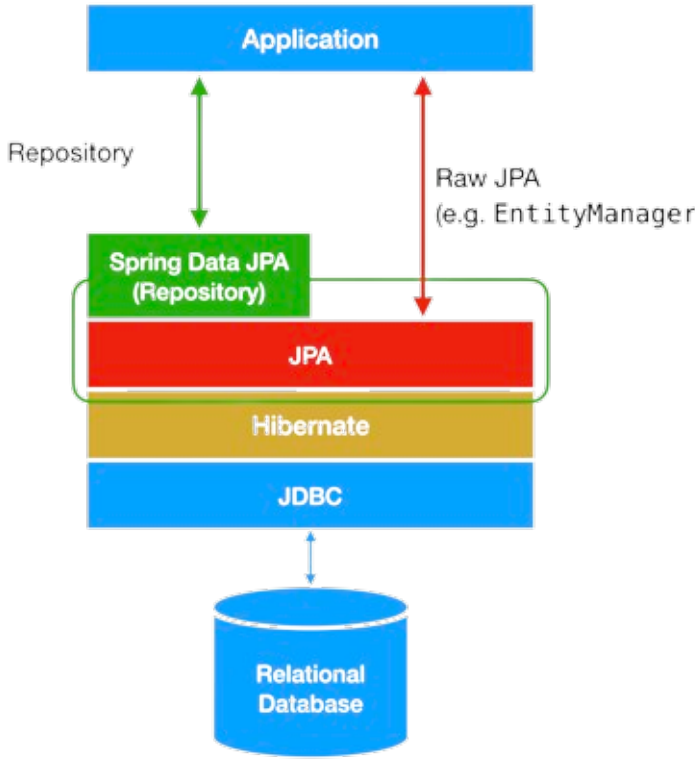
The Persistence Layer includes all the data persistence logic required to convert business objects into database rows. *Database* on this layer provides all *CRUD operations* . [21]

Thus, the multi-layered architecture *of Spring Boot* provides a clear separation of duties between effective system components, contributing to increasing the modularity, comprehensibility and scalability of applications.

2.3.2 Spring Data JPA

Spring Data JPA is part of the larger *Spring Data ecosystem designed to facilitate the implementation of JPA*- based repositories . Its goal is to significantly reduce the amount of boilerplate code needed to implement data access layers, making the development of applications that use databases faster and easier.

Spring Data JPA provides a repository abstraction. It is a set of interfaces that define database operations. Query methods are implemented automatically based on method names. Custom queries are also supported. They can use *JPQL* , *SQL* and native queries.



Spring Data JPA interaction architecture with the database

In fig. 2.3.4 shows the layers and interaction between an application and a relational database using *Spring Data JPA* . The application is the top layer. It represents the business logic and user interface that interacts with the data layer. *Spring Data JPA* , the green layer, provides an abstraction over *JPA* . This layer allows the application to use repository interfaces to perform *CRUD operations* . For this, it is not necessary to write their implementation.

JPA is standard *Java Persistence API* . It defines an interface for managing relational data in *Java applications* . *Hibernate* is an implementation of the *JPA specification* . It is a widely used *ORM* framework. *Hibernate* converts *Java* objects to database tables and vice versa. *JDBC* , which stands for *Java Database Connectivity* , is a standard *API* for connecting and querying databases. A relational database represents a database where data is stored.

2.3.3 Spring Security

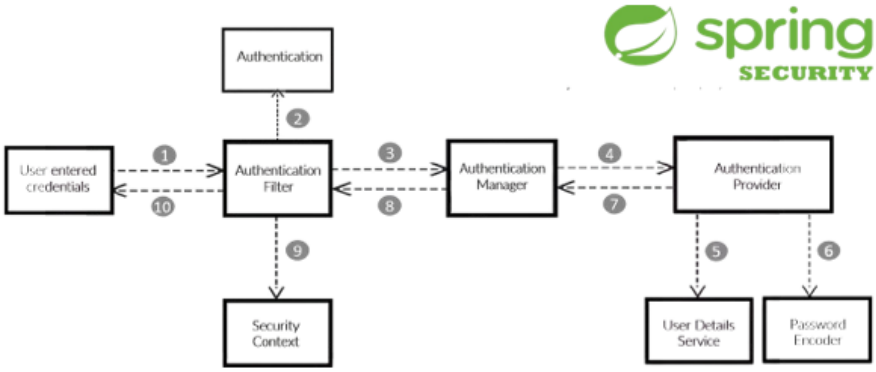
Spring Security is a component *of Spring Framework* that provides comprehensive protection of web applications on the *Java platform* . It offers

flexible capabilities for authentication, authorization, attack protection, and user session management.

The main purpose *of Spring Security* consists of securely authenticating users using various methods such as login forms, basic authentication, and tokens. *Spring Security* also allows you to configure authorization, determining access to resources based on user roles or other conditions. This provides protection against unauthorized actions.

Attack protection is an important aspect *of Spring Security* . It includes mechanisms to protect against attacks such as code injection, session hijacking, and password attacks. User session management includes setting timeouts, session identification, and protection against session attacks, ensuring secure session management.

Integration with other *Spring modules* such as *Spring Data JPA* and *Spring Security* provides a powerful tool for implementing security in web applications and provides a unified mechanism for configuration and security management.



Spring Security execution flow [22]

On rice 2.3.5 presented flow implementation *Spring Security* , that illustrates the main ones classes and their interaction Interaction between classes in *Spring Security* takes place according to the following scenario. In the beginning, there is the *Unprotected* class *Resources* . This is the starting point where the resource access request is not yet secured. Next comes the *Authentication class Filter* . It handles incoming requests and performs initial user authentication. If the authentication is successful, the request is forwarded.

Authentication Manager manages authentication requests. It verifies them using different authentication providers. Upon successful authentication, *Authentication is created Result* . This class contains the authentication results, including information about the user and their access rights.

Security Context stores authentication and authorization data. This is necessary for further use within the current session. *Password Encoder* is used to encode and verify passwords. This provides an additional level of security when storing and verifying user passwords.

User Details The Service provides information about the user that is used for authentication. It interacts with data sources to get details about users. [23]

In general, *Spring Security* is a powerful and flexible tool. It provides security for *Spring*- based web applications *Framework* . Provides a variety of capabilities for authentication, authorization, attack protection, and user session management.

2.4 Selection of tools for saving data

Data storage is an important part of any application, and there are various tools and technologies for working with databases.

2.4.1 Hibernate

Hibernate is a powerful *ORM framework* for the *Java programming language* that allows you to work with *Java objects* and represent the database as objects. The main features *of Hibernate* :

- Automatic mapping of objects to database tables: allows you to define the structure of object modeling using *Java classes* , ensuring that these classes correspond to tables in the database.
- Mechanisms of relationships between objects: supports different types of relationships (one-to-one, one-to-many, many-to-one) using annotations or *XML* configuration.
- *HQL* Query and Language Support : Provides an object-oriented *HQL* query language for executing complex database queries using the application object model.

- Management of the state of objects and transactions: allows you to manage the state of objects in the context of sessions and to make changes in objects and then save them in the database using transactions.
- Caching support: provides the ability to cache objects and queries, which increases the performance of the application, reducing the frequency of interaction with the database.

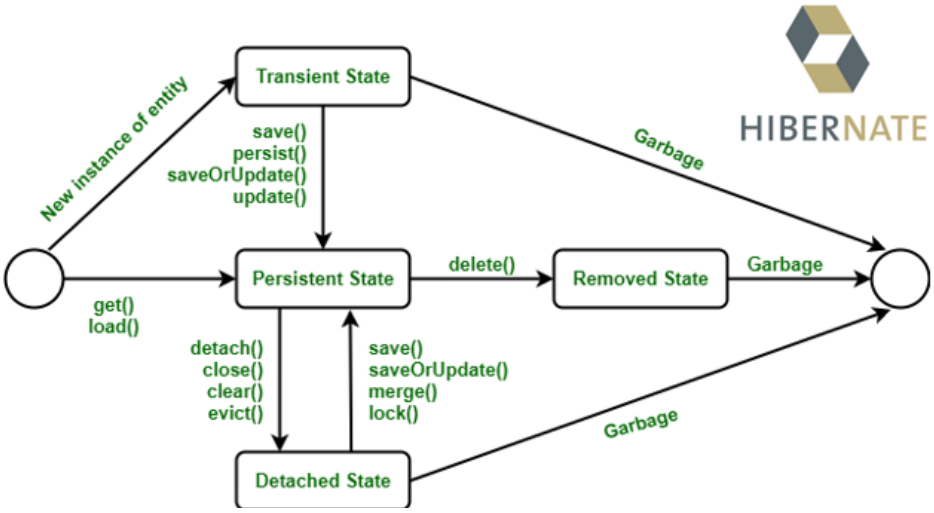


Figure 2.4.1 – Life cycle of an object in *Hibernate*

On fig . 2.4.1 shows the lifecycle of an object in *Hibernate* , which has four main states. The first state is *Transient State* . In this state, a new object is created, but not associated with a *Hibernate session* and not stored in the database. The object exists only in RAM. Next comes *Persistent State* . In this state, the object is stored in the database and associated with the *Hibernate session* . Changes to the object are automatically reflected in the database.

There are several transition methods such as *save (), persist (), saveOrUpdate (), update (), merge () and lock ()* . Next, *Detached is defined State* . In this state, the object is disconnected from the *Hibernate session* , but remains stored in the database. Changes to the object are not reflected in the database until it is reconnected. Transition methods include *detach (), evict (), clear (), and close ().*

The last state *is Removed State* . In this state, the object is deleted from the database. The transition method *is delete()* .

2.4.2 PostgreSQL

PostgreSQL is one of the most powerful and extensible object-relational database management systems (*ORDBMS*) on the market today. It has built-in data integrity controls, disaster recovery, and advanced database monitoring and management tools.

The main advantages *of PostgreSQL :*

- Scalability : Supports horizontal and vertical scaling to meet growing application needs.
- Modern features : support for complex data types (*JSON* , *XML* , geometric types) makes *PostgreSQL* ideal for a variety of applications.
- Transactions : adheres to *ACID* properties , ensuring the integrity and reliability of data operations.
- Replication : The ability to implement replication allows you to synchronize data between distributed database systems for high availability and load handling.

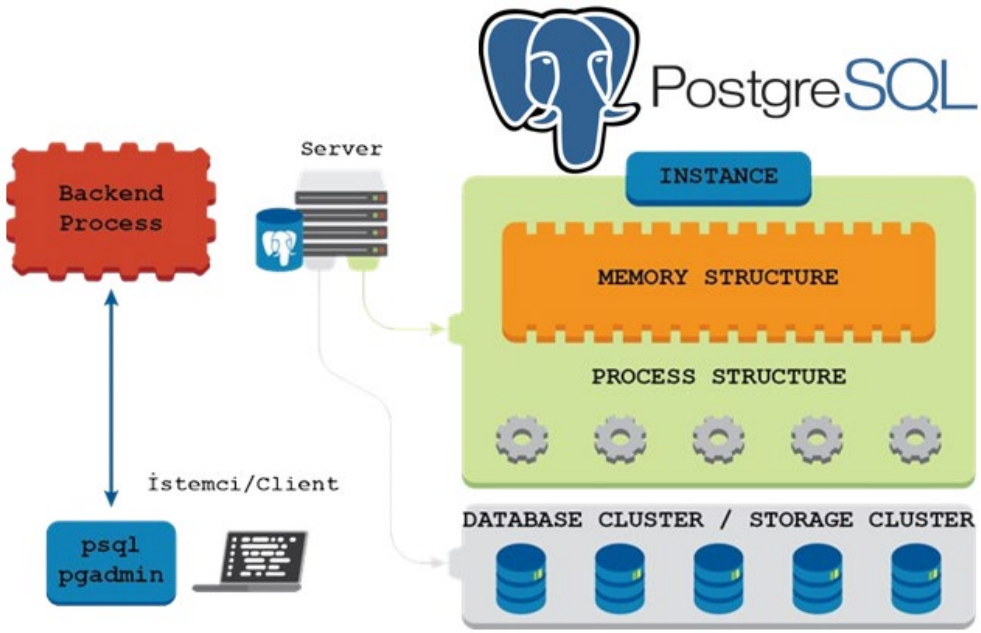


Figure 2.4. 2 – Architecture *of PostgreSQL*

PostgreSQL architecture has three main components. The first component is memory, which is known as *Memory Structure* . It is responsible for memory

management in the system. This part includes various data structures that are stored in RAM. This helps to ensure efficient database operation. The main components of memory include caches, buffers and other temporary structures. They facilitate fast data access and processing.

The second component is processes, known as *Process Structure* . Processes are responsible for executing queries and managing the database. There are different types of processes in *PostgreSQL , including the Backend Process* and daemon process.

The third component is the storage cluster, or *Storage Cluster* . This part includes the physical files on disk where the database data is stored. *PostgreSQL* organizes data into database clusters. They can contain multiple databases. Each database consists of a set of tables, indexes, and other objects stored on disk.

This architecture allows *PostgreSQL* to efficiently manage data, providing high performance, reliability, and scalability. *PostgreSQL* is known for its high reliability and stability. It has built-in data integrity controls, failure recovery mechanisms, and advanced database monitoring and management tools.

In conclusion, *PostgreSQL* is a high-performance, reliable and extensible database that provides a number of modern features and mechanisms. [24]

2.5 Selection of the project assembly tool

Effective dependency and project management is extremely important in modern software development. With the increase in the complexity of software systems and requirements for the quality and speed of development, there is a need for tools that automate routine tasks and ensure the convenience of working with large projects. Tools like *Maven* make it easy to manage dependencies, standardize project structure, and automate build and test processes, which increases productivity and ensures the reliability of the final product.

2.5.1 Maven

Maven is a powerful project and build management tool used in Java software development . It was designed to simplify the processes involved in creating,

compiling, testing and deploying software. One of the main reasons for *Maven's popularity* is its ability to automate dependency management and provide a standardized project structure. In the context of *Spring* -based service development , *Maven* plays a critical role, providing organization and coordination of many aspects of development.

Maven's core features include several key aspects. *Maven* uses the *pom.xml configuration file* to define all project dependencies. This file allows you to automatically download required libraries from the central *Maven repository* or other specified repositories. For example, for a *Spring*- based project, you can add dependencies to modules as *Spring Core* , *Spring Boot* , *Spring Data* needed to build the service.

Maven offers a standardized directory structure for projects that simplifies code navigation and organization. A typical *Maven* project structure includes directories such as *src/main/java* for source code and *src/main/resources* for resources, *src/test/java* for test classes. In fig. 2.5.1 shows the automatically generated project structure using *Maven* .

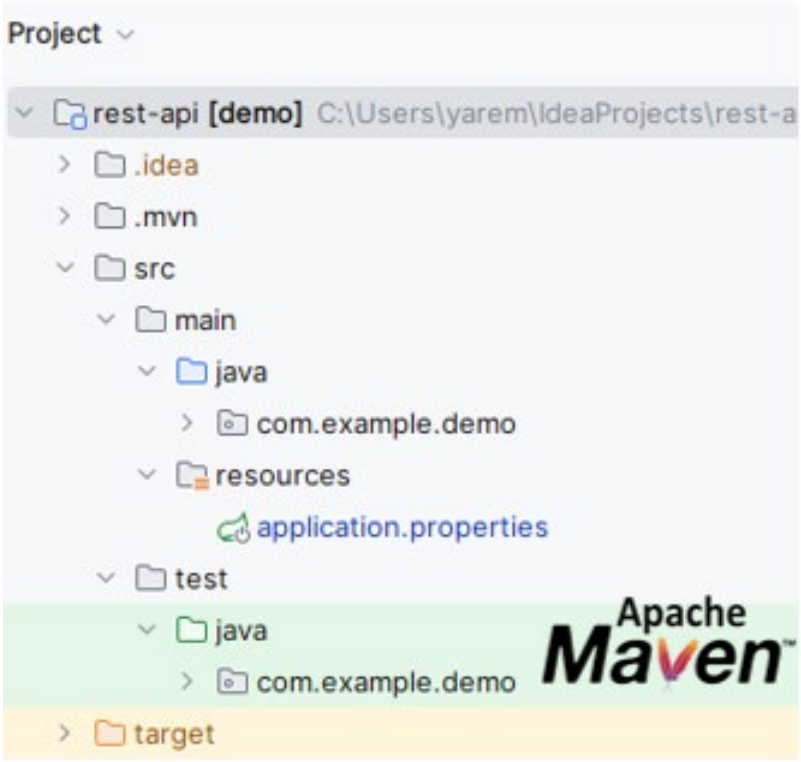


Figure 2.5.1 – Automatically generated project structure

Maven defines a standard project life cycle. This cycle includes steps like compilation, testing, packaging, validation, installation and deployment. *Maven* uses plugins to perform these tasks . These plugins can be configured and extended according to the needs of the project.

Maven also automates the project build process. Build automation includes compiling source code, testing, creating executable files as *JAR* or *WAR* , and deployment. This allows you to reduce the human factor and minimize errors associated with manual actions.

Spring -based services , *Maven* significantly simplifies the processes of managing dependencies and configuring projects. You can easily add the necessary dependencies to *the pom . xml* and be sure that all necessary libraries will be automatically loaded and included in the project. This will allow *Maven* to automatically load all necessary libraries for working with services, *JPA* for database access, and testing tools.

Therefore, *Maven is an indispensable tool in modern Java* software development , especially in the context of *Spring* services development. It provides effective dependency management, project structure standardization, build automation, and integration of various project lifecycle stages. This allows you to focus on writing code, reducing the burden of configuration and project management, and provides a reliable and consistent platform for building high-quality software.

3 DESIGN AND DEVELOPMENT OF TRACKING AND IDENTIFICATION SERVICE

This section describes in detail the structure of the service, which is a fundamental part of the service. It consists of various components, each of which plays a key role in ensuring the functionality of the system.

3.1 Service structure

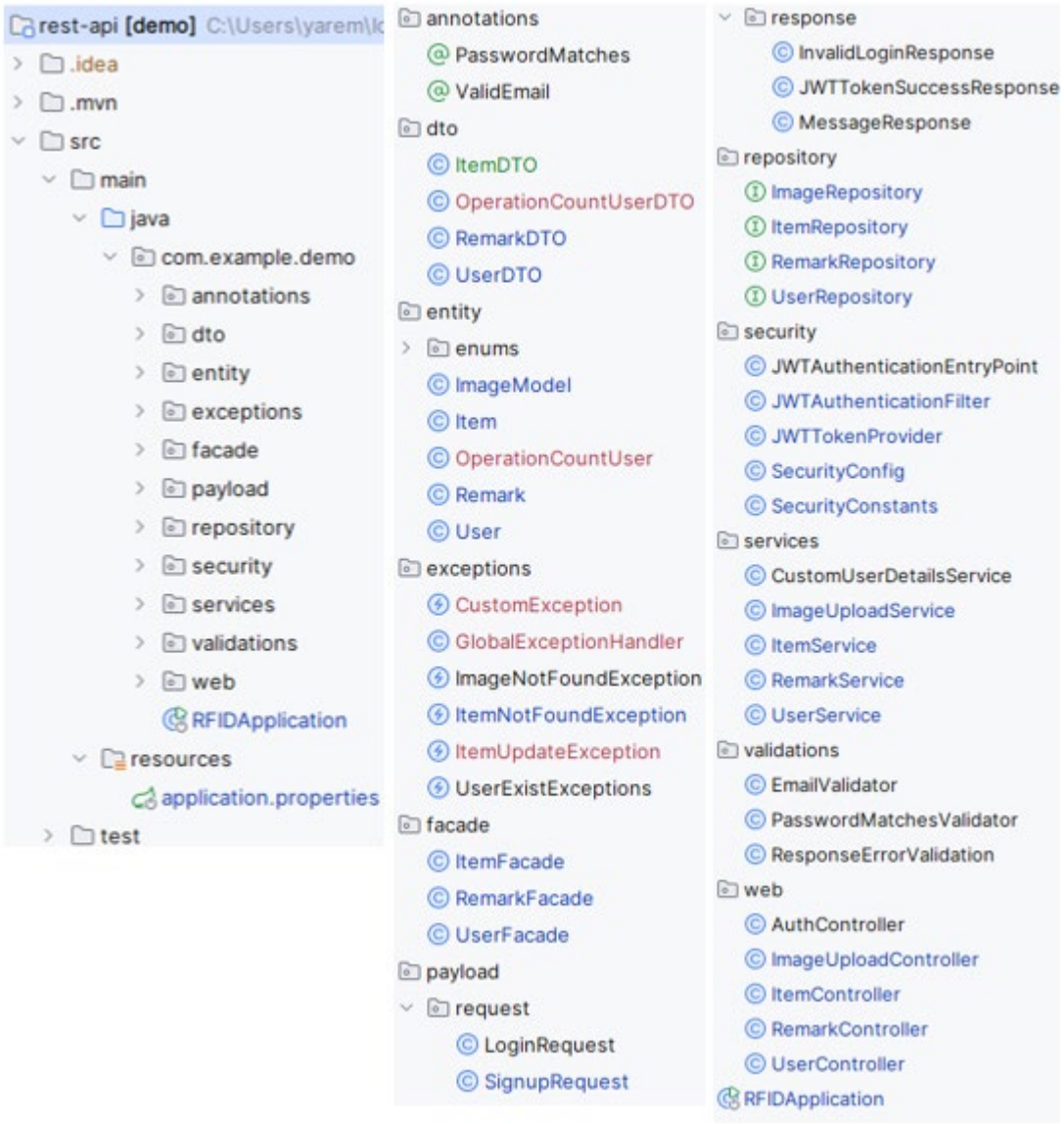


Figure 3.1.1 – Structure of project packages and classes

The service consists of several main components:

- *DTO* - objects for data transfer between different service layers.
- *Entity* - objects representing data in the database.

- *Repository* - interfaces for accessing the database.
- *Service* - business logic of the application.
- *Facade* - a facade pattern to simplify interaction with other components
- *Security* - application security. Includes classes that provide configuration and work with application security, in particular for working with *JWT* tokens.
- *Web* - *REST* controllers for handling *HTTP* requests.
- *Exception Handling* - handling of exceptions that provide handling of various types of errors when executing program logic.
- *Validation* - data validation, checking for correctness.

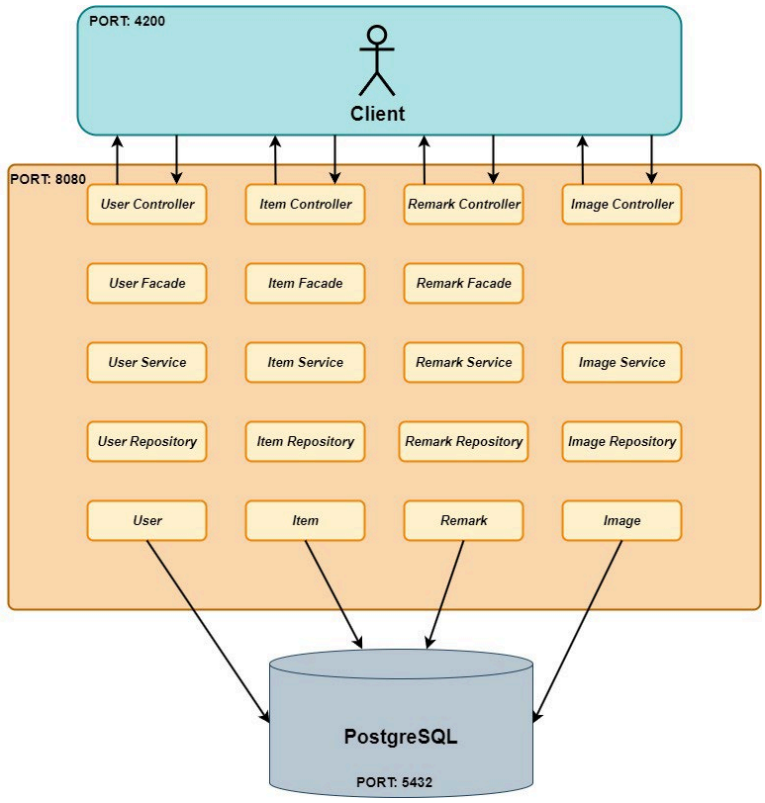


Figure 3.1.2 – *UML* diagram of class interaction architecture

3.2 Implementation of Spring Security and JSON Web Token

Securing web applications is one of the key aspects of software development. One effective approach is to use *JWT* to authenticate and authorize users. Combined with *Spring Security* , it allows you to create reliable and secure services.

JWTAuthenticationEntryPoint class is used to handle failed authentication attempts. It implements the *AuthenticationEntryPoint interface* and is called when a user tries to access a protected resource without a valid token. If authentication fails, this class generates a JSON error response *indicating* that access is denied.

The JWTAuthenticationFilter filter intercepts all incoming *HTTP* requests and checks whether they contain a valid *JWT* token. If the token is valid, the filter retrieves the user's details from the database and configures the security context for the current request. This ensures that every request that contains a valid token will be associated with the appropriate user.

JWTTokenProvider class is responsible for creating and validating *JWT* tokens. It generates a token using a secret key for signing and includes the user's information in the token. In addition, the class validates the token by checking its integrity and expiration, and extracts user information from the token when necessary.

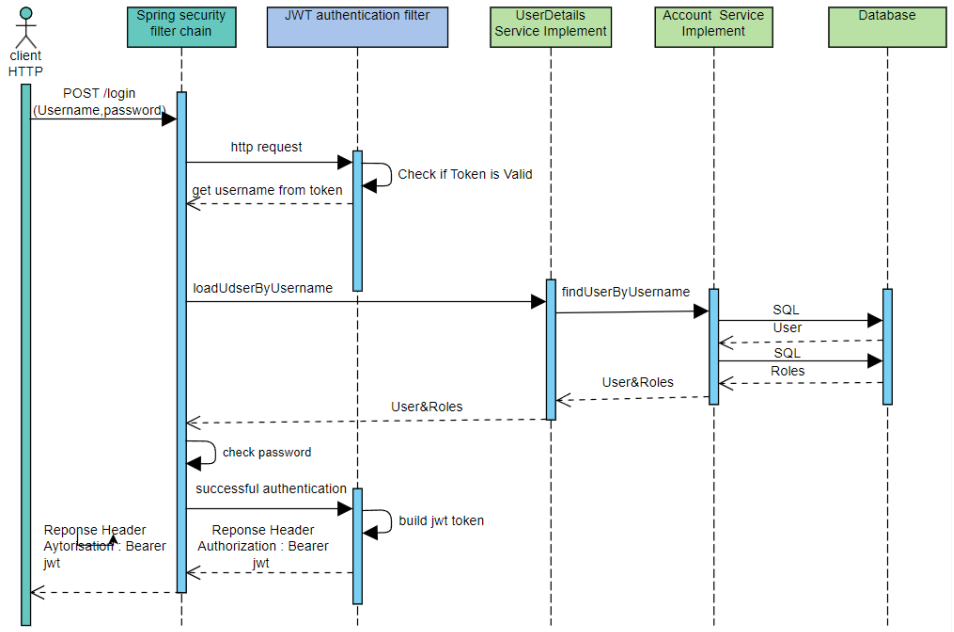


Figure 3.2.1 – Sequence diagram for *JSON Web Token* authentication [25]

Fig . 3.2.1 presents a sequence diagram for the *JWT* authentication process .

HTTP request processing process begins when the client sends a request to *the endpoint /login* . He provides his credentials such as username and password. Next, the request goes through *the Spring Security filter chain* . This is where user

credentials are verified. If they are true, *Spring Security* creates an authentication object.

After successful authentication, this object is passed to *the JWT Authentication Filter* . This filter checks the presence and validity of *the JWT* token in the request. If the token is valid, the filter extracts the username from the token.

UserDetailsService class loads user details by name. This includes user roles and privileges. This data is returned via *Spring Security* . The user's password is verified. If the validation is successful, a new *JWT* token is generated. This token contains information about the user. The new *JWT* token is passed to the client in the response header. The client will use this token for authentication on subsequent requests.

Using classes for security with *Spring Security* and *JWT* tokens are an important aspect of securing web applications. These classes provide user authentication, resource protection, and a convenient token-based access control mechanism that makes the system more secure and efficient.

3.3 DB tables and repositories

Modern software development often uses database management systems to store and process large amounts of data. Using *JPA* simplifies this process by allowing developers to automatically generate database schemas based on *Java classes* .

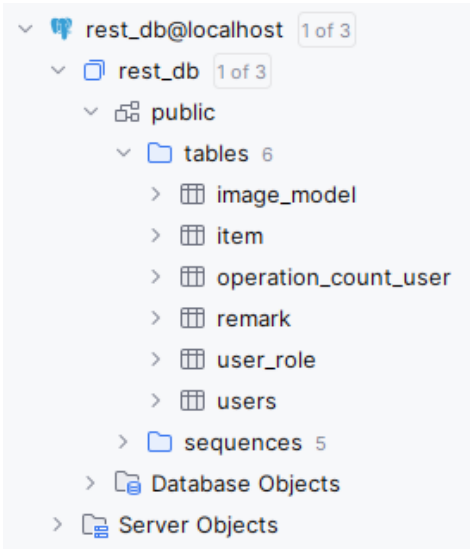


Figure 3.3.1 – Created tables in the database

In this paper, the structure of the database generated using *JPA is displayed* and the main tables are characterized.

The users table stores information about system users. It includes fields *like id, firstname, lastname, username, email, phoneNumber, position, rfReaderIdToken, responsibility, characteristic, password* and *createdDate* . This table helps identify and authenticate users and define their roles in the system.

item table contains data about items used in the system. It includes fields like *id, rfTag, name, type, description, status, createdDate and updatedDate* . This table allows you to store information about each item, its properties and state.

Operations are tracked using *operation_count_user* , which users perform on items. It contains fields *id, user_id, item_id, updatedDate* and *status* . This table is used to store the history of operations and interactions between users and items.

remark table is used to store user comments and remarks about items . It includes fields like *id, item_id, username, userId, message* and *createdDate* . This table helps to store and organize user communications about specific subjects.

image_model table stores images related to objects or users. It contains fields like *id, name, imageBytes, userId, itemId* . This table is used to store media files associated with other system objects.

User roles in the system are stored in the *user_role table* . It contains *user_id* and *role fields* . This table provides flexible access control based on user roles.

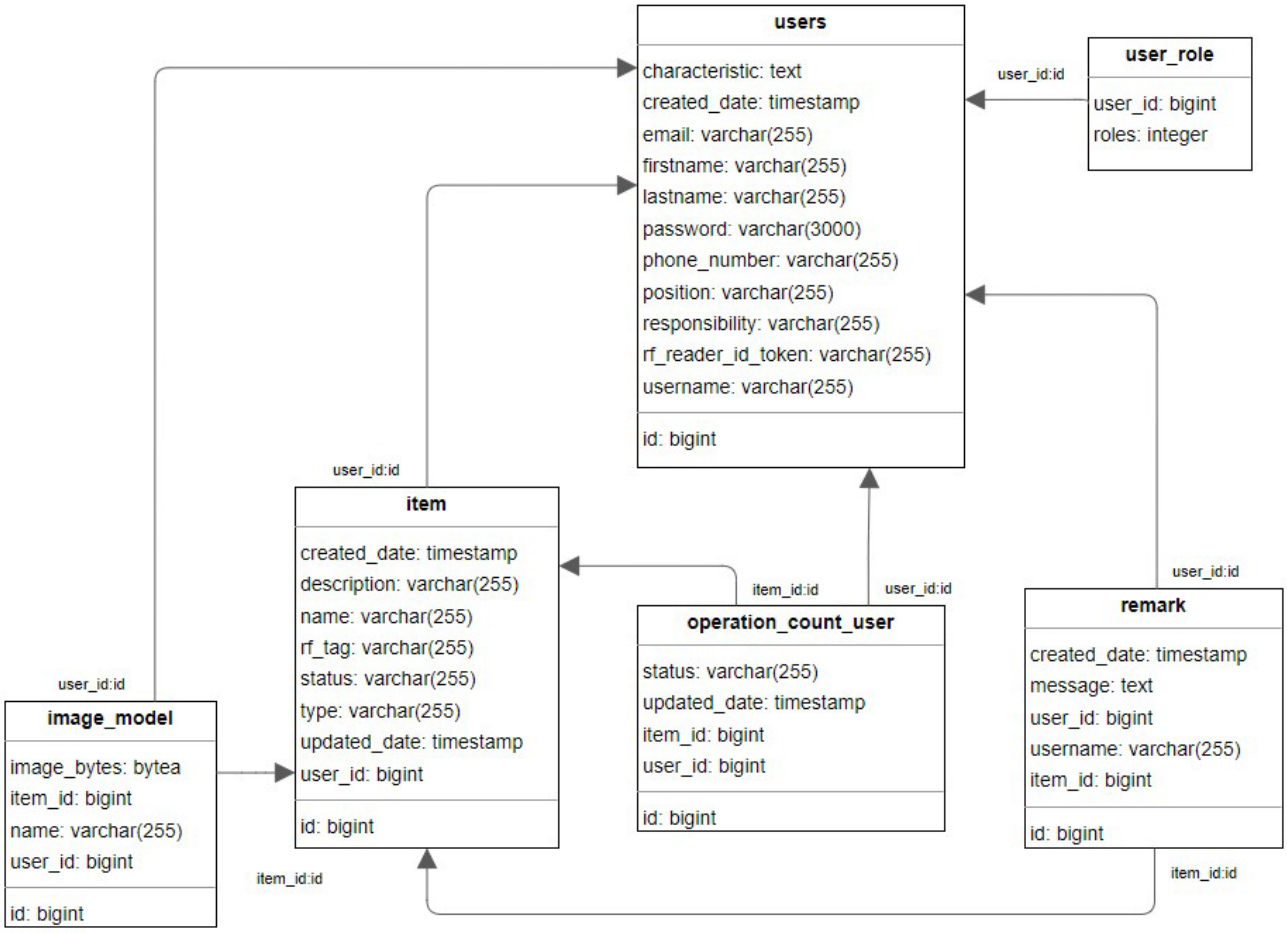


Figure 3.3.2 – Connections between tables

Applications using *JPA* use repositories to interact with the database. They provide convenient and efficient access to data, allowing to perform *CRUD* operations.

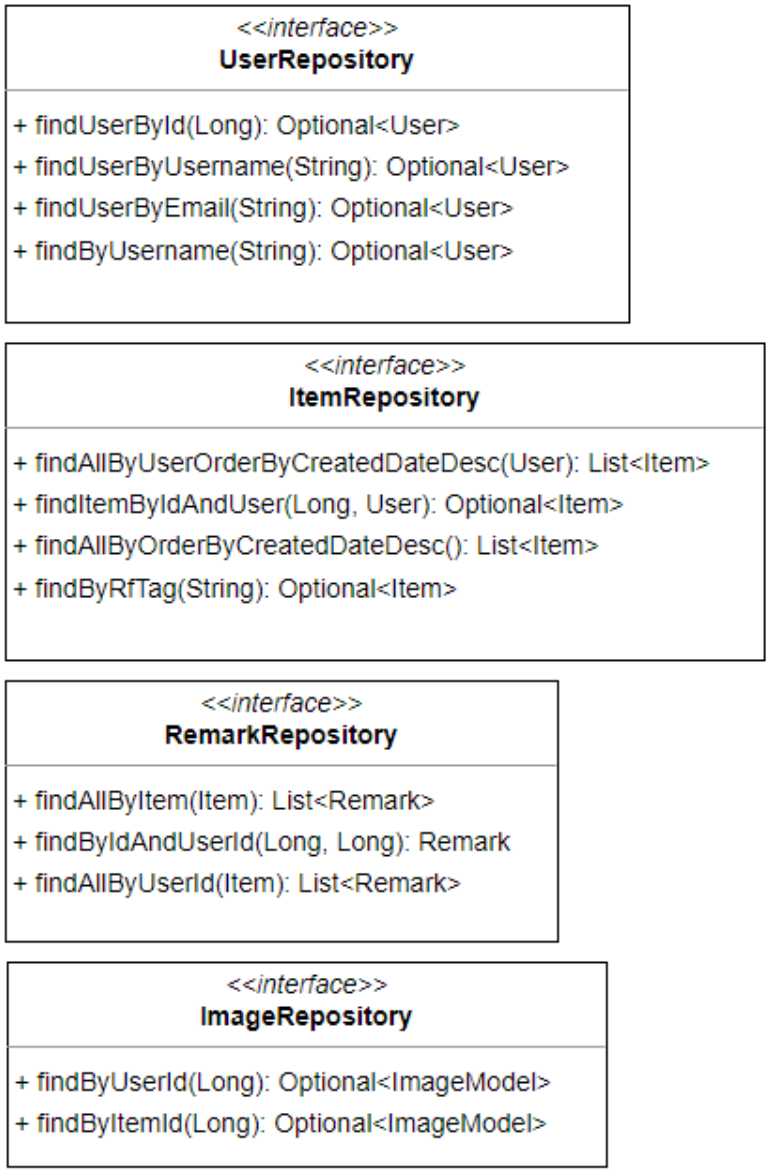


Figure 3.3.3 – Repositories for working with tables

UserRepository is a repository for working with the *users table* . It provides methods for searching users by criteria such as *id, username, email* . *ItemRepository* acts as a repository for the *item table* . This repository allows you to find items by *id, rfTag* . You can also get all of a user's items sorted by creation date.

RemarkRepository works with the *remark table* . It provides methods for searching comments by subject and user ID. It also allows you to receive all comments related to a specific subject. ImageRepository is used to work with the *image_model table* . This repository allows you to find images by user and item IDs.

Automatic generation of database tables using *JPA* makes the development of complex information systems convenient and efficient. The use of repositories

facilitates easy interaction with the database. This allows you to perform the necessary operations with minimal effort. This approach improves performance and ensures reliability and consistency of data in the system.

3.4 Classes of services

Services that provide business logic and interaction with the database play an important role in the development of a modern application. Services are responsible for performing various operations, processing data and ensuring the functionality of the application. Below is a description of the main services used in this system, as well as their functions and corresponding classes.

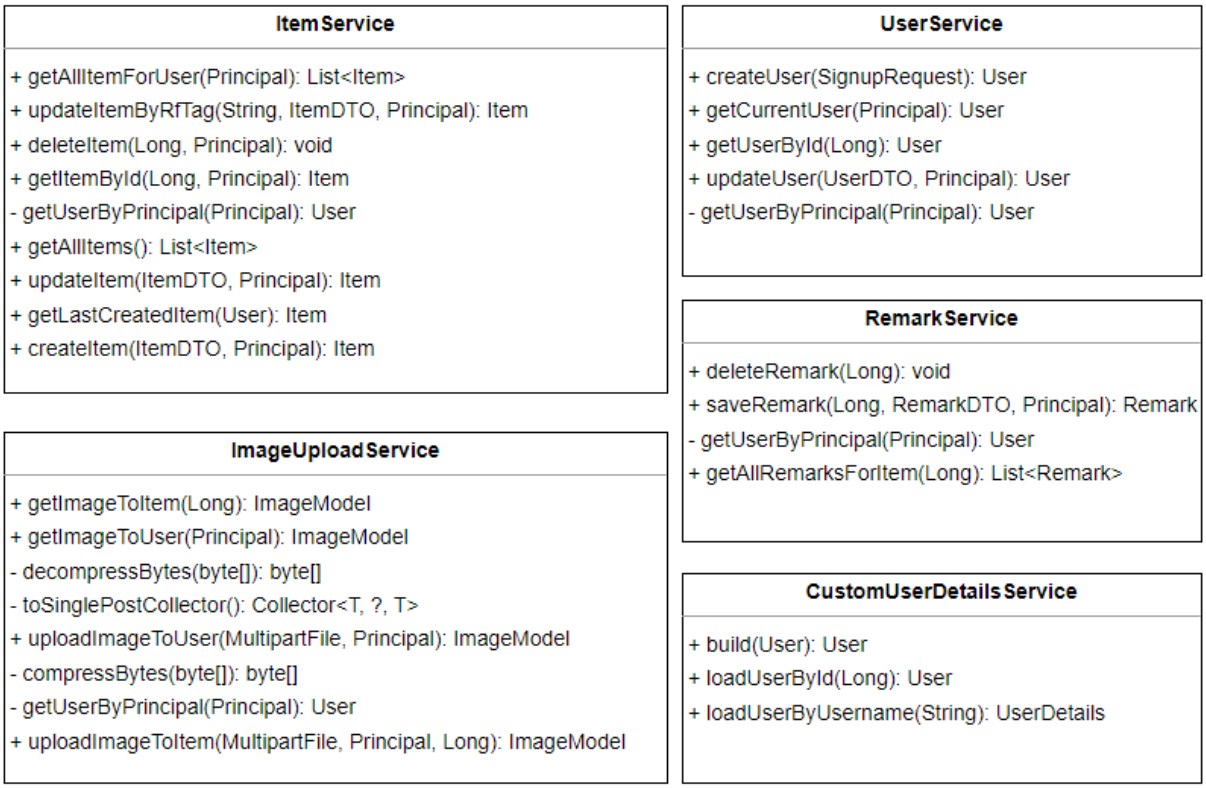


Figure 3.4.1 – Service classes and their properties

ItemService deals with the management of items in the system. Its main tasks include creating, updating, deleting, and retrieving items. *UserService* is responsible for user management. This service provides opportunities to create, update and receive information about users.

RemarkService manages item comments. It allows you to create, retrieve and delete comments.

ImageUploadService is responsible for uploading and saving images in the system. This service allows users to upload and receive images for profiles and items.

CustomUserDetailsService is responsible for loading user details for the security system. It implements interfaces that allow you to retrieve information about users by their IDs and names.

The services in this system provide key business operations such as managing items, users, comments, and images. They are responsible for data processing and interaction with the database, ensuring the proper functioning of the application. The use of well-structured services allows increasing the efficiency of software development and support, ensuring system flexibility and reliability.

3.5 REST controllers

Controllers in web applications play a key role in providing interaction between users and the server. They process *HTTP* requests, interact with services and generate responses for clients. The main controllers used in the system, their functions and corresponding classes are described below.

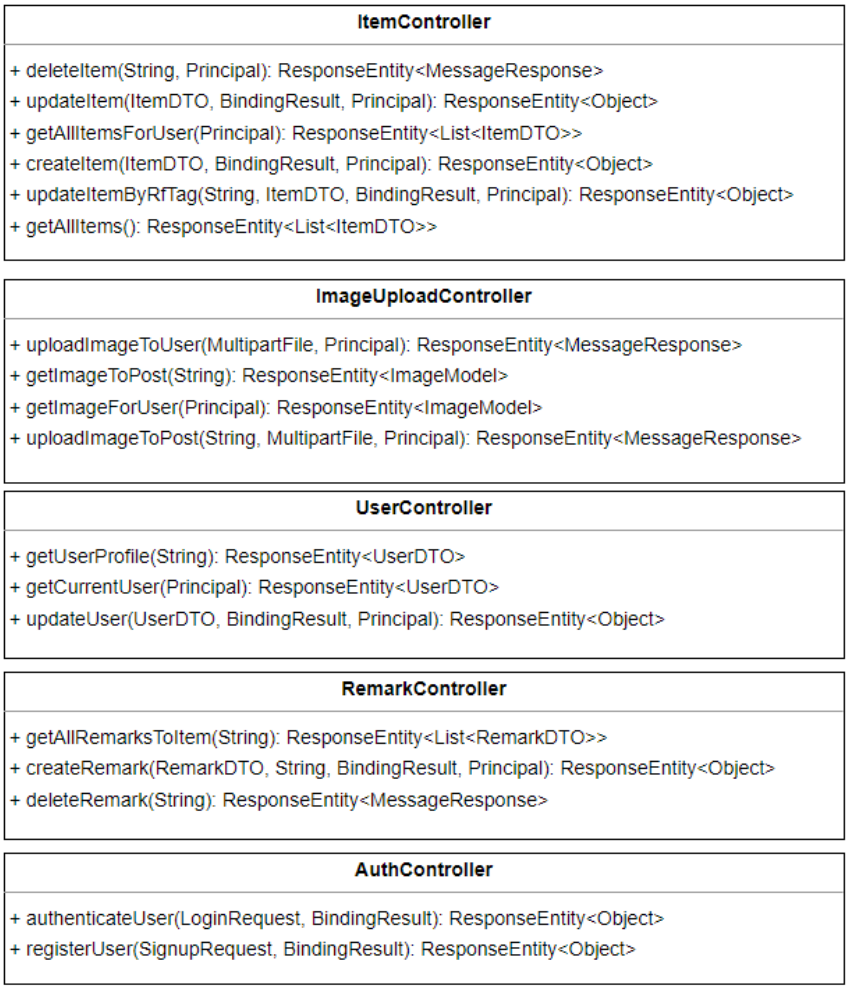


Figure 3.5.1 – Controller classes and their properties

AuthController manages user authentication and registration. It handles login and registration requests, cooperates with services to create users and generate *JWT* tokens for authentication. *ImageUploadController* is responsible for uploading and retrieving images. This controller allows users to upload and retrieve images for profiles and items.

ItemController manages items in the system. It handles requests to create, update, delete, and retrieve items. *The RemarkController* manages item comments. It allows you to create, retrieve and delete comments. *UserController* is responsible for managing user profiles. It handles requests to retrieve and update user data.

So, in this section, a service for tracking and identifying objects was designed and developed, which includes an analysis of the structure of the service, its components, and the interaction between them. The main components of the system, such as *DTO* , *Entity* , *Repository* , *Service* and others, form a strongly connected

architecture that ensures security, stability and efficiency of data processing. Using *Spring Security* and *JWT* tokens increase the level of security by authenticating and authorizing users. Database tables, formed using *JPA* , allow you to effectively manage a large amount of data, which contributes to the flexibility and scalability of the system. Services that process the business logic of the system guarantee the correctness and relevance of data through various operations with objects and users. *REST* controllers provide a convenient interface for interacting with clients, maintaining system stability and reliability through web requests.

4 MICROCONTROLLER BASED RADIO FREQUENCY TAG READER

For the correct operation of the service, it is necessary to test it. For this, you need to use an *RFID* reader to identify objects. This section reviews the main components of the prototype, their interactions, and the software that controls the processing and data transfer processes. The development process includes *NodeMCU v3 ESP8266* using the popular *Arduino IDE environment* , which provides flexibility and the ability to quickly start the project.

4.1 NodeMCU v3

NodeMCU v3 The ESP8266 is a development board that allows you to conveniently control various circuits remotely over a local network or the Internet using *Wi-Fi* . The board has 4 MB of *Flash* memory and supports 21 input-output ports. It is equipped with *a UART-USB interface* with a micro *USB connector* . For power supply, the board supports a voltage from 5 to 12V, it can be powered both from *Micro USB* , and from the *Vin contact* (from *5V*).

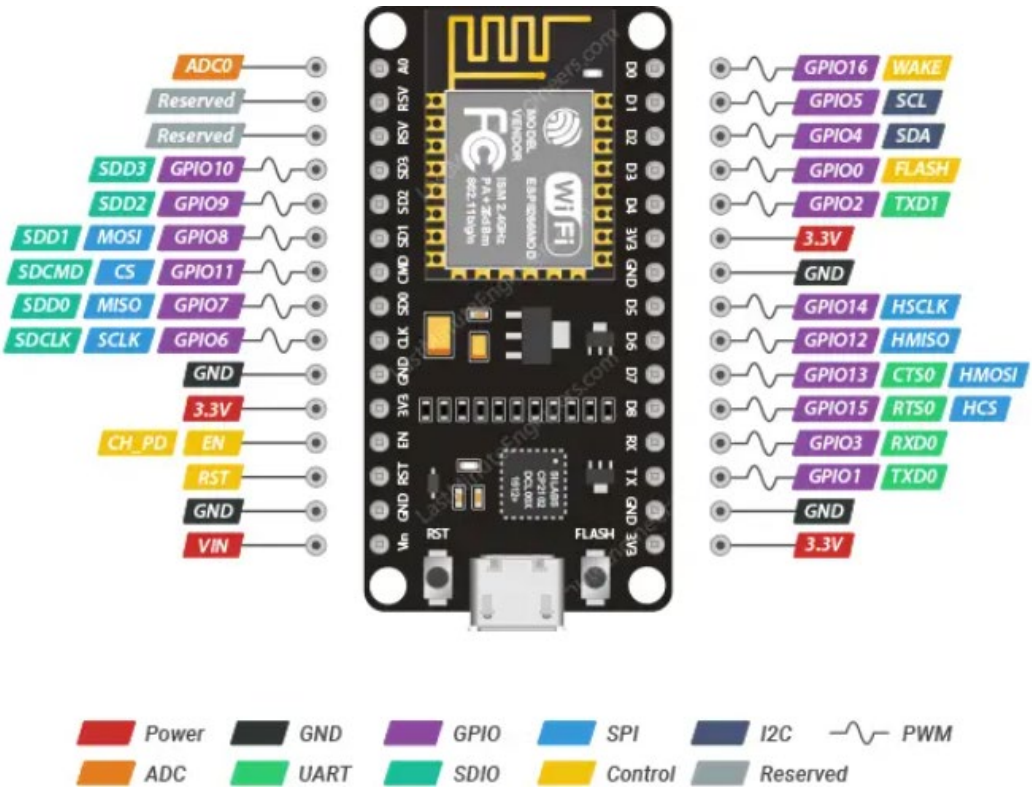


Figure 4.1.1 – Location of *ESP 8266 outputs*

The ESP8266 is a low-cost *Wi-Fi* microcontroller developed by *Espressif Systems* . It has an integrated *TCP/IP* protocol stack that allows any microcontroller to join a *Wi-Fi* network. Key features of *the ESP8266* include:

- Processor : 32-bit *Tensilica L106 RISC* with an operating frequency of up to 160 MHz.
- Memory : 32 kB instruction cache and 80 kB RAM.
- *Wi-Fi* support : *802.11 b/g/n standards* , with support for *STA/AP/STA+AP modes* .
- *GPIO* : up to 17 universal I/O ports supporting *SPI, I2C, UART, PWM, and ADC* .
- Power consumption : Low power consumption, making it suitable for *IoT* devices. [26]

4.2 Arduino IDE

NodeMCU v 3 can be flashed using *Arduino IDE* in the *C language* , which provides considerable flexibility in creating your own software solutions. *Arduino The IDE* is a powerful tool that allows you to write, compile and load code directly onto the *NodeMCU board v3* . This process provides a high level of integration between hardware and software, which is extremely important for developers of *IoT* solutions.

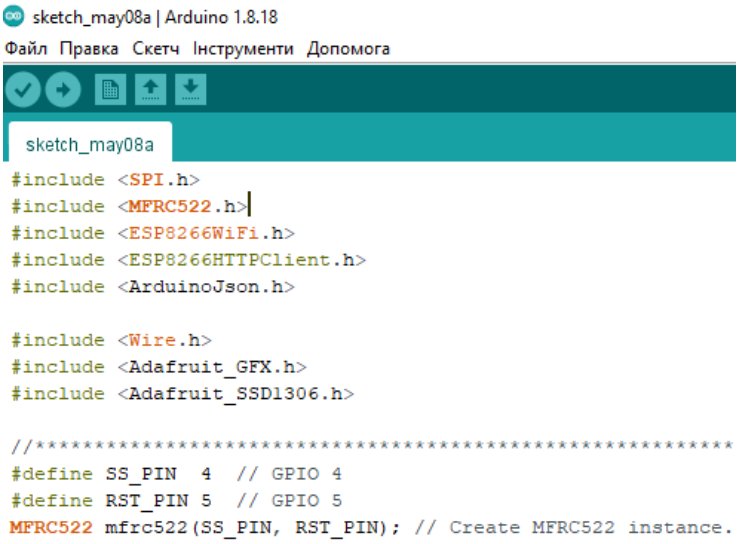


Figure 4. 2 .1 - *Arduino* development environment *IDE*

Arduino The IDE supports many libraries, which greatly simplifies the work with various modules and sensors. For example, using the *ESP 8266 WiFi library* allows you to easily configure a *Wi - Fi* connection, and the library for working with the *MQTT protocol* provides integration with popular IoT platforms. In addition, there is a large number of code examples that you can use as a basis for your own projects or as a source of inspiration for developing new features.

Development process using *Arduino The IDE* is intuitive and user-friendly. *The IDE* graphical interface allows you to quickly go from writing code to testing and debugging it

Arduino IDE also supports extensions that allow you to integrate additional tools and functionality, including for working with other programming languages or for debugging hardware components. This makes the platform even more versatile and allows you to use it for a wide range of tasks, from simple projects to complex commercial solutions.

4.3 Prototype reading device

NodeMCU v3 and the *RFID-RC522 module* are used for the prototype of the device for reading radio frequency tags .

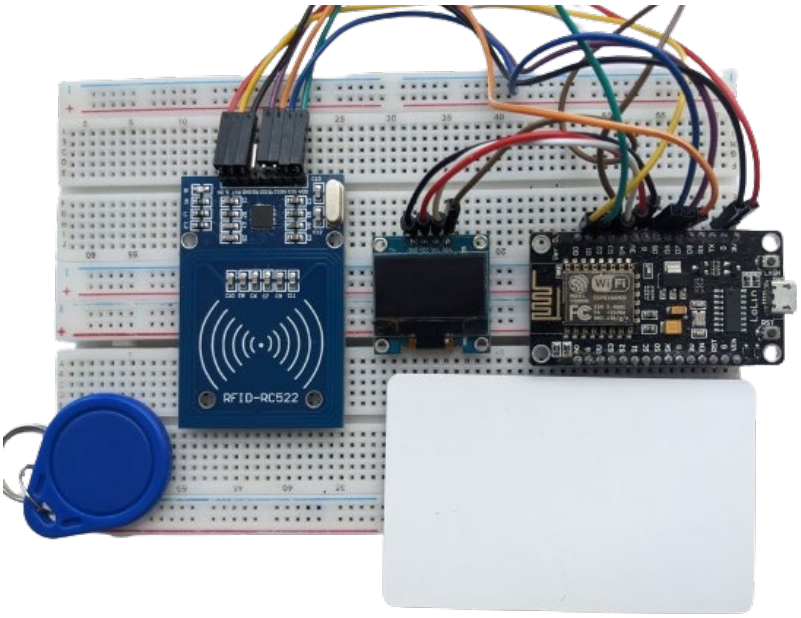


Figure 4. 3 .1 – Assembled reader prototype on a breadboard

The main components of the prototype include several important elements. *The NodeMCU V3 ESP8266* board is the main controller of the device. It provides data processing and management of all other components. This board uses an *ESP8266 microcontroller* that supports *Wi-Fi* connectivity. The *UART-USB* interface allows you to easily program and power the board through the *micro connector USB* .

RFID reader MFRC522 is responsible for reading data from *RFID* tags. It operates at a frequency of 13.56 MHz and uses *the SPI* interface to communicate with *the NodeMCU board* . *The MFRC522* has a high read speed and provides reliable data transmission.

RFID tags are used to identify objects. Each label has a unique identifier that is read by the *MFRC522 module* . Tags operate at 13.56 MHz and can come in a variety of shapes and sizes, including cards, tags, and stickers.

A layout board is used for convenient placement and connection of prototype components. It allows you to quickly connect and disconnect components without the need for soldering. This greatly simplifies the process of debugging and testing the device.

Jumper cables are used to connect components to the breadboard. They provide reliable electrical contact and flexibility in the location of components on the breadboard. Jumper cables allow you to easily change the connection scheme during experiments and testing.

Micro cable *USB* is used to power the *NodeMCU board* and download firmware to the microcontroller. It connects to a computer or charger and provides stable power for the entire system. In addition, the micro *USB cable allows you to program the NodeMCU* board with *Arduino IDE*

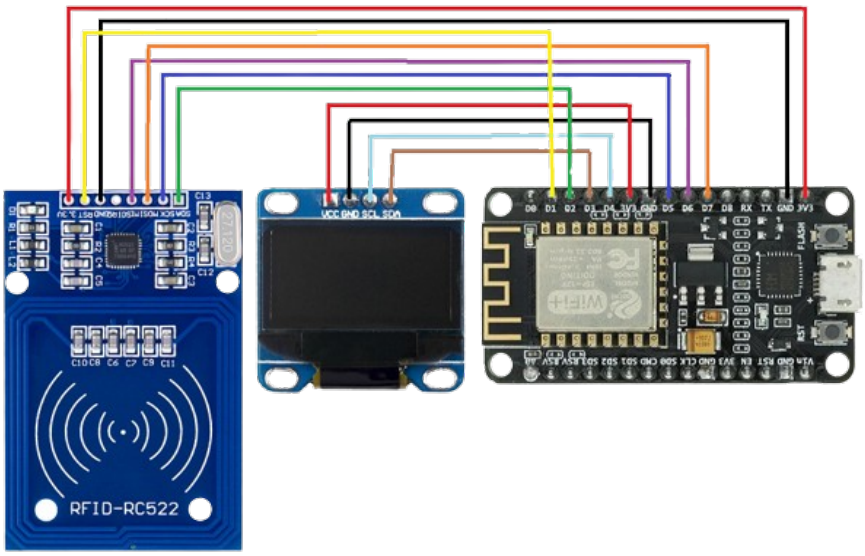


Figure 4. 3.2 – Connection diagram

The components of the prototype are connected as follows:

- *The NodeMCU V3* board is connected to a computer via a *Micro USB cable* for power and programming.
- *MFRC522 RFID* module is connected to the *NodeMCU board* via *the SPI* interface. The outputs of the module (*SDA, SCK, MOSI, MISO, RST, GND, VCC*) are connected according to the outputs on *the NodeMCU board* .
- *The OLED* display is connected to *the NodeMCU* via *the I2C* interface to display the work status and information from the labels.
- *RFID* tags are placed in the range of the reader for identification.

4.4 Testing and work algorithm

After turning on the device in fig. 4.4.1, the system automatically initializes all components. This includes setting the initial parameters for the *ESP8266 microcontroller* , configuring the connections and checking their functionality. At this stage, there is a check for the presence of errors in the equipment, and ensuring the correctness of the system boot.

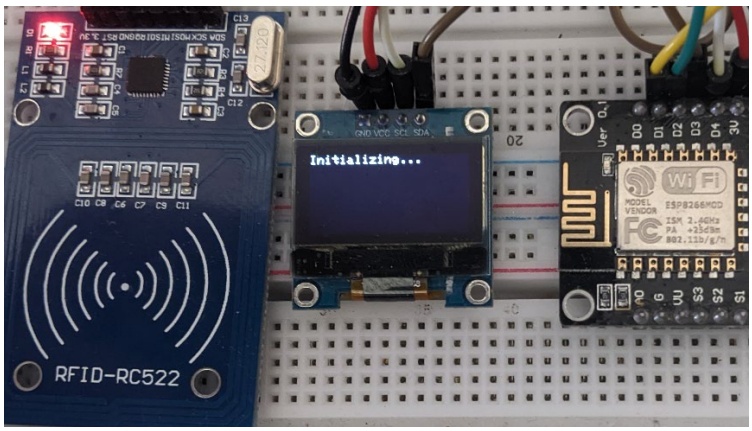
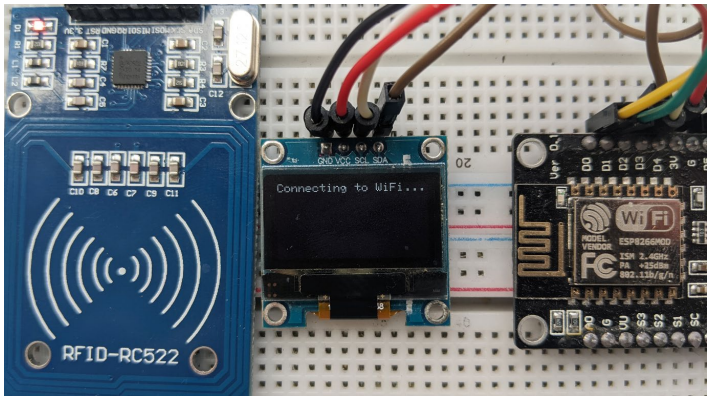


Figure 4.4.1 – Initialization of components

In fig. 4.4.2 the reader connects to an available *Wi-Fi* network. The device scans available networks, authenticates in the selected *Wi-Fi* network using the specified parameters and receives *an IP* address to ensure network connection. After successful connection to the network, in fig. 4.4.3 the system performs the user authentication procedure. Authentication includes login and password verification, which ensures secure access to system resources and data.



Wi - Fi connection

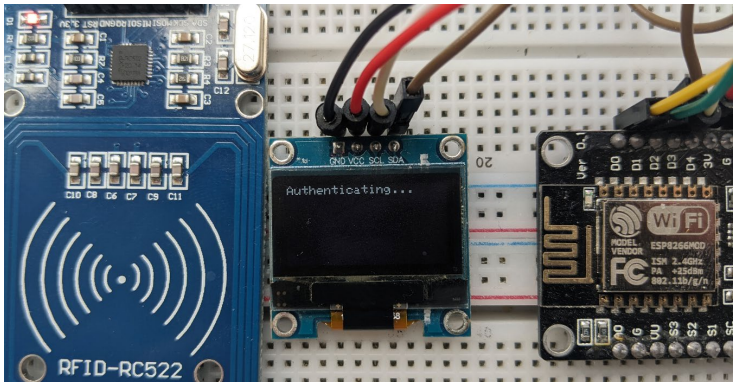


Figure 4.4.3 – Authentication process

Next, the device goes into standby mode, where it waits for *the RFID* tag to approach the reader. The device constantly scans the range of *the RFID* reader for the presence of approaching tags.

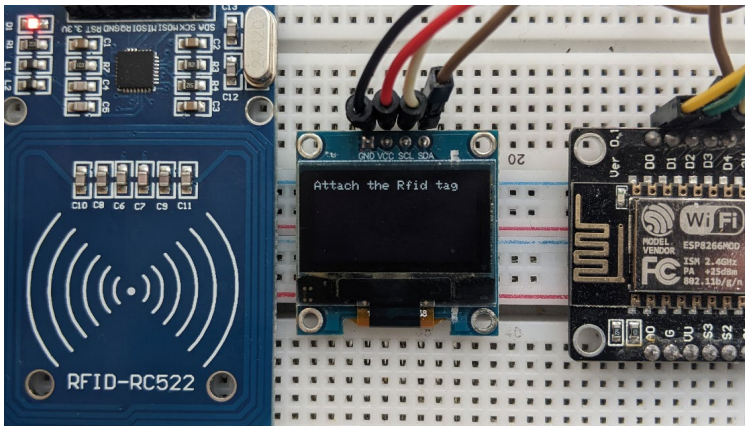


Figure 4.4.4 – *RFID* tag standby mode

When *an RFID* tag enters the reader's range, *the RFID-RC522* immediately reads its unique identifier. This information is passed on for processing, providing instant object recognition.

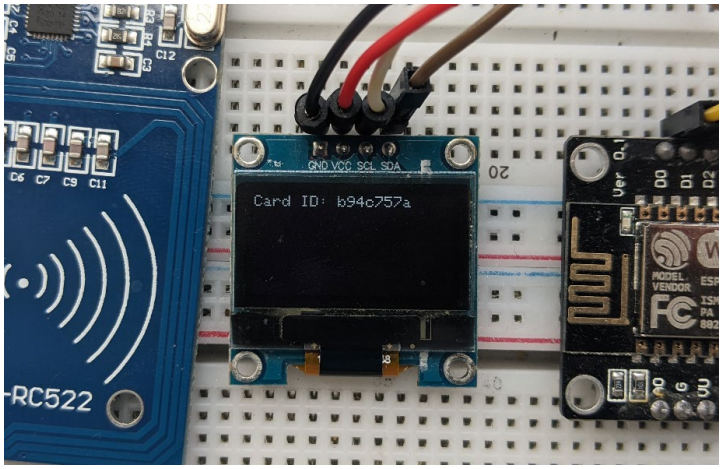


Figure 4.4.5 – Label identification

The read identifier of the label is transmitted via *Wi-Fi* to the server, where further data processing is carried out. The *OLED* display shows the status of the operation, including the successful reading of the tag and sending data to the server. This process ensures that all necessary information successfully reaches the server for further processing.

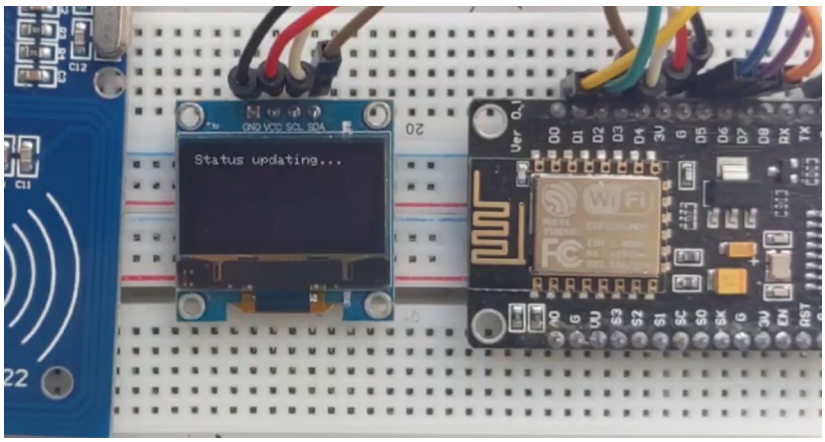


Figure 4.4.6 – Data processing and transfer to the server

After processing the data on the server, the system receives a response that contains detailed information about the identified object. This information includes data on the status, location, time of the last identification, allowing users to get a complete picture of the state of the object.

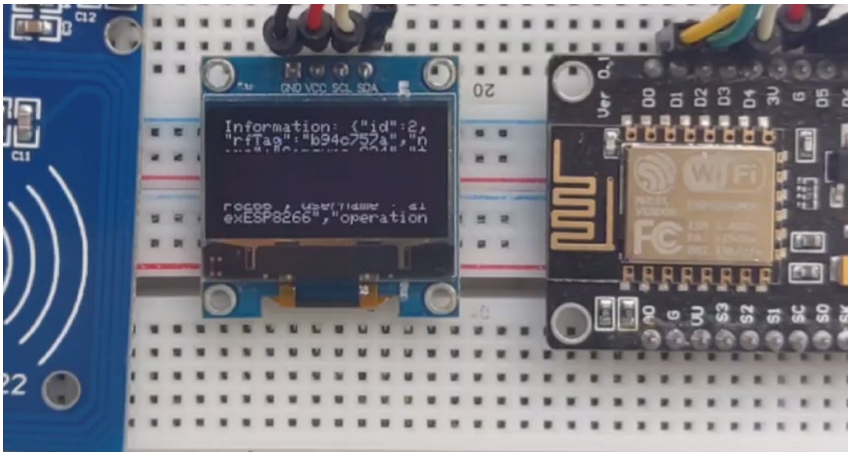


Figure 4.4.7 - Response from the server with information about the item

Having tested the successful operation of the reader, we can conclude that the service successfully authorizes, accepts data from *RFID* reader and replies with useful information.

As a result, the prototype of the reader of radio frequency tags successfully demonstrates the possibilities of integrating *RFID* technology with *IoT* devices using the *NodeMCU v3 microcontroller* . Using *the Arduino IDE* for programming allowed development flexibility and affordability, allowing the prototype to be quickly adapted to changing test and operational requirements. The received data from *RFID* tags are efficiently processed and transmitted through the network,

which confirms the high reliability and accuracy of the system. Testing showed stable operation of all components, and data received from the server provide users with up-to-date information on the status of identified objects. Thanks to the successful implementation of this prototype, it can be considered that the technology is ready for further commercialization and integration.

5 SPECIFICATION AND TESTING OF THE DEVELOPED SERVICE

An important component of any application is its *API endpoints* , which provide interaction between the client and server parts of the system. *API endpoints* allow external applications, users, and other systems to interact directly with web application services

Testing *API* endpoints is an extremely important stage during development. In the testing process, it is checked whether the implemented *API* endpoints meet the specifications and expectations, as well as whether they work correctly and stably in various usage scenarios.

5.1 API endpoints

The code presents several controllers for handling requests in a web application that uses *Spring Boot* and other technologies from the *Spring stack* . Detailed description of each endpoint :

AuthController:

- *POST /api/auth/signin* - authentication user with further return *JWT* token
- *POST /api/auth/signup* - registration new user

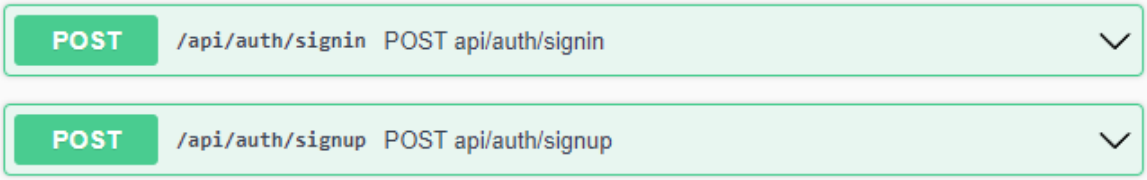


Figure 5.1.1 - *Endpoints AuthController*

ImageUploadController:

- *POST / api / image / upload* - upload an image for the current user.
- *POST / api / image /{ itemId }/ upload* - from uploading an image to a specific post.
- *GET / api / image / profileImage* - returns the profile image of the current user.

- *GET / api / image /{ itemId }/ image* - returns the image attached to a specific post.

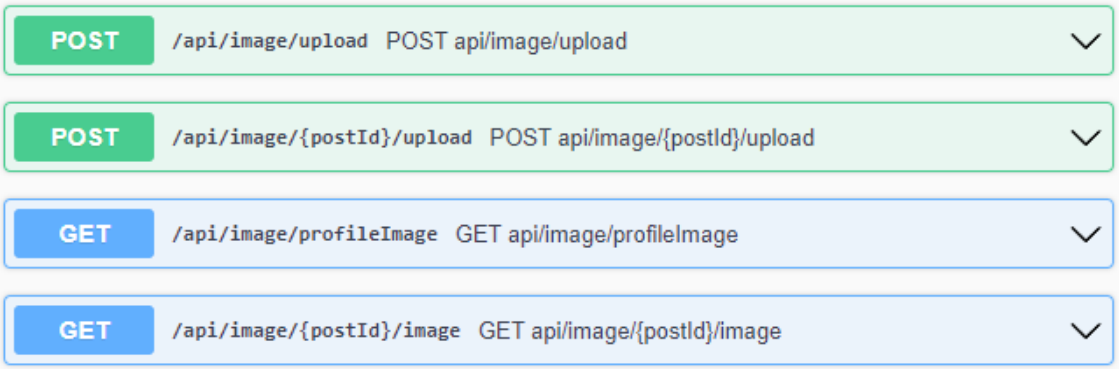


Figure 5.1.2 - *Endpoints ImageUploadController*

ItemController:

- *POST / api / item / create* - creation of a new item
- *POST /api/item/update* - updating item data
- *POST / api / item /{ rfTag }/ update* - update data about the item using *an RFID* tag.
- *GET /api/item/all* - getting a list of all items.
- *GET / api / item / user / items* - getting a list of all items of the current user.
- *POST / api / item /{ itemId }/ delete* - deleting a certain item.

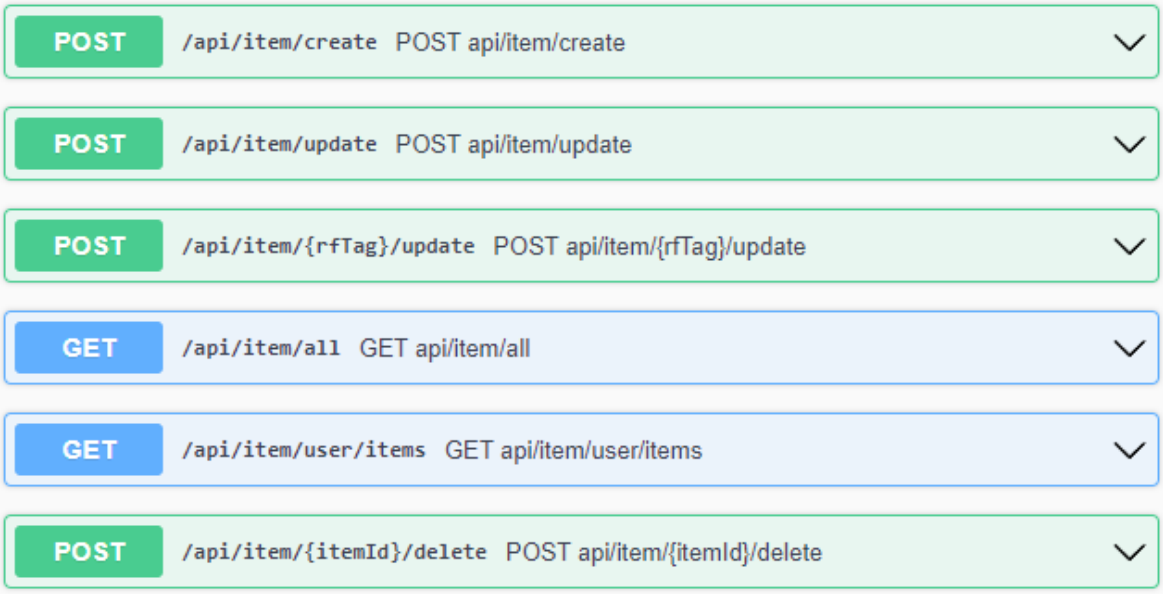


Figure 5.1.3 - *Endpoints ItemController*

RemarkController :

- *POST / api / remark /{ itemId }/ create* - creating a comment on a certain subject.
- *GET / api / remark /{ itemId }/ all* - about keeping all comments on a certain item.
- *POST /api/remark /{ remarkId}/delete* - delete comment

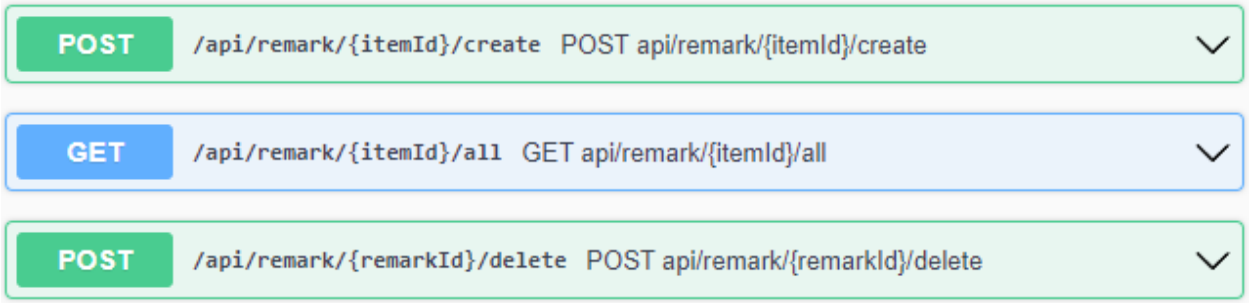


Figure 5.1.4 - *Endpoints RemarkController*

UserController:

- *GET / api / user /* - obtaining data about the current user.
- *GET / api / user /{ userId }* - getting the user profile by his *ID* .
- *POST /api/user/ update* - user profile update.

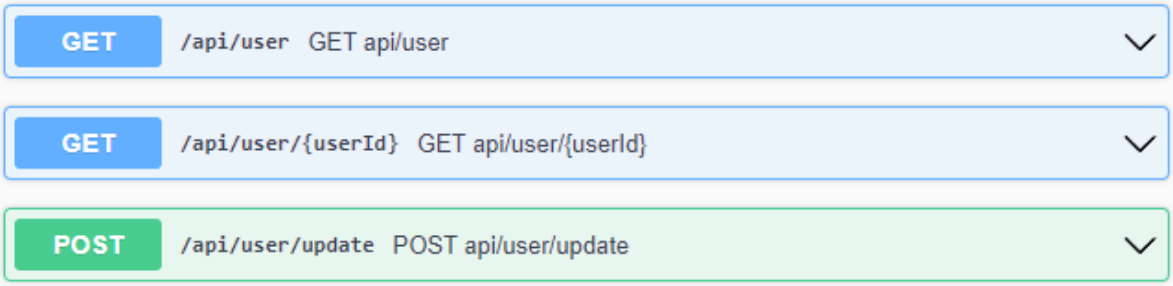


Figure 5.1.5 - *Endpoints UserController*

5.2 Testing of the developed service

PostMan is a popular tool for developers that allows you to test *APIs* , in particular *REST* services. It helps to send requests to the server and view responses, simplifying the process of debugging and developing software interfaces. *PostMan* provides the ability to group requests into collections and includes tools for testing authentication, caching, and other aspects of web requests. It is a convenient tool for quickly creating and testing *APIs* without the need to write additional code . [27]

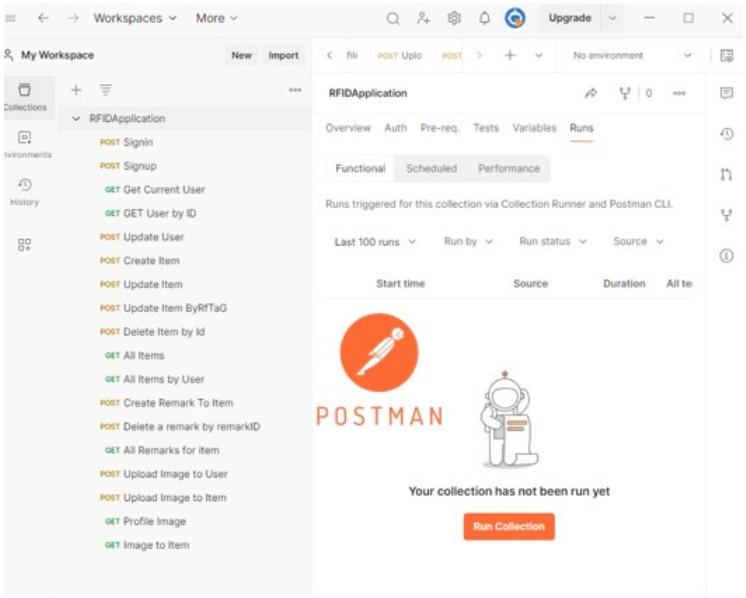


Figure 5.2.1 – *PostMan test environment*

PostMan interface with an open project in Fig. 5.2.1, which shows the tabs for different query methods and their configurations. The window is divided into parts: a navigation panel with collections of requests on the left side, a central part with parameters of the selected request, and a right part for displaying the results of responses to requests.

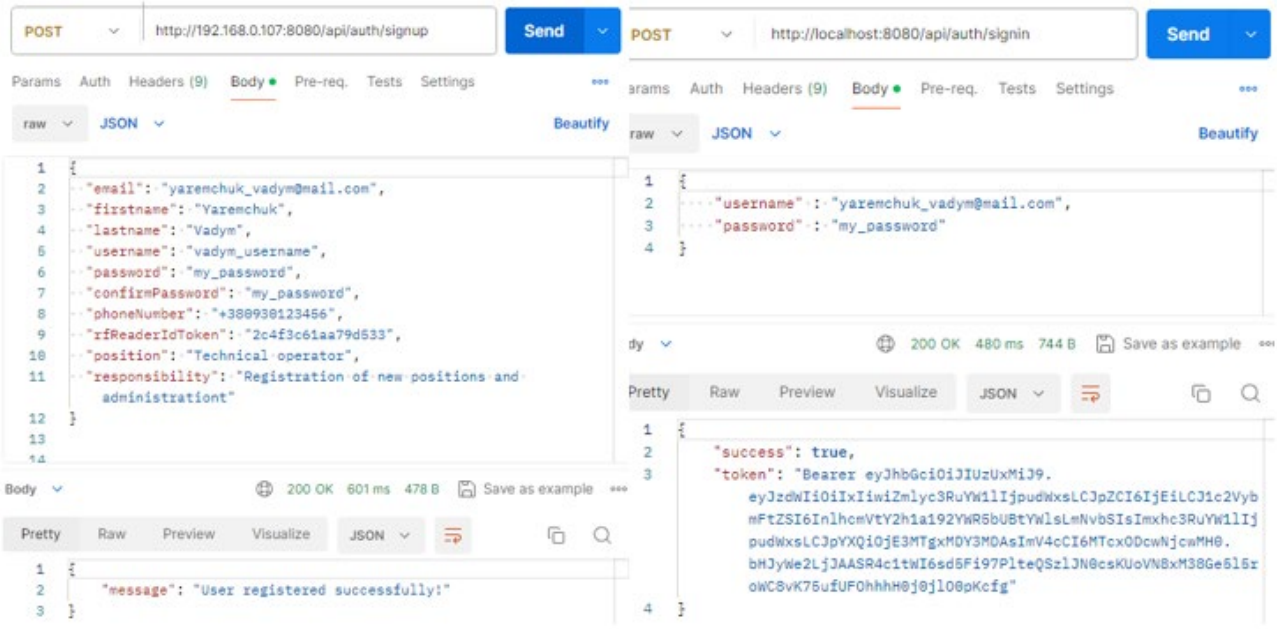


Figure 5.2.2 - *Endpoints AuthController*

Rice. 5.2.2 demonstrates the requests to *the AuthController* in *PostMan* that include the */signup* and */signin* user capabilities. Each request contains fields for

entering user data, such as name, e-mail address, password, which can be sent to the server for processing.

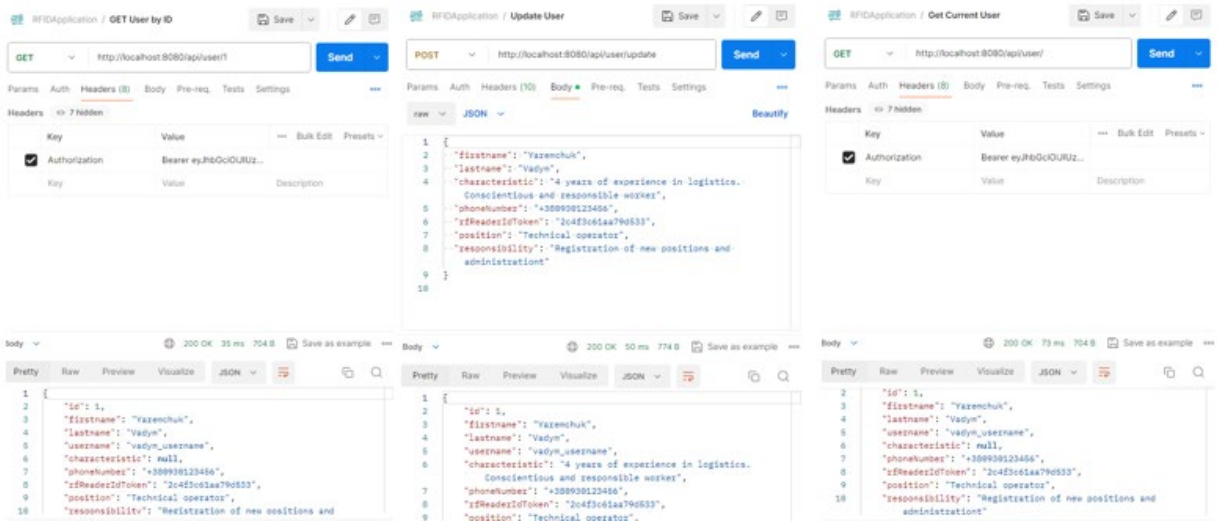


Figure 5.2.3 – *Endpoints UserController*

With the use of *PostMan* for the purpose of testing *UserController* , fig. 5.2.3 displays *endpoints* for viewing and updating user profiles, including getting user data */getuser* , updating user data */update user* , and deleting user */deleteuser* .

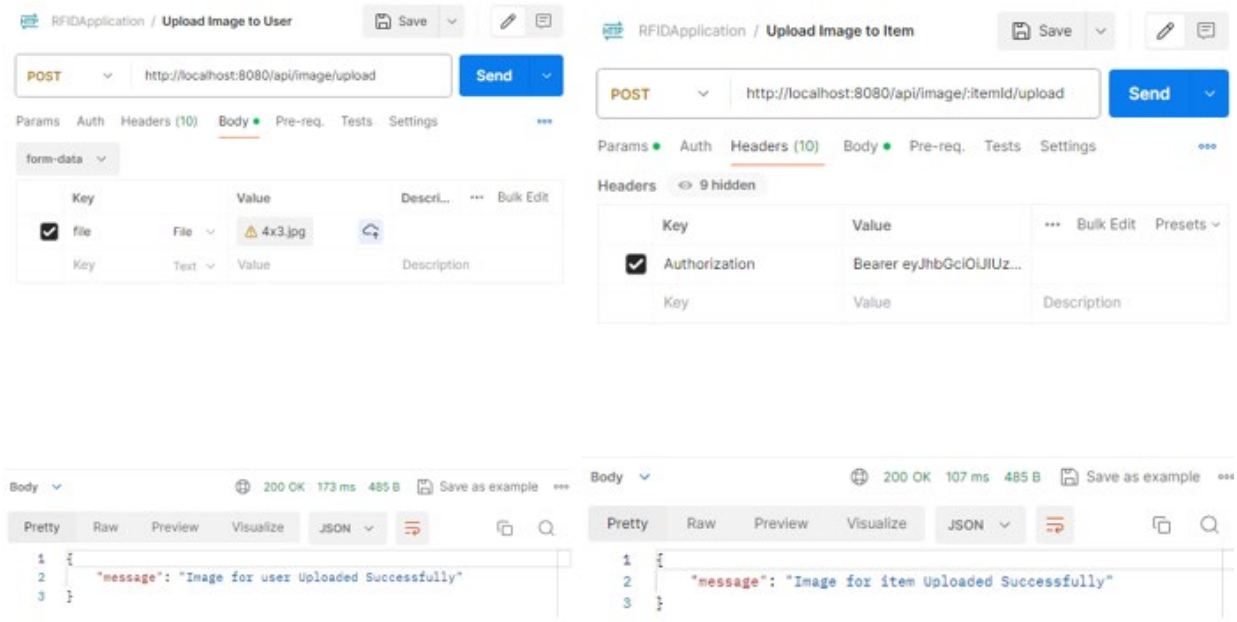


Figure 5.2. 4 – *Endpoints ImageController*

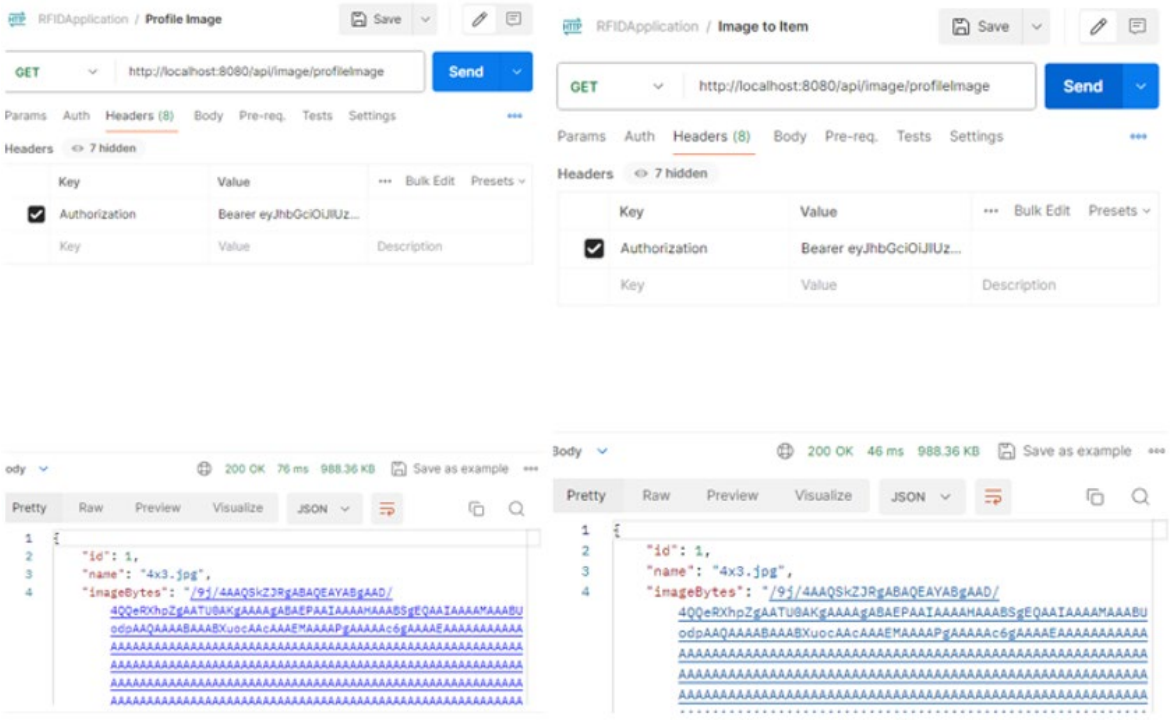


Figure 5.2.5 - *Endpoints ImageController*

Requests for managing user images through *ImageController* are presented in Fig. 5.2.4 and 5.2.5. This includes actions such as uploading an image */uploadImage* , getting an image */getImage* , deleting an image */deleteImage* , as well as additional options related to processing user images.

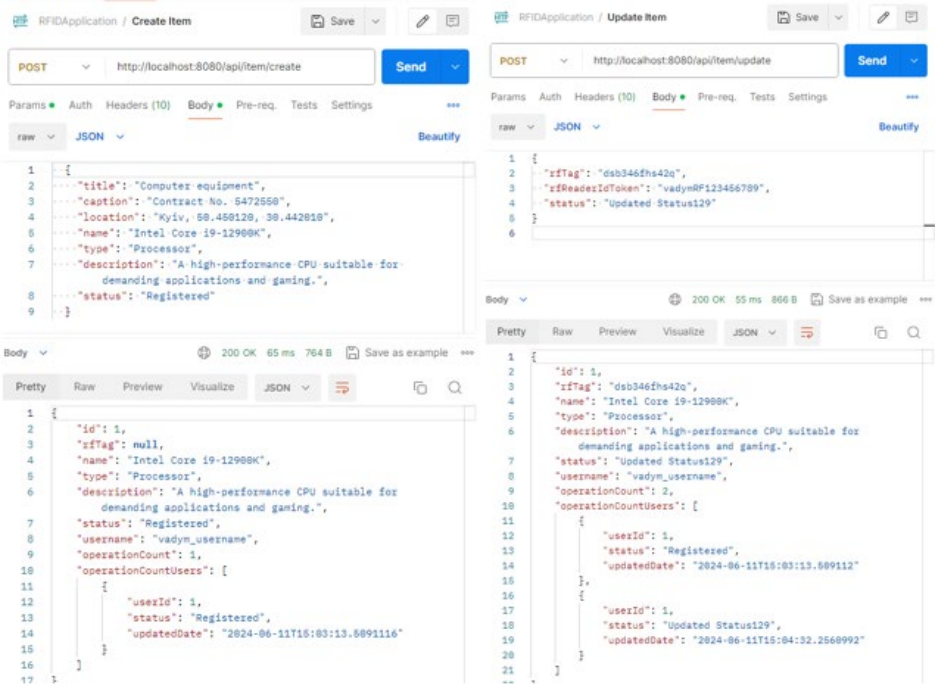


Figure 5.2. 6 – *Endpoints ItemController*

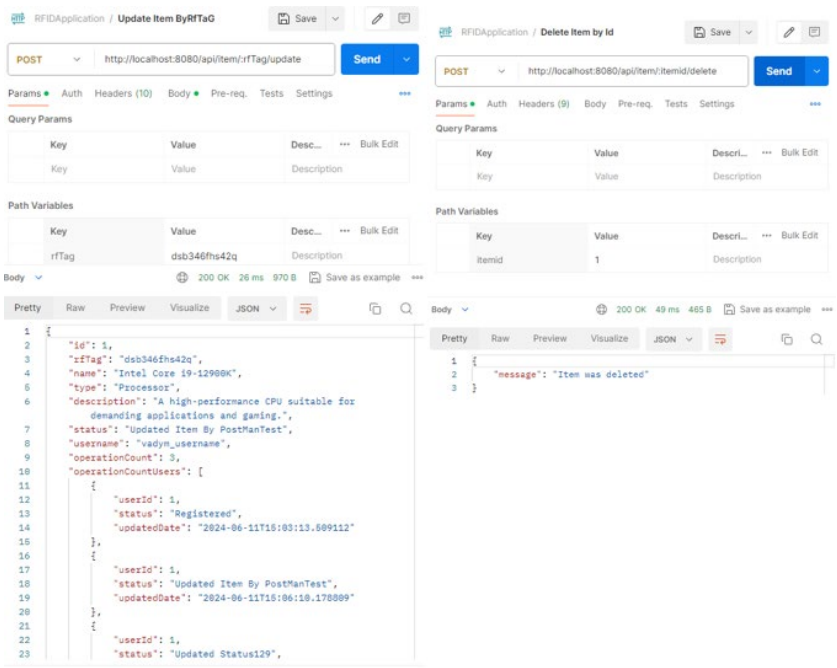


Figure 5.2.7 - *Endpoints ItemController*

The configuration of requests for creating, updating, obtaining and deleting object data through *the ItemController* is illustrated in fig. 5.2.6 and 5.2.7. They show a window with the possibility of entering data for a new object or updating an existing one, and also include additional functions for working with objects, such as deleting and viewing a list of all objects.

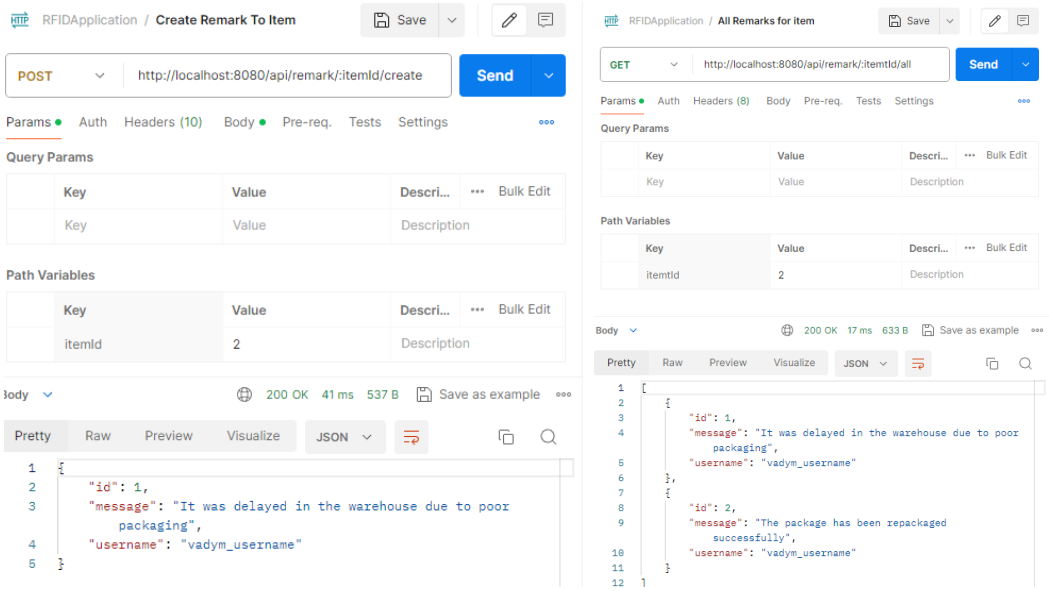


Figure 5.2.8 - *Endpoints RemarkController*

Rice. 5.2.8 displays requests to manage object comments via *RemarkController* . Functions include creating comments *addRemark* , viewing comments / *getRemarks* and deleting comments / *deleteRemark* .

During testing, using *PostMan* , it was proven that all *API* endpoints work according to the specification, and ensure stable and reliable operation of the service. Each of the controllers efficiently processes requests and responds in accordance with the business logic of the application, which demonstrates the high integration and scalability of the developed solution. Testing also revealed no significant issues or bugs, highlighting the effectiveness of the original *API design and implementation* .

IN CONCLUSIONS

As a result of the diploma work, a universal *IoT* service was created for the identification and tracking of objects using *RFID* technology. The main goal of the work was the development and testing of a reliable system that provides effective management of objects. In the course of the work, an analysis of the subject area was carried out, which included consideration of various identification technologies, as well as *IoT* services and existing solutions on the market.

To implement the service, modern technologies were chosen, in particular, the *Java programming language, the integrated IntelliJ* development environment *IDEA , as well as Spring* frameworks *Framework* , *Spring Boot* , *Spring Data JPA* and *Spring Security* . The *PostgreSQL* database was used to store data , and the project was managed using *Maven* .

RFID reader based on the *NodeMCU microcontroller* was created *v3 ESP8266 and RFID* module - *RC522* , and the reader control and data processing software was developed using *Arduino IDE An API* has been created *endpoints* for interaction with the service, and testing was carried out using *PostMan* , which confirmed the stable and reliable operation of all system components.

Developed by *REST The API* service allows you to process requests and identify objects using *RFID* tags, effectively tracking the movement of objects, controlling their statuses and history. Assigning *an RFID* tag to each object simplifies identification and tracking, which facilitates management.

This project demonstrates technical achievements in the field of *IoT* and *RFID* technologies, opening wide prospects for the implementation of innovative solutions in various industries. Thanks to this system, you can effectively track the movement of objects, control their statuses and history, which allows you to increase productivity.

LIST OF REFERENCE SOURCES

1. QR codes [Electronic resource] // NBookpart – Resource access mode: https://nbookpart.com.ua/yak-skanuvaty-qr-kod-na-android-z-dodatkom-ta-bez/

2. Wi-Fi and Bluetooth [Electronic resource] // Embarcados - Resource access mode: https://embarcados.com.br/wi-fi-bluetooth/

3. Wi-Fi and Bluetooth [Electronic resource] // Klaster - Resource access mode: https://klaster.ua/ua/stati-i-obzory/wi-fi-i-bluetooth-besprovodnye-sistemy-peredachi- information/

4. NFC [Electronic resource] // InStor – Resource access mode: https://instor.com.ua/komplekt-nfc-metok-broadlink-nfc-tag-srn1-10-shtuk-id1340/

5. NFC [Electronic resource] // Wikipedia - Resource access mode: https://uk.wikipedia.org/wiki/Near-field_communication

6. RFID [Electronic resource] // ID Card - Resource access mode: https://idcard.com.ua/ua/blog/chto-takoe-sistema-rfid-v-chem-ee-osobennosti-ispolzovaniya/

7. IoT [Electronic resource] // Cyber1Defense - Resource access mode: https://cyber1defense.com/tag/iot/

8. IoT [Electronic resource] // Amazon Web Services - Resource access mode: https://aws.amazon.com/ru/what-is/iot/

9. IoT [Electronic resource] // Atiko - Resource access mode: https://www.atiko.com.ua/articles-ua/chto-takoe-iot-prostymi-slovami/

10. Zebra Technologies [Electronic resource] // Zebra - Resource access mode: https://www.zebra.com/us/en/products/rfid.html

11. Avery Dennison [Electronic resource] // YouTube - Resource access mode: https://www.youtube.com/supported_browsers?next_url=https%3A%2F%2Fwww.youtube.com%2Fwatch%3Fv%3Dt7qe61Wxdf0&ab_channel=AveryDennison

12. Avery Dennison [Electronic resource] // Avery Dennison RFID - Resource access mode: https://rfid.averydennison.com

13. Honeywell [Electronic resource] // Honeywell - Resource access mode: https://honeywell-spsservice.com/Content/RMA_Portal_User_Manual_Customers_ru.pdf

14. Alien [Electronic resource] // AtlasRFIDstore – Resource access mode: https://support.atlasrfidstore.com/article/59-using-the-alien-gateway-command-line

15. JVM [Electronic resource] // LinkedIn - Resource access mode: https://www.linkedin.com/pulse/java-virtual-machine-kishan-kumar-1f/

16. JVM [Electronic resource] // Foxminded - Resource access mode: https://foxminded.ua/jvm-tse/

17. IntelliJ IDEA [Electronic resource] // JetBrains - Resource access mode: https://www.jetbrains.com/idea/

18. Eclipse [Electronic resource] // Eclipse - Resource access mode: https://www.eclipse.org/

19. NetBeans [Electronic resource] // NetBeans - Resource access mode: https://netbeans.apache.org/front/main/index.html

20. Spring [Electronic resource] // Spring - Resource access mode: https://docs.spring.io/spring-framework/docs/4.3.x/spring-framework-reference/html/overview.html

21. Spring Boot [Electronic resource] // GeeksforGeeks - Resource access mode: https://www.geeksforgeeks.org/how-spring-boot-application-works-internally/

22. Spring Security drawing [Electronic resource] // Medium - Resource access mode: https://medium.com/@greekykhs/springsecurity-part-3-spring-security-flow-7da9cc3624ab

23. Spring Security [Electronic resource] // Spring - Resource access mode: https://docs.spring.io/spring-security/reference/servlet/architecture.html

24. PostgreSQL Architecture [Electronic resource] // PGDataEra - Resource access mode: https://pgdataera.com/category/postgresql-database-architecture/

25. Uml JWT [Electronic resource] // Visual-Paradigm - Resource access mode: https://online.visual-paradigm.com/community/share/jwt-vpd-jx9azc0t5

26. Esp8266 [Electronic resource] // Arduino - Resource access mode: https://arduino.ua/prod1492-wi-fi-modyl-nodemcu-esp8266

27. Postman [Electronic resource] // Postman - Resource access mode: https://www.postman.com/

APPENDIX A

Code listing of the microcontroller software module

```
#include <SPI.h>
#include <MFRC522.h>
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>
#include <ArduinoJson.h>

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SS_PIN 4
#define RST_PIN 5
MFRC522 mfrc522(SS_PIN, RST_PIN);

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_SDA 0
#define OLED_SCL 2
#define OLED_RESET -1

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
OLED_RESET);
const char *ssid = " SSID ";
const char *password = " Password ";
const char* auth_url = "http://192.168.0.107:8080/api/auth/signin";
const char* status_assign = "Assigned by ESP8266";
const char* status_update = "Updated by ESP8266";
const char* device_token = "2c4f3c61aa79d533";
const char* username = "yaremchuk_vadym@mail.com";
```

```
const char* user_password = "my_password";

String OldCardID = "";
unsigned long previousMillis = 0;
String jwt_token = "";

void displayStatus(String message) {
display.clearDisplay();
display.setCursor(0, 0);
display.println(message);
display.display();
Serial.println(message);
}

void setup() {
delay(1000);
Serial.begin(115200);
SPI.begin(); // Init SPI bus
mfrc522.PCD_Init(); // Init MFRC522 card

// Initialize OLED display
Wire.begin(OLED_SDA, OLED_SCL);
if (!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
Serial.println(F("SSD1306 allocation failed"));
for(;;); // halt
}
display.clearDisplay();
display.setTextColor(WHITE);
display.setTextSize(1);
display.setCursor(0,0);
```

```
display.println("Initializing...");
display.display();
connectToWiFi();
authenticate();
}
void loop() {
if (!WiFi.isConnected()) {
connectToWiFi();
authenticate();
}
displayStatus("Attach the Rfid tag");
delay(500);
if (millis() - previousMillis >= 4000) {
previousMillis = millis();
OldCardID = "";
}
delay(50);
// look for new card
if (!mfrc522.PICC_IsNewCardPresent()) {
return;
}
// Select one of the cards
if (!mfrc522.PICC_ReadCardSerial()) {
return;
}
String CardID = "";
for (byte i = 0; i < mfrc522.uid.size; i++) {
CardID += String(mfrc522.uid.uidByte[i], HEX);
}
if (CardID == OldCardID) {
```

```
return;
} else {
OldCardID = CardID;
}
Serial.print("Card detected, ID: ");
displayStatus("Card ID: " + CardID);
delay(1500);
Serial.println(CardID);
SendCardID(CardID);
delay(1000);
}
void SendCardID(String Card_uid) {
Serial.println("Sending the Card ID");
if (WiFi.isConnected() && jwt_token != "") {
WiFiClient client;
HTTPClient http;
http.begin(client, "http://192.168.0.107:8080/api/item/update");
http.addHeader("Content-Type", "application/json");
http.addHeader("Authorization", jwt_token);
DynamicJsonDocument doc(256);
doc["rfTag"] = Card_uid;
doc["rfReaderIdToken"] = device_token;
doc["status"] = status_assign;

String requestBody;
serializeJson(doc, requestBody);

int httpCode = http.POST(requestBody);
String payload = http.getString();
```

```
Serial.print("HTTP Response code: ");
Serial.println(httpCode);
Serial.print("Response payload: ");
Serial.println(payload);

if (httpCode == 200) {
displayStatus("RFid assigned");
delay(2000);
displayStatus("Information: " + payload);
delay(4500);
}

if (httpCode == 400 || httpCode == 500 || httpCode == 401 || httpCode == 501)
{

http.end();
displayStatus("Status updating...");
delay(2000);
retrySendCardID(Card_uid);
}
http.end(); // Close connection
} else {
Serial.println("Failed to send card ID. No Wi-Fi or missing token.");
displayStatus("Failed to send card ID. No Wi-Fi or missing token.");
}
}
void retrySendCardID(String Card_uid) {
Serial.println("Retrying to send Card ID with different endpoint");
if (WiFi.isConnected() && jwt_token != "") {
WiFiClient client;
```

```
        HTTPClient http;
        String retryUrl = "http://192.168.0.107:8080/api/item/" + Card_uid +
"/update";
        http.begin(client, retryUrl);
        http.addHeader("Content-Type", "application/json");
        http.addHeader("Authorization", jwt_token);

        DynamicJsonDocument doc(256);
        doc["status"] = status_update;

        String requestBody;
        serializeJson(doc, requestBody);

        int httpCode = http.POST(requestBody);
        String payload = http.getString();

        Serial.print("HTTP Response code on retry: ");
        Serial.println(httpCode);
        Serial.print("Response payload on retry: ");
        Serial.println(payload);

        if (httpCode == 200) {
        displayStatus("Status updated");
        delay(1000);
        displayStatus("Information: " + payload);
        delay(4500);
        }
        if (httpCode == 400 || httpCode == 500 || httpCode == 401 || httpCode == 501)
{
        displayStatus("Unidentified");
```

```
delay(1500);
displayStatus("Try another tag");
delay(1000);
}

http.end(); // Close connection
} else {
Serial.println("Failed to send card ID on retry. No Wi-Fi or missing token.");
displayStatus("Failed to send");
delay(1000);
}
}
void connectToWiFi() {
WiFi.mode(WIFI_OFF);
delay(1000);
WiFi.mode(WIFI_STA);
Serial.print("Connecting to WiFi...");
displayStatus("Connecting to WiFi...");
delay(1000);
Serial.println(ssid);
WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("Connected to WiFi: " + String(ssid));
displayStatus("Connected to WiFi");
delay(1000);
```

```
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
}
void authenticate() {
unsigned long lastAttemptTime = 0;
const unsigned long retryInterval = 10000;
bool authenticated = false;

while (!authenticated) {
if (WiFi.isConnected()) {
if (millis() - lastAttemptTime > retryInterval || lastAttemptTime == 0) {
WiFiClient client;
HTTPClient http;
http.begin(client, auth_url);
http.addHeader("Content-Type", "application/json");

DynamicJsonDocument doc(256);
doc["username"] = username;
doc["password"] = user_password;

String requestBody;
serializeJson(doc, requestBody);

Serial.println("Authenticating...");
displayStatus("Authenticating...");
delay(1000);
int httpCode = http.POST(requestBody);
String payload = http.getString();

Serial.print("HTTP Response code: ");
```

```
Serial.println(httpCode);
displayStatus("HTTP " + String(httpCode));
delay(1000);
Serial.print("Response payload: ");
Serial.println(payload);

if (httpCode == 200) {
DynamicJsonDocument responseDoc(512);
deserializeJson(responseDoc, payload);
bool success = responseDoc["success"];
if (success) {
jwt_token = responseDoc["token"].as<String>();
Serial.println("Authentication successful");
displayStatus("Authenticated");
delay(1500);
Serial.println("Token: " + jwt_token);
authenticated = true;
} else {
Serial.println("Authentication failed");
displayStatus("Not authenticated");
delay(1000);
}
}

else {
Serial.println("Failed to authenticate");
displayStatus("Not authenticated");
delay(1000);
}
http.end(); // Close connection
```

```
lastAttemptTime = millis();
}
} else {
Serial.println("No Wi-Fi connection");
displayStatus("No Wi-Fi connection");
delay(1000);
}

if (!authenticated) {
delay(1000);
}
}
}
```