BINUS UNIVERSITY

BINUS INTERNATIONAL

Object Oriented Programming

Final Project Report

Student Information:

Surname: Leewin **Given Name:** Jonathan Leewin **Student ID:** 2702319455

Course Code : COMP6175 **Course Name :** Object Oriented Programming

Class : L1AC **Lecturer :** Jude Joseph Lamug Martinez, MCS

Type of Assignment: Final Project Report

Submission Pattern

Due Date : 15 January 2023 **Submission Date :** June 20 2024

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as perlecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, whichautomatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of thesubmission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating, and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity, and other possible penalties executed by the university. Please refer tothe related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept, and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:

[illegible]

Jonathan Leewin

Table of Contents

A. Introduction ... 5
1. Background ... 5
2. Problem Identification ... 6

B. Project Specification ... 7
1. User Management ... 7
   a. Create User Accounts ... 7
   b. Update User Accounts ... 7
   c. Role-Based Access Control (RBAC) ... 7
2. Call Data Management ... 8
   a. Record Call Data ... 8
   b. Store Call Data ... 8
3. Billing and Reporting ... 8
   a. Generate Billing Information ... 8
   b. Provide Reporting Capabilities ... 9
4. Authentication and Security ... 9
   a. User Authentication ... 9
   b. Data Security ... 9
5. Integration ... 10
   a. Integration with TDS 600 ... 10

C. Spring Framework and Technologies Used ... 10
1. Spring Boot Starters ... 10
   a. spring-boot-starter-data-jpa ... 10
   b. spring-boot-starter-thymeleaf ... 10
   c. spring-boot-starter-web ... 10
   d. spring-boot-starter-validation ... 11
   e. spring-boot-starter-security ... 11
2. Other Libraries ... 11
   a. Java-WebSocket (version 1.5.6) ... 11
   b. jSerialComm (version between 2.0.0 and 3.0.0) ... 11
   c. jython-standalone (version 2.7.2) ... 11
   d. mysql-connector-j (version 8.2.0) ... 12
   e. Lombok ... 12
3. Python for WebSocket and Serial Reader ... 12

a. WebSocket Communication 12
b. Serial Data Reading 12
4. HTML, CSS, and JavaScript Integration and MySQL 13
a. HTML Integration 13
b. CSS Styling 13
d. JavaScript Functionality 13
e. Thymeleaf Templating Engine 13
f. MySQL Database Integration 13
D. Solution design 14
1. Class Diagram 14
2. Flow Chart Spring Architecture 15
E. Essential Algorithms 15
User Authentication and Authorization 15
Call Data Recording 16
Billing Calculation 18
Data Management 18
F. Lesson Learned 19
Self-Reflection 19
Conclusion 20

A. Introduction

1. Background

The Billing App for the Transtel Communications TDS 600 is an advanced software solution designed to automate and optimize the billing processes for a private branch exchange (PABX) system. The TDS 600, a product of Transtel Communications, is an integral part of many organizations' telecommunication infrastructure, providing features such as call transfer, voicemail, call queuing, and automated attendant services. These features are crucial for ensuring smooth and efficient communication within organizations and between organizations and external parties.

Despite the sophisticated capabilities of the PABX TDS 600, managing the extensive call data and associated billing information can be challenging. Organizations that rely on manual methods for tracking call details, calculating charges, and generating billing reports often encounter significant inefficiencies and a high potential for errors. Manual processes can lead to inaccuracies in billing, delays in processing, and increased operational costs. Additionally, the security of sensitive call data and the management of user access are critical concerns that require robust solutions to prevent unauthorized access and data breaches.

The Billing App leverages modern software development practices and technologies to provide an automated and reliable solution for managing the billing processes of the PABX TDS 600 system. By automating these processes, the application enhances accuracy, efficiency, and security, thereby addressing the primary challenges faced by organizations using the PABX system.

2. Problem Identification

Accurate Tracking of Call Data: Each call made or received through the PABX system needs to be accurately logged with details such as call duration, caller ID, callee ID, and timestamps. Manual logging can lead to discrepancies, data loss, and inaccurate billing.

User Account and Role Management: Organizations often have multiple users with varying levels of access and permissions. Managing these user accounts manually is inefficient and increases the risk of unauthorized access and security breaches. There is a need for a system that can efficiently manage user accounts and enforce role-based access control.

Data Security and Authentication: Sensitive information, including call data and user credentials, needs to be securely stored and accessed. Implementing secure authentication mechanisms is essential to prevent unauthorized access and data breaches. This requires robust security protocols to safeguard data integrity and confidentiality.

Efficient Data Retrieval and Reporting: Generating detailed and accurate billing reports requires quick access to call data. Manual retrieval of data is slow and can delay the billing process, affecting operational efficiency. There is a need for a system that can generate comprehensive reports on call data in a timely manner.

integration with Existing Telecommunication Infrastructure: The billing system must integrate seamlessly with the existing PABX TDS 600 infrastructure to ensure real-time data capture and processing. Any disconnect between the systems can lead to data inconsistencies and operational issues. Seamless integration is crucial for accurate and efficient billing.

To address these challenges, the Billing App for the Transtel Communications PABX TDS 600 has been designed to automate and optimize the billing process, ensuring accuracy, efficiency, and security.

B. Project Specification

The Billing App for the Transtel Communications PABX TDS 600 aims to provide a comprehensive solution to the identified problems through a series of well-defined features and specifications.

1. User Management
   The user management module is designed to handle the creation, updating, and deletion of user accounts while implementing role-based access control (RBAC) to ensure secure and efficient management of user permissions.

   a. Create User Accounts
      Administrators can create new user accounts, ensuring that each user has the appropriate access rights and credentials to use the system. The system supports the creation of multiple user roles, each with specific permissions. This functionality is crucial for onboarding new employees or contractors who need access to the billing system.

   b. Update User Accounts
      Existing user details, such as names, passwords, and roles, can be updated to reflect any changes in the organization’s structure or personnel. This ensures that user information is always up-to-date and accurate, allowing for seamless transitions when employees change roles or responsibilities.

   c. Role-Based Access Control (RBAC)
      Existing user details, such as names, passwords, and roles, can be updated to reflect any changes in the organization’s structure or personnel. This ensures that user information is always up-to-date and accurate, allowing for seamless transitions when employees change roles or responsibilities.

2. Call Data Management
   The call data management module ensures that all call information is accurately recorded and stored in the system, providing a reliable basis for billing and reporting..

   a. Record Call Data
      The system automatically logs detailed information for each call made or received through the PABX system. This includes call duration, numbers dialed, caller and callee details, timestamps, and other relevant data. The automated logging ensures that all call data is accurately recorded without manual intervention, reducing the risk of errors and omissions.

   b. Store Call Data
      Call data is efficiently stored in a database, ensuring that it is easily accessible for retrieval and processing. The storage mechanism is designed to handle large volumes of data while maintaining quick access times. This ensures that the system can scale with the organization's needs, providing reliable performance even as the volume of call data grows.

3. Billing and Reporting
   The billing and reporting module provides the core functionalities for calculating charges and generating comprehensive reports on call data.

   a. Generate Billing Information
      The system calculates billing amounts based on call duration, predefined rates, and other billing parameters. This ensures accurate and consistent billing for all users and departments. The billing information is generated automatically, reducing the risk of human error and ensuring that charges are applied fairly and consistently.

b. Provide Reporting Capabilities

The system generates comprehensive reports on call data, including summaries, trends, and detailed records. These reports are essential for analysis, decision-making, and auditing purposes. The reporting capabilities help organizations in monitoring and managing their telecommunication expenses effectively, providing valuable insights into usage patterns and potential areas for cost savings.

4. Authentication and Security

The authentication and security module ensures that only authorized users can access the system and that sensitive data is protected from unauthorized access and breaches.

a. User Authentication

The system implements secure authentication mechanisms to ensure that only authorized users can access the system. This includes password protection, role-based access control, and potentially multi-factor authentication for enhanced security. Secure authentication helps in preventing unauthorized access to the system, protecting sensitive data and maintaining the integrity of the billing process.

b. Data Security

Sensitive call data and user information are protected through encryption and other security measures. This ensures that data is secure both at rest and in transit, preventing unauthorized access and data breaches. Robust data security measures are crucial for maintaining the confidentiality and integrity of the data, ensuring compliance with industry regulations and protecting the organization’s reputation.

5. Integration

The integration module ensures that the billing app works seamlessly with the existing TDS 600 system, enabling real-time data capture and processing.

a. Integration with TDS 600

The billing app integrates smoothly with the existing PABX TDS 600 system, allowing for real-time data capture and processing. This integration ensures that call data is accurately recorded and processed without manual intervention, providing reliable and up-to-date information for billing and reporting. Seamless integration helps in achieving operational efficiency and accuracy in billing, reducing the risk of data discrepancies and improving overall system performance.

C. Spring Framework and Technologies Used

1. Spring Boot Starters

a. spring-boot-starter-data-jpa

Provides JPA (Java Persistence API) support for database access, allowing for the easy implementation of data access layers. This starter simplifies database operations and integrates seamlessly with Spring Data JPA, enabling efficient data storage and retrieval.

b. spring-boot-starter-thymeleaf

Integrates the Thymeleaf templating engine, which is used for rendering dynamic web pages. Thymeleaf is a modern server-side Java template engine for web and standalone environments, providing a natural way to work with HTML and XML.

c. spring-boot-starter-web

Facilitates the development of web applications by providing embedded Tomcat, web starter configurations, and RESTful web services. This starter is essential for building and running Spring-based web applications, enabling the creation of robust and scalable web services.

d. spring-boot-starter-validation

Adds support for bean validation, ensuring that data inputted into the system adheres to defined constraints and rules. This starter helps in validating user input and ensuring data integrity, reducing the risk of errors and improving the overall quality of the application.

e. spring-boot-starter-security

Incorporates security features into the application, including authentication, authorization, and secure session management. This starter helps in securing the application and protecting sensitive data, providing robust security mechanisms to prevent unauthorized access and data breaches.

2. Other Libraries

a. Java-WebSocket (version 1.5.6)

Enables WebSocket communication, allowing for real-time interaction and data transfer between the client and server. WebSockets provide full-duplex communication channels over a single TCP connection, enabling efficient and low-latency data transfer.

b. jSerialComm (version between 2.0.0 and 3.0.0)

Provides serial communication capabilities, essential for interfacing with hardware components connected via serial ports. This library supports various serial communication features and protocols, enabling reliable and efficient data transfer between the billing app and the PABX system.

c. jython-standalone (version 2.7.2)

Implements Python in Java, allowing the integration of Python scripts and functionality within the Java-based application. Jython is an implementation of the Python language written in Java, enabling the use of Python’s rich libraries and tools within the billing app.

d. mysql-connector-j (version 8.2.0)

The MySQL JDBC driver enables the application to connect and interact with a MySQL database. This driver provides connectivity between Java applications and MySQL databases, enabling efficient and reliable data storage and retrieval.

e. Lombok

Provided by Spring Boot, Lombok is used to reduce boilerplate code by generating common methods like getters, setters, equals, and hashCode at compile time. Lombok helps in simplifying the code and improving readability, reducing the amount of repetitive code and making the development process more efficient

3. Python for WebSocket and Serial Reader

In addition to the Spring Framework and associated libraries, Python is utilized for WebSocket communication and serial data reading. Python's flexibility and extensive libraries make it an ideal choice for handling these functionalities

a. WebSocket Communication

Python's websocket library is used to establish and manage WebSocket connections. This allows for real-time data transfer between the PABX system and the billing application, ensuring that call data is captured and processed in real-time. WebSockets provide a reliable and efficient way to handle real-time communication, enabling quick and accurate data capture.

b. Serial Data Reading

Python's pyserial library is used for reading data from serial ports. This is essential for capturing call data from the PABX system and integrating it into the billing application. The pyserial library provides a simple interface for accessing serial ports, enabling reliable and efficient data transfer between the hardware and the application.

4. HTML, CSS, and JavaScript Integration and MySQL

a. HTML Integration

HTML files are utilized to structure the user interface components of the Billing App. They define the layout and structure of web pages, including forms for user input, tables for displaying billing information, and elements for navigation and interaction

b. CSS Styling

CSS files are employed to style the HTML components, ensuring a visually appealing and consistent user interface across the application. CSS rules define the colors, fonts, spacing, and layout of elements, enhancing usability and accessibility.

d. JavaScript Functionality

JavaScript is used to add interactivity and dynamic behavior to the Billing App's frontend. It facilitates client-side validation of user input, asynchronous communication with the backend through AJAX calls, and real-time updates of data displayed on the user interface

e. Thymeleaf Templating Engine

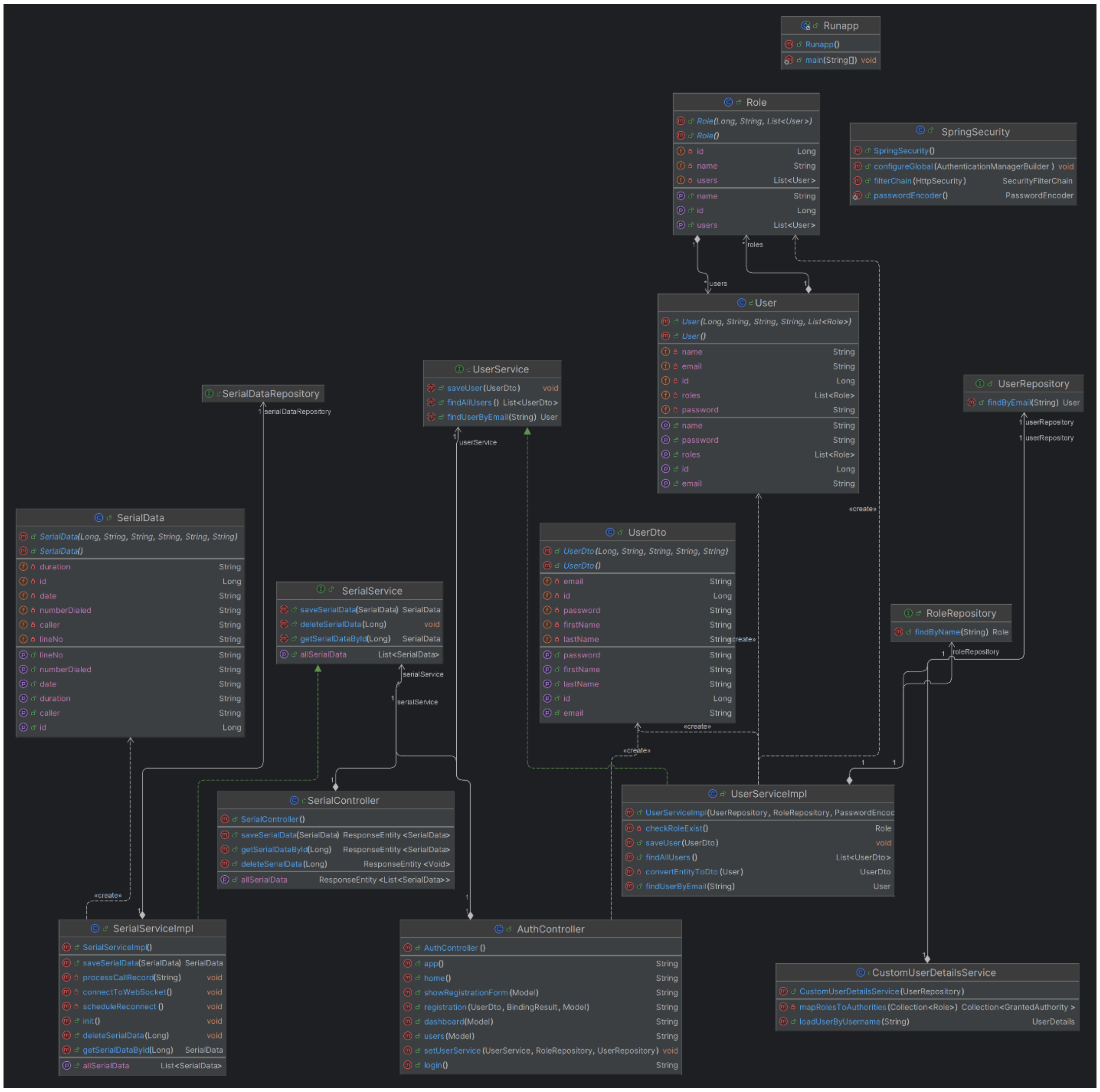
.The frontend components built with HTML, CSS, and JavaScript are seamlessly integrated into the Spring Boot application. Thymeleaf, a modern server-side Java template engine, is leveraged to render dynamic HTML content and embed server-side variables into web pages, enhancing the interaction between frontend and backend components.

f. MySQL Database Integration

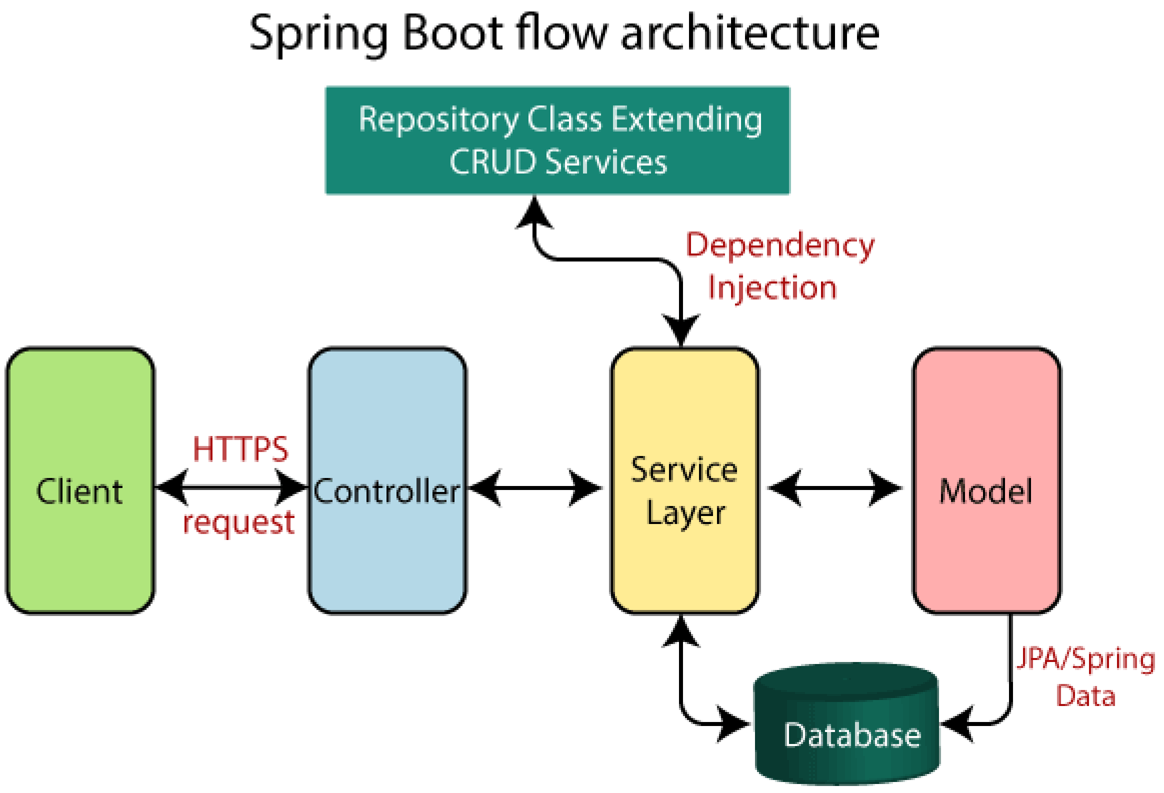
MySQL is utilized as the relational database management system for storing and managing persistent data in the Billing App. It provides a robust and scalable solution for storing call details, user information, billing records, and other application data.

D. Solution design

1. Class Diagram



2. Flow Chart Spring Architecture



E. Essential Algorithms

User Authentication and Authorization

- Utilizing Spring Security for handling user authentication and role-based authorization.

```
@PostMapping("/register/save")
public String registration(@Valid @ModelAttribute("user") UserDto
userDto,
                           BindingResult result,
                           Model model) {
    User existingUser =
userService.findUserByEmail(userDto.getEmail());

    if (existingUser != null && existingUser.getEmail() != null
&& !existingUser.getEmail().isEmpty()) {
        result.rejectValue("email", null,
                "There is already an account registered with the
same email");
    }

    if (result.hasErrors()) {
        model.addAttribute("user", userDto);
        return "/register";
    }
    userService.saveUser(userDto);
    return "redirect:/register?success";
}
```

- CustomUserDetailsService loads user details and maps roles to authorities.

```
@Override
public void saveUser(UserDto userDto) {
    User user = new User();
    user.setName(userDto.getFirstName() + " " +
userDto.getLastName());
    user.setEmail(userDto.getEmail());
    //encrypt the password using spring security

user.setPassword(passwordEncoder.encode(userDto.getPassword()));

    Role role = roleRepository.findByName("ROLE_ADMIN");
    if (role == null) {
        role = checkRoleExist();
    }
    user.setRoles(List.of(role));
    userRepository.save(user);
}

private Role checkRoleExist() {
    Role role = new Role();
    role.setName("ROLE_ADMIN");
    return roleRepository.save(role);
}
```

Call Data Recording

- Methods in SerialServiceImpl handle the logic for saving and retrieving call data.

```
// Method to process and save call record data
private void processCallRecord(String line) {
    String[] parts = line.split("\\s+");
    if (parts.length >= 6) {
        String callerNumber = parts[0];
        String lineNumber = parts[1];
        String numberDialed = parts[2];
        String date = parts[3];
        String duration = parts[5];

        SerialData serialData = new SerialData();
        serialData.setCaller(callerNumber);
        serialData.setLineNo(lineNumber);
        serialData.setNumberDialed(numberDialed);
        serialData.setDate(date);
        serialData.setDuration(duration);

        try {
            serialDataRepository.save(serialData);
            logger.info("Data saved successfully: {}", serialData);
        } catch (Exception e) {
            logger.error("Error saving data: {}", serialData, e);
        }
    } else {
        logger.error("Invalid call record format: {}", line);
    }
}
```

- The SerialController interfaces with HTTP endpoints to manage call data operations.

```
@RestController
@RequestMapping("/api/serial")
public class SerialController {

    @Autowired
    private SerialService serialService;

    @GetMapping("/data")
    public ResponseEntity<List<SerialData>> getAllSerialData() {
        List<SerialData> serialDataList =
serialService.getAllSerialData();
        return ResponseEntity.ok(serialDataList);
    }

    @GetMapping("/data/{id}")
    public ResponseEntity<SerialData>
getSerialDataById(@PathVariable Long id) {
        SerialData serialData = serialService.getSerialDataById(id);
        return ResponseEntity.ok(serialData);
    }

    @PostMapping("/data")
    public ResponseEntity<SerialData> saveSerialData(@RequestBody
SerialData serialData) {
        SerialData savedSerialData =
serialService.saveSerialData(serialData);
        return ResponseEntity.ok(savedSerialData);
    }

    @DeleteMapping("/data/{id}")
    public ResponseEntity<Void> deleteSerialData(@PathVariable Long
id) {
        serialService.deleteSerialData(id);
        return ResponseEntity.noContent().build();
    }
}
```

Billing Calculation

- The billing algorithm processes recorded call data to calculate the cost based on duration and rates.

```
          if (!durationMatch) {
            console.error("Invalid duration format:", durationText);
            alert("Invalid duration format. Please check the
duration format (e.g., 00:00'06\").");
            return;
          }

          const [, hours, minutes, seconds] =
durationMatch.map(Number);
          const durationInSeconds = hours * 3600 + minutes * 60 +
seconds;

          const companyName = prompt('Enter company name to load
settings:');
          if (!companyName) {
            alert('Company name is required.');
            return;
          }
```

- Generates billing reports that summarize the call data for each user.

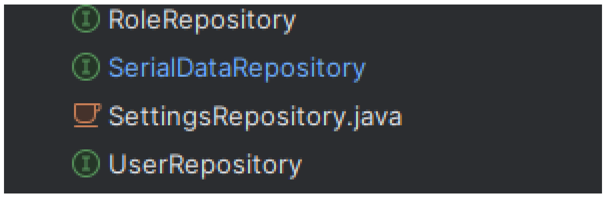
```
          const doc = new jsPDF();
          doc.text(`Bill for ${companyName}`, 10, 10);
          doc.text(`Caller: ${billData.caller}`, 10, 20);
          doc.text(`Line No: ${billData.lineNo}`, 10, 30);
          doc.text(`Number Dialed: ${billData.numberDialed}`, 10,
40);
          doc.text(`Date: ${billData.date}`, 10, 50);
          doc.text(`Duration: ${billData.duration}`, 10, 60);
          doc.text(`Cost: $${billData.cost.toFixed(2)}`, 10, 70);

          // Download the generated PDF
          doc.save(`bill_${companyName}_${Date.now()}.pdf`);
```

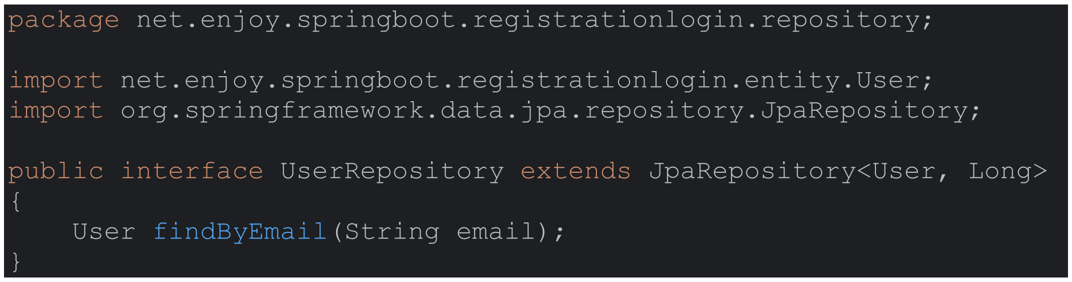
Data Management

- UserServiceImpl and SerialServiceImpl handle CRUD operations for user and call data as seen above.

- Repositories interface with the database to persist and retrieve data.



- Example for UserRepository:



```
package net.enjoy.springboot.registrationlogin.repository;

import net.enjoy.springboot.registrationlogin.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long>
{
    User findByEmail(String email);
}
```

F. Lesson Learned

Self-Reflection

During the development journey of the Billing App for Transtel Communications PABX TDS 600, I faced the dual challenges of sparse documentation for older PABX technology and the consequences of procrastination. The PABX TDS 600 system, being an older technology, lacked comprehensive documentation and established best practices, requiring me to rely extensively on trial and error to navigate its integration intricacies effectively.

Procrastination, unfortunately, compounded these challenges. Delaying tasks hindered the exploration of alternative integration methods and innovative solutions that could have streamlined development and enhanced the application's functionality. This experience served as a stark reminder of the importance of disciplined time management and proactive planning in software development. It highlighted how timely action and consistent effort are essential in overcoming hurdles, especially when working with less-documented or legacy technologies.

Moving forward, I have learned valuable lessons from these experiences. I am committed to prioritizing proactive planning and disciplined execution in all my projects, aiming to minimize the impact of unforeseen challenges and maximize opportunities for innovation. By embracing a mindset of continuous learning and resilience, I am determined to refine my skills, deliver high-quality solutions, and effectively navigate the complexities of diverse technological landscapes.

Conclusion

Reflecting on the development of the Billing App for Transtel Communications TDS 600 underscores the importance of selecting appropriate technologies, adhering to best practices, and continuously improving processes. The lessons learned from this project not only contributed to the successful delivery of a secure, scalable, and user-friendly billing solution but also laid a foundation for future software development endeavors within the organization.

G. Recourses

https://fazecast.github.io/jSerialComm/

https://dev.mysql.com/doc/

https://www.youtube.com/watch?v=Nv2DERaMx-4

https://www.youtube.com/watch?v=9SGDpanrc8U&t=695s

https://docs.spring.io/spring-framework/reference/index.html