**Developing an Application named APP and Migrating and Encrypting the Migrated Data.**

*An Internship Report*

*submitted in partial fulfillment of the requirements for the*

*award of the Degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**

BY

**DIVYANSHU KUMAR**

**(BTECH/10159/21)**

Under the Supervision of

**Dr. Aminul Islam**

**(Associate Professor)**

Logo

Description automatically generated

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BIRLA INSTITUTE OF TECHNOLOGY

MESRA-835215, RANCHI

2025

### APPROVAL OF THE GUIDE

Recommended that the Project entitled **Developing an Application named APP and Migrating and Encrypting the Migrated Data.** submitted by **Divyanshu Kumar (BTECH/10159/21)** is approved by me for submission. This should be accepted as fulfilling the partial requirements for tthe award of Degree of Bachelor of Technology in **Electronics & Communication Engineering**. To the best of my knowledge, the content of this report does not form a basis for the award of any previous degree to anyone else.

**Date: Aminul Islam**

**Associate Professor**

**Department of Electronics & Communication Engineering**

**Birla Institute of Technology, Mesra**

### DECLARATION CERTIFICATE

I certify that,

1. The work contained in the report is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any other degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the report.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

**Date: Divyanshu Kumar**

**BTECH/10159/21**

**Department of Electronics & Communication Engineering**

**Birla Institute of Technology, Mesra**

# CERTIFICATE OF APPROVAL

This is to certify that the work embodied in this thesis entitled **“\_\_Developing an application named APP and migrating and encrypting the migrated data\_”**, is carried out by **Divyanshu Kumar (BTECH/10159/21)** has been approved for the degree of Bachelor of Technology of Birla Institute of Technology, Mesra, Ranchi.

Date:

Place:

**Internal Examiner External Examiner**

**(Chairman)**

**Head of Department**

#### ABSTRACT

The internship involved the development of two critical tasks combining full-stack application development and secure data processing. The first task focused on building a Spring Boot and Angular-based full-stack application for APP Auto Postage Data Migration. The application was structured using layered architecture principles, including Controller, Service, and DAO layers. MySQL was used as the database, integrated through JDBC and ORM (Hibernate), with data entities mapped using JPA annotations.

Key Spring components such as @RestController, @Service, and @Autowired were used, alongside DTOs and ModelMapper to facilitate secure and efficient data exchange. Custom exception handling using @ControllerAdvice and @ExceptionHandler, bean validation via JSR 380 annotations, and robust HTTP configuration—including CSRF handling, CORS, and JWT-based authentication—ensured application security. The frontend, built with Angular, included dynamic navigation, role-based access control, and responsive component-based UIs.

The second task involved backend logic to extract data from a MySQL database, convert it into CSV format, and securely encrypt the file using PGP encryption with public/private key pairs. JDBC was used to connect and query the database, and a Singleton pattern ensured controlled database connections. The conversion logic utilized PrintWriter and ResultSetMetaData for structured output. PGP encryption ensured data confidentiality, leveraging asymmetric key principles for secure file exchange.

Both tasks followed SOLID design principles, promoting clean, maintainable, and extensible code. The project effectively demonstrated proficiency in full-stack development, secure data handling, and scalable software architecture.

#### ACKNOWLEDGEMENT

I express my deep gratitude **to Dr. Aminul Islam**, Professor, Department of Electronics & Communication Engineering, Birla Institute of Technology, Mesra for his exemplary guidance, monitoring and constant encouragement. The help and guidance given by him from time to time shall carry me a long way in the journey on which I am about to embark.

I would like to extend my gratitude to **Broadridge Financial Solutions** for providing me the opportunity to work with them and the team for their constant support and guidance, which helped me in acquiring knowledge about the various technologies to be used for this project.

I would also like to thank **Mohit Rastogi, Sarath Gollapalli** and the entire Broadridge family for trusting my ability and providing me with this great opportunity to learn more and more about the working of the organization.

Lastly, I am very thankful to my parents, teachers, and friends for motivating and blessing me throughout this internship program to make it a success.

Date: Divyanshu Kumar

(BTECH/10159/21)

#### CONTENTS

[ABSTRACT i](#_TOC_250003)

[ACKNOWLEDGEMENT ii](#_TOC_250002)

[LIST OF FIGURES iii](#_TOC_250001)

[LIST OF TABLES iv](#_TOC_250000)

CHAPTER 1 <NAME OF THE CHAPTER> 1

<SECTION NO. (e.g., 1.1)> 1

<SUBSECTION NO. (e.g., 1.1.4)> 3

CHAPTER 2 <NAME OF THE CHAPTER> 10

**The maximum number of Chapters should be Five.**

1. **Introduction: Overview of the project**
2. **Literature Review: Background of the project**
3. **Methodology: Description of the methods or algorithms applied.**
4. **Experimental Results and Discussion: Discussion and analysis of results.**
5. **Conclusion: Summary of the work accomplished and future scopes.**

**APPENDIX A ACCEPTED / COMMUNICATED PAPERS 34**

**REFERENCES 35**

#### LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Figure 1.1 | Caption or Label of the Figure | 5 |
| Figure 2.1 | Caption or Label of the Figure | 8 |

***CHAPTER 1***

**INTRODUCTION**

**1.1** [**ABOUT**](https://www.businesswebsolutions.in/) **BROADRIDGE**

Better financial lives are made possible by **Broadridge**, a pioneer in worldwide Fintech with more than $5 billion in annual revenue, which offers the vital infrastructure that drives investing, corporate governance, and communications. We provide technology-driven solutions for our clients that promote digital transformation and assist them in surmounting current obstacles in order to profit from the future.

Banks, broker-dealers, mutual funds, and corporate issuers around the world rely on Broadridge Financial Solutions as a top supplier of investor communications and technology-driven solutions. The business was established in 2007 and has its main office in Lake Success, New York.

Solutions for investor communication, processing of securities, data and analytics, and customer interactions are all provided by Broadridge. It supports clients in lowering operational expenses, controlling risk, and improving customer relationships.

Many of the biggest financial organizations in the world are among Broadridge's clients, and the business is active in 18 different nations. The business has received awards from the Ethisphere Institute and Forbes magazine for being among the most ethical and innovative businesses in the globe.

Broadridge is a significant participant in the financial services sector overall.

**1.2 SERVICES PROVIDED BY BROADRIDGE**

**Broadridge Financial Solutions** provides a wide range of services to banks, broker-dealers, mutual funds, and corporate issuers globally. Here are some of the key services offered by Broadridge.

**Investor Communication Solutions:** Broadridge provides a comprehensive suite of investor communication solutions, including proxy services, print and digital communication, and shareholder analytics.

**Securities Processing:** Broadridge offers a range of securities processing services, including trade processing, clearance and settlement, and custody and securities lending.

**Data and Analytics:** Broadridge provides data and analytics solutions to help clients manage risk and make informed decisions. Its solutions include reference data management, corporate actions, and risk management.

**Customer Communications:** Broadridge helps clients enhance their customer relationships through a range of communication solutions, including customer statements, regulatory documents, and marketing materials.

**Digital Solutions:** Broadridge provides a range of digital solutions, including electronic voting, digital shareholder engagement, and digital account opening.

Overall, Broadridge's services enable clients to reduce costs, manage risk, and enhance customer relationships, while improving operational efficiency and regulatory compliance.

**1.3 ABOUT THE INTERNSHIP**

An internship as a software developer at Broadridge is a great opportunity for individuals who are interested in gaining experience in the financial services industry and developing their skills as a software developer. Here are some key points to consider about an internship at Broadridge:

**Responsibilities:** As an intern at Broadridge, I was involved in developing and testing software solutions for the company's clients. We also worked on projects related to software architecture, design, and implementation.

**Training and Support:** Broadridge provides interns with training and support to help them succeed in their roles. I received guidance from senior developers, participated in training programs, and had access to learning resources to help myself in developing my skills.

**Exposure to the Financial Services Industry:** As an intern at Broadridge, I had the opportunity to gain exposure to the financial services industry and learn about the company's role in providing critical infrastructure and services that enable the efficient functioning of financial markets.

**Networking Opportunities:** Broadridge provides interns with networking opportunities to meet and connect with professionals in the industry. I attended events, participated in company meetings, and collaborated with teams across different departments.

**Career Development:** An internship at Broadridge provides valuable experience and helped me to develop my skills as a software developer. It also opened opportunities for future employment within the company or in the financial services industry.

Overall, my internship as a software developer at Broadridge was a rewarding experience that provides valuable learning opportunities and exposure to the financial services industry.

***CHAPTER 2***

**LITERATURE REVIEW**

The internship project was grounded in several modern technologies and design principles. On the backend, the Spring Boot framework was used to build RESTful web services with a layered architecture, while JPA and Hibernate provided ORM capabilities to map Java objects to MySQL tables. JWT (JSON Web Token) was implemented for stateless authentication, with Spring Security configured to manage token validation and secure endpoints.

Data Transfer Objects (DTOs) and the ModelMapper library facilitated secure and structured data exchange between the frontend and backend. Input validation was enforced using Java Bean Validation (JSR 380), and robust exception handling was achieved through @ControllerAdvice and @ExceptionHandler. The frontend was developed using Angular, a component-based framework that enabled dynamic user interfaces, form handling, and role-based routing. Additionally, for the data migration task, JDBC was used to connect to the MySQL database, extract data, and convert it to CSV format.

PGP encryption was then applied using public/private key pairs to secure the exported files. Design patterns like Singleton ensured efficient resource use, and the SOLID principles were followed throughout the application to promote clean, maintainable, and scalable code.

In addition to the core technologies, the project emphasized secure and scalable software development practices. The use of the Singleton design pattern helped manage database connections efficiently, ensuring only a single instance handled all queries, which is crucial for performance in high-load environments. Cross-Origin Resource Sharing (CORS) and CSRF configurations in Spring Security allowed safe communication between the frontend and backend hosted on different domains. Angular's routing and component-based structure enhanced user experience by supporting modular development and seamless navigation. The implementation of PGP encryption with public and private keys ensured the confidentiality and integrity of sensitive data during export, aligning with best practices in secure file transmission. Together, these tools and principles formed a robust foundation for developing enterprise-grade, full-stack applications and secure data pipelines.

***CHAPTER 3***

**METHODOLOGY**

**3.1 TRAINING**

**3.1.1 LANGUAGES**

* **Logo, company name

  Description automatically generatedJAVA:** Java is a popular programming language that is strong and flexible. It is especially used to create web and mobile apps. Developers favor it because of its portability, automated memory management, and support for OOP.

OOPs (Object-Oriented Programming)is a programming paradigm that is widely used in Java. It is based on the concept of objects, which can contain data and code to manipulate that data. Java is an object-oriented language that supports the four main principles of OOPs:

* **Encapsulation:** This pertains to the practice of only exposing the information and methods required to interact with an object, while concealing the internal details of the object from the outside world. Data protection, code reuse, and maintainability are all aided by this.
* **Inheritance:** A new class can be derived from an established class using this mechanism. The new class can both add additional properties and methods while also inheriting all of the original class's. This makes it possible to arrange the code hierarchically and promotes code reuse.
* **Polymorphism:** This alludes to an object's capacity for multiple forms. This is accomplished in Java by method modification and overloading. Multiple methods can share the same name but have distinct parameters thanks to method overloading. Contrarily, method overriding enables a subclass to offer a particular implementation of a method already given by its parent class.
* **Abstraction:** By hiding an object's complexity and only displaying its required features to the user, this is meant. Java's generic classes and interfaces are used to achieve this. A class that cannot be created but can be subclassed is an abstract class. On the other hand, an interface is a group of abstract methods that can be used by any class that uses it.

In summary, OOPs in Java is a powerful way to design and organize code in a modular and reusable way. By using these principles, developers can write code that is more efficient, maintainable, and scalable.

* **PYTHON:**

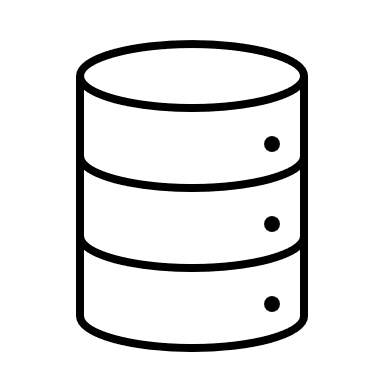
Icon

Description automatically generatedPython is renowned for being straightforward, readable, and user-friendly. It makes use of clear syntax that prioritizes readability and lowers upkeep costs. It allows a variety of paradigms for programming, such as procedural, object-oriented, and functional programming approaches. Python is also well-known for its enormous range of applications, including the creation of machine learning models using libraries like TensorFlow and Keras, the development of desktop apps using PyQT, and the creation of web applications using frameworks like Django and Flask. Python is an interpreted language, so there is no need for compilation because the code is run line by line. Beginners will find it simpler to learn and use as a result because they can concentrate on the reasoning of the code rather than the specifics of the compilation process.

* **LINUX COMMANDS:**

Linux commands are used to interact with the Linux operating system and perform various tasks. Here are some commonly used Linux commands:

* + - ls: lists all files and directories in the current directory.
    - cd: changes the current directory.
    - pwd: prints the current working directory.
    - mkdir: creates a new directory.
    - rm: removes a file or directory.

**3.1.2 DATABASE MANAGEMENT SYSTEM :**

A database management system (DBMS) is a software system that allows users to store, organize, and retrieve data from a database. A database is a group of data that has been arranged in a particular manner to make it easy to retrieve and manipulate the data. Using a query language, users can input, change, and retrieve data using a DBMS, which gives them a means of interacting with the database. Oracle, MySQL, Microsoft SQL Server, and PostgreSQL are a few prominent DBMSs. For many business applications and data-driven decision making, the use of a DBMS is crucial for handling large quantities of data effectively and securely.

Atomicity, Consistency, Isolation, and Durability are collectively referred to as ACID. It alludes to a group of characteristics that guarantee operations in a database management system are trustworthy and consistent, despite multiple users having access to the same data at once and technical difficulties. Atomicity guarantees that a transaction is treated as a single, indivisible unit of work, consistency guarantees that a transaction changes the database's state from one valid to another, isolation guarantees that concurrent transactions don't interfere with one another, and durability guarantees that once a transaction is committed, its changes are permanent and will survive any subsequent system failures.

**3.1.3 APIs & WEB SERVICES:**

Application Programming Interface is referred to as API. It is a collection of procedures, instruments, and rules for creating software programs. Developers can access the capability of one system from another system thanks to APIs, which specify how various software components should communicate with one another. In simpler words, an API can be compared to a waiter who receives diners' orders and relays them to the kitchen. After being prepared in the restaurant, the food is then returned to the waiter, who serves it to the customers. Similar to this, an API serves as a bridge between two software systems, enabling communication and the exchange of information or services.

**REST API:** REST stands for Representational State Transfer. Roy Fielding first presented RESTful APIs in the 2000s. In REST APIs the request is sent from the client side to the server and the server sends XML/ HTML/ JSON files in response. Nowadays JSON is the most popular format for file transfer. It has been the industry practice whenever it comes to creating backends. We are using Spring Boot to create a custom backend for our application.

The Guiding Principles of RESTful architecture are:

Diagram

Description automatically generated

* Uniform Interface
* Client Server
* Stateless
* Cacheable
* Layered System

**WEB SERVICES:**

Diagram, schematic

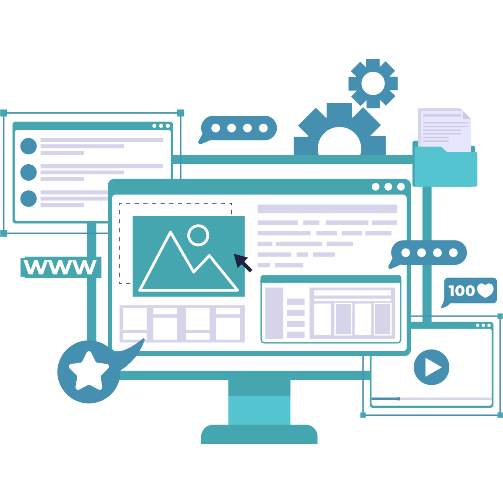
Description automatically generatedA collection of protocols and standards known as web services are used to exchange data between various software programs over the internet. Regardless of the platform or computer language employed, they enable standardized data exchange and system communication. A client submits a request to a server using a client-server architecture for web services, and the server replies with the outcome in a predefined format, such as XML or JSON. Web services come in a variety of forms, each with their own set of protocols and standards, such as SOAP, REST, and XML-RPC.

**SOFTWARE DEVELPOMENT LIFE CYCLE:**

The process used by software development teams to plan, design, develop, test, distribute, and manage software products is known as the SDLC, or software development life cycle. It is a framework that aids in making sure software tasks are finished promptly, affordably, and satisfactorily for all parties involved. Requirements gathering, analysis, design, implementation, testing, deployment, and maintenance are a few of the stages that make up the standard SDLC. The actions and deliverables for each phase differ, and the results of one phase serve as the basis for the results of the next. It gives software development teams a uniform language and technique to use, ensuring that everyone on the team is working toward the same objective.

**SCRUM:** Scrum is a rapid project management and completion technique. It is founded on a collection of ideals, rules, and procedures that place an emphasis on teamwork, self-management, and constant development. A cross-functional team works in brief periods of time called sprints to produce a working product increment according to the iterative and incremental Scrum development methodology. The team is headed by a Scrum Master, who streamlines the procedure, gets rid of roadblocks, and encourages the team to get better. The daily stand-up meetings, sprint planning, sprint review, and sprint retrospective meetings that are a part of Scrum also aid the team in maintaining alignment, concentration, and adaptability. Although Scrum is frequently used in the software industry, it can also be used in industries like marketing, education, and healthcare.

**DEVOPS:**

The goal of DevOps is to close the gap between development and operations teams and increase the efficiency and caliber of software distribution. It includes a continuous integration and delivery (CI/CD) pipeline that automates the development, testing, and deployment procedures so that programmers can reliably and rapidly release new features and updates. Additionally, DevOps places a strong emphasis on cooperation, communication, and feedback between stakeholders, including developers, operations, and other groups, allowing teams to collaborate more effectively. Configuration management, infrastructure as code, monitoring and logging, continuous testing and integration, and agile methods are some of the DevOps tools and techniques. DevOps is a key enabler of digital transformation, enabling organizations to deliver software products and services faster, better, and more cost-effectively, while increasing customer satisfaction and business agility.

**CLOUD:**

A network of distant servers is referred to as the "cloud," and it offers online, on-demand access to a variety of computing tools and services, including storage, software, and processing power. When compared to conventional on-premises IT infrastructure, cloud computing offers greater flexibility, scalability, and cost effectiveness. It has completely transformed how businesses and individuals access and manage their data and computing requirements. Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) are the three main models that can be used to classify cloud services. These models each offer users a distinct level of management and control. In recent years, the adoption of cloud computing has grown significantly, with many organizations adopting its advantages to improve their operations and competitiveness.

**3.2 AGILE METHODOLOGY:**

Shape, circle

Description automatically generated A project can be managed using the AGILE methodology by being divided into different phases. It requires ongoing user/consumer participation and continuous development at every stage. Teams cycle through planning, carrying out, and evaluation once the task begins. Collaboration between team members and project stakeholders must continue.

**BENEFITS OF AGILE:**

**1. Satisfied clients/customers**

1. After involving clients/customers during the software development journey, Agile keeps them in a loop of feedback and continuous corrections. Such iterations value their requirements and vision.
2. Keeping clients involved throughout the development life cycle of the project provides accurate feedback and ensures that the final delivery of the product will not just satisfy their needs but also exceed their expectations.

**2. Improved quality**

1. Agile methodology uses an iterative approach to project development and management i.e. the development and management processes are iterated each time a period is repeated.
2. Such consistent and repetitive focus on the development and quality cycle of the project is one of the main objectives of Agile. It certainly helps developers to build high-quality products.

**3. Adaptability**

Agile teams are responsive to continuous change, even at the last minute of the delivery. It can adapt to such last-minute changes without much fuss.

**3.3 PLATFORM SPECIFICATION:**

**HARDWARE:**

PROCESSOR: AMD RYZEN 7 PRO

RAM: 16 GB

SYSTEM TYPE: X64 BASED PROCESSOR

**SOFTWARE:**

OPERATION SYSTEM: WINDOWS 10 ENTERPRISE

IDE: INTELLIJ

TOOLS: JIRA

API PLATFORM: POSTMAN

**TECHNOLOGY:**

LANGUAGES: JAVA, LINUX, PYTHON

METHODOLOGY: AGILE

FRAMEWORKS: SPRING, JPA, REACT, ANGULAR

**TEAM ALLOTED:**

TEAM NAME: E-BILL (BRCC-DIGITAL)

TEAM DESCRIPTION: SUPPORTING THE DIGITAL COMMUNICATION IN TERMS OF VIEWING BILLS, E-MAILS AND NOTIFICATIONS AND MAKING PAYMENTS TO BILLS FOR OUR CLIENTS.

**3.4 SPRING JAVA:**

Spring Boot is an extension of the Spring framework that takes one step ahead and simplifies the configuration in order to fasten the development and execution of the spring application. It eliminates the boilerplate configuration required for a spring application. It is a module that enriches the Spring framework with Rapid Application Development (RAD) feature. It provides an easy way to create a stand-alone and production ready spring application with minimum configurations. Spring Boot provides comprehensive infrastructure support for the development and monitoring of the enterprise-ready applications based on micro services. Spring Boot is a combination of Spring framework with auto-configuration and embedded Servers.

Spring Boot provides a vast number of features and benefits. A few of them are as

follows:

1) Everything is auto-configured in Spring Boot.

2) Spring Boot starter eases the dependency management and application

configuration

3) It simplifies the application deployment by using an embedded server

4) Production ready features to monitor and manage applications, such as health

checks, metrics gathering etc.

5) Reduces the application development time and run the application

independently

6) Very easy to understand and develop Spring application

7) Increases productivity and reduces the cost.

**3.4 Spring Boot Working and Annotations :**

A class with the main method and annotated with @SpringBootApplication is the entry point of the Spring Boot application. Spring Boot auto-configures all required configurations. It performs auto-configuration by scanning the classes in class-path annotated with @Component or @Configuration. @SpringBootApplication annotation comprising the following three annotations with their default values-

• @EnableAutoConfiguration

• @ComponentScan

• @Configuration

Various annotations that I used are:

* **@RestController**: Making restful web services uses it. When used at the class level, this annotation enables the class to respond to client queries. The @restcontroller's combination of the @controller and @ResponseBody annotation is the primary distinction between it and the @controller.
* **@Service**: Moreover, classes use it. It informs Spring that the business logic is contained in that class.
* **@Component**: A class-level annotation, that is. A Java class can be designated as a bean using this. On the classpath, a Java class with the annotation @Component is discovered. It is taken up by the Spring Framework and set up as a Spring Bean in the application context.
* **@Controller**: An annotation at the class level is the @Controller. It is a @Component specialisation. It identifies a class as handling web requests. Web pages are frequently served using it. It typically produces a string indicating the redirection route. It is typically used with the annotation @RequestMapping.
* **@GetMapping**: It assigns the particular handler method to the HTTP GET requests. Create a web service endpoint that gets data using it. It replaces: @RequestMapping(method = RequestMethod.GET)
* **@PostMapping**: The HTTP POST requests are mapped to a particular handler function. A web service endpoint that creates is made using it. It replaces: @RequestMapping(method = RequestMethod.POST)
* **@PutMapping**: It assigns the specific handler method to the HTTP PUT requests. It is utilised to develop a web service endpoint that adds or modifies data. It replaces @RequestMapping(method = RequestMethod.PUT)
* **@DeleteMapping**: It assigns the specific handler method to the HTTP DELETE requests. It is utilised to build a web service endpoint for resource deletion. It replaces @RequestMapping(method = RequestMethod.DELETE)
* **@PatchMapping**: It assigns the specific handler method to the HTTP PATCH requests. It replaces @RequestMapping(method = RequestMethod.PATCH)
* **@RequestBody**: It is employed to link a method parameter's object with an HTTP request. Internally, the request body is converted using HTTP MessageConverters. The Spring framework links the incoming HTTP request body to a method parameter when we annotate it with the @RequestBody annotation.
* **@ResponseBody**: It connects the response body and the method return value. It instructs the Spring Boot Framework to serialize a returned object and return it in XML and JSON formats.
* **@PathVariable**: From the URL, it is utilized to extract the values. It works best for RESTful web services if the path variable is present in the URL. Several @PathVariables can be defined in a method.
* **@RequestParam**: It is employed to extract the URL's query parameters. Another name for it is a query parameter. It works best with web applications. If the URL does not contain the query parameter, it can specify default values.
* **@RequestHeader**: It is employed to obtain information about HTTP request headers. This annotation is a method parameter that we use. The annotation's optional components are name, required, value, and defaultValue. We should specify distinct annotations for each element in the header. Several instances of it are possible in the method @RestController: The annotations @Controller and @ResponseBody can be combined to form this. The @ResponseBody annotation is attached to the @RestController annotation. It does away with the requirement to annotate each method with @ResponseBody.
* **@RequestAttribute**: It links the request property to a method argument. It gives a controller method easy access to the request attributes. We may access objects that are filled out on the server-side with the help of the @RequestAttribute annotation.

**3.5 Spring Framework:**

Spring Framework includes technologies like: Aspect-oriented programming (AOP) Dependency injection (DI) Plain Old Java Object (POJO).

* IOC Container:

One of the fundamental components of Spring is the IoC container, which offers an efficient method for configuring and managing Java objects. The lifespan of a specified Java object is managed by this container, greatly enhancing the configurability of a Spring-based application. IoC provides the object reference at runtime using dependency injection or dependency lookup patterns. To make these tasks easier, Spring offers the org.springframework.beans and org.springframework.context packages.

* AOP Programming(Aspect-oriented):

AOP provides a different technique to structure the program than object-oriented programming, whose modularity is built on classes. An aspect serves as the primary modularity unit in AOP (cross-cutting concern). This makes it possible for users to design unique characteristics using AOP. AOP separates the program logic into separate components (called concerns). With the utilization of intersecting issues, it helps to promote modularity. As far as possible, a cross-cutting concern should be centralized in one place in the code. Examples include transaction management, authentication, logging, security, etc.

* Spring MVC:

With the aid of DispatcherServlet, Spring MVC offers a beautiful way for using MVC within the spring framework. A class called DispatcherServlet is used in this instance to map incoming requests to the appropriate controllers, models, and views. Client --request--> Web.xml --requests--> Front-Controller(DispatcherServlet) --request--> @Controllers(sends back model & view name) (...-servlet.xml).

* Model: The application's data is contained in a model. A data set might consist of a single object or a group of objects. Controller: A controller houses an application's business logic. The class is designated as the controller in this instance using the @Controller annotation. View: A view displays the information given in a specific format. Typically, a view page is made using JSP+JSTL. Besides from Thymeleaf, FreeMarker, and Apache Velocity, spring also supports various view technologies. Front Controller - The DispatcherServlet class serves as the front controller in Spring Web MVC. It is in charge of controlling the Spring MVC application's flow.
* Flow: The DispatcherServlet, which serves as the front controller, intercepts every incoming request. The request is forwarded to the controller by the dispatcherservlet after receiving an entry of handler mapping from the XML file. The controller gives back a ModelAndView object. The specified view component is called when the DispatcherServlet has checked the entry for the view resolver in the XML file. The spring jar files should be loaded, or in the case of Maven, dependencies should be included before creating the controller class. In the web.xml file, include a controller entry. Place the definition of the bean in a different XML file. The message will appear on the JSP page. Launch the server, then launch the project.

**3.6 Spring Boot Starter:**

Development of a web application using Spring MVC will require identifying all the required dependencies, compatible versions and how to connect them together. Dependency Injection is one of the key aspects of a spring boot application. For all the dependencies we need to choose the compatible version dependencies.

Various dependencies that I used:

* ***spring-boot-starter-web***: It is utilised to create web applications, especially Spring MVC-based RESTful apps. Tomcat is employed as the standard embedded container.
* ***spring-boot-starter-jdbc***: With the Tomcat JDBC connection pool, it is used for JDBC.
* **spring-boot-startervalidation**: It is utilised with the Hibernate Validator for Java Bean Validation.
* ***spring-boot-starter-security***: For Spring Security, it's employed. spring-boot-starter-data-jpa With Hibernate, it is utilised for Spring Data JPA.
* ***spring-boot-starter***: It serves as a foundation beginning and supports auto-configuration, logging, etc. spring-boot-starter-test It is utilised to test Spring Boot applications using JUnit, Hamcrest, and Mockito frameworks.
* ***spring-boot-starterthymeleaf***: It is used to build MVC web applications using Thymeleaf views.
* ***mysql-connector-java***: With the help of MySQL Connector/J, client applications written in the Java programming language can connect to MySQL. The Java Database Connectivity (JDBC) API is implemented by Connector/J, along with a variety of value-adding enhancements.
* ***spring-boot-starter-data-jpa***: This dependency is offered by Spring Boot to effectively connect Spring applications with relational databases. The spring-boot-jpa dependency is used internally by the spring-boot-starter-data-jpa. Tables and relationships are used in the databases' design. In earlier methods (JDBC), SQL queries had to be written. We will store data from objects into tables in the JPA and vice versa. JPA, however, developed as a result of an alternative way of thinking. Earlier than JPA, ORM was the name that was more frequently used to describe these frameworks. Hibernate is referred to as the ORM framework for this reason.
* ***spring-core***: An essential Spring building piece that works with Spring Beans to enable dependency injection and IoC functionality.
* ***org.bouncycastle.bcpg-jdk15, org.buncycastle.bcprov-jdk15***: One of the most popular open source Java cryptography APIs is called Bouncy Castle. It employs a thorough collection of tools covering all core security domains, such as cryptography, public key infrastructure, digital signatures, authentication, and secure communication. For those needing a high level of assurance and compliance, there are also FIPS-certified versions of the APIs for Java.

**3.7 Spring Boot Runners:**

Spring Boot provides two runner interfaces named as ApplicationRunner and CommandLineRunner. These interfaces enable you to execute a piece of code just after the Spring Boot application is started. Both interfaces are Functional Interface. If any piece of code needs to be executed when Spring Boot Application starts, we can implement either of these functional interfaces and override the single method run.

* *Application Runner*

Spring bean which implements the ApplicationRunnerinterfaces in a Spring Boot Application, the bean gets executed when application is started.

* *Command Line Runner*

The Command LineRunner interface is also meant for the same purpose as ApplicationRunner interface. Spring bean which implements the CommandLineRunner interface in a Spring Boot Application, the bean gets executed when the application is started.

**3.8 Tasks Assigned:**

**3.8.1 1st Task:**

Create a full stack application based on Spring Boot & Angular for APP Auto Postage Data Migration according to the given requirements.

**Planning & Execution**

1. **The controller service Dao model**

Client(Browser) sends HTTP request to the server(that I am creating).

Server contains:

* Presentation Layers(Controllers)->The first facing layer is the controller layer which accepts request from the client and directs these requests to the various services that are available in the application.
* Service Layer(Services) -> The main business logic is present in the service layer(classes present in the service would be executing the main business logic. They provide service to our controllers.
* DAO/Repository Layer -> The DAO layer is in direct contact with the main repository and provides/saves data into the database. The Service layer will pass the request to DAO for saving the data.

1. **JDBC(Java database Connectivity Theory) & ORM(Object Relational Mapping)**

As the name suggests, JDBC is basically establishing connection between the Java application/module and a database. In my app I used MYSQL Database.

Parameters like: driver-className, url, username and pwd are used in order to make the connection. In MYSQL Tables we have rows & columns whereas in java world we have objects. In order to map these objects as rows & columns we make use of ORM(Object Relational Mapping). This is implemented using Hibernate.

An ORM tool makes it easier to create, manipulate, and access data. It is a programming method that associates the object with the database data.

The JDBC API is used internally by the ORM tool to communicate with the database. A Java specification called Java Persistence API (JPA) offers specific capabilities and standards to ORM tools. The JPA classes and interfaces(@Entity, @Table, @Getter, @Setter, @Id, @GeneartedValue, @ManyToMany, @JoinColumn etc…) are contained in the javax.persistence package.

1. **Various Spring Annotations**

@RestController -> Combination of @Controller along with @Responsebody annotation.

@Service -> Business Logic

@RequestMapping, @GetMapping, @PostMapping, @DeleteMapping

@Autowired -> In one class if we want to inject the object of some other class we use Autowired. It automatically creates an object for the implementation class of the interface and injects it in the variable declared.

1. **DTO(Data Transfer Objects)**

An object that reduces the amount of method calls by transporting data between processes. In order to avoid making several calls to a remote server, it is primarily used to transfer data with multiple attributes from the client to the server in one go. DTOs on Java-based RESTful APIs (and Spring Boot) have the added benefit of helping to mask implementation details of domain objects (JPA entities). If we do not carefully control what properties can be altered by what activities, exposing entities through endpoints could become a security problem.

Model Mapper determines how an object model maps to another. In my application I have used this concept in Users entities.

1. **Custom Exception Handling**

To catch exceptions and convert them into HTTP Responses, several @RestController components share global @RestControllerAdvice methods.

Which type of exception we wish to handle is indicated by @ExceptionHandler.

Using method parameters, the request and the exception instance will be injected.

By using these 2 annotations:

* Control the body of the response along with status code.
* Handle several exceptions in the same class.

1. **Validation**

Java Bean is validated using JSR380(a specification for the Java API for bean validation). Properties of Bean must meet a criteria Annotation like @Notnull, @Min, @Size etc. are used.

Steps Involved:

=> Adding dependency

=> Apply annotations in your bean

=> Enable the annotations from controller.

1. **Configure Http Requests**

We use @Configuration, @EnableWebSecurity, @EnableGlobalMethodSedcurity(prePostEnabled = true

CORS essentially mandates that the server of domain B expressly approve every request made from domain A. In other words a request from any 2 domains is blocked by default. To enable requests coming from domain A, we need to set the access control policy on domain B. Spring adds a CSRF token to each produced view to secure MVC applications. On each HTTP request that alters state, the server must receive this token (post,put,delete,patch etc..). Due to the fact that an attacker cannot obtain the token from his own page, this safeguards our application from CSRF assaults.

Csrf.disable() -> Since, I have used another token mechanism.

Adding Filter Before (jwtAuthenticationFilter) -> With every request we make to the server, we need to pass a bearer token in authorization section. Firstly, we will be extracting the token and getting the username assossiated with the token with the help of getSubject. After that, we get the details of the logged in user and validate the token by comparing the logged in username & username that we get from the token.

1. **JWT Token:**

**Overview:**

* Implementing JWT(JSON Web Token)
* Must be used to secure REST APIs(Communication between client & server securely)
* It is a stateless authentication mechanism.(way to verify users by having much of session information such as user properties. It is also termed as token based authentication)
* JWT is composed of 3 parts: Header->Algorithm + token type Payload(Data about the user verify Signature->Encoded header)+Encoded payload+key
* Client =(request)(Sends Data)=> Server(Spring Boot Rest API)(validates) Server sends a token back which needs to be stored and used for access to various APIs.(Token which is generated during login is used during the entire session)

**Execution:**

Steps Involved:

* Injecting the dependancy(io.jsonwebtoken)
* Create JWTAuthenticationEntryPoint which implements Authentication entry point.
* Create JWTTokenHelper(Generation & Token Validator)
* JwtAuthenticationFilter extends OnceRequestFilter(We might want to make sure that each request only uses a certain filter once. Working with Spring Security is a frequent use case. We might want certain of the authentication processes to take place just once for the request as it passes through the filter chain. In these circumstances, we can expand the OncePerRequestFilter. The OncePerRequestFilter will only be used once for a certain request, according to Spring.)
* Get the jwt token from request. Validate token Get user from token. Load user assossiated with the token. Set spring security(setting authentication)
* Create JwtAuthResponse(store the response)
* Configure JWT in spring security config.
* Create login api to return token.

**Token Creation:**

The HTTP "Authorization" header contains a single string called a "Bearer Token" that serves as the API request's authentication. As bearer tokens do not require each request to be cryptographically signed, they are a far more straightforward method of performing API queries.

Bearer Token-> This one string, which is submitted in an HTTP "Authorization" header, serves as the API request's authentication. As bearer tokens do not require each request to be cryptographically signed, they are a far more straightforward method of performing API queries.

During the login process , token needs to be created by setting Claims, subject as Username, IssuedAt Date, Expiration Date, signWith (Algorithm (hs512) & SecretKey) and then compaction of the JWT to a URL safe string.

1. **Frontend**
   * Creating an angular Project
   * Creating and designing Components:
     + Login & Register Page
     + Toggling Side-nav & Navbar
     + Dashboard
     + Various Forms & Tables linked with Sub-Menus
   * Developing a Global Service File on Angular
   * Making use of Local Storage for various purposes.
   * Handling Routes & Sub Menus from the Backend based on User Role.
   * Using various directives in order to bind and maintain the flow of data.

**3.8.2 2nd Task:**

I was assigned the task of creating the logic for format conversion and PGP Encryption.

I was provided with a MySQL Database through which I needed to migrate data into a CSV File.

After the migration, I needed to encrypt the data using PGP Encryption(by means of PUBLIC & PRIVATE keys) and store it in a PGP File.

**Planning & Execution :**

1. **Connecting to database using JDBC(Java Database Connectivity):**

Connection needs to be established using: DriverManager.getConnection(url,username,pwd)

 Below is an example of making connections to database through different means:

1. **Singleton Design Pattern:**

The Singleton Pattern describes a class that only has one instance in the JVM and offers a universal point of access. As the database can only accept a certain amount of connections, this is done. We need a static class member, a private function Object() { [native code] }, and a static factory function in order to create a singleton class.

* Static member(shared by all the objects, only 1 copy exists): Because it is static, it only receives memory once, containing a Singleton class instance.
* Private constructor: The Singleton class will not be able to be instantiated from outside the class.
* Static factory method: This gives the caller access to the Singleton object from anywhere in the program and returns the instance.

There are 2 things:

* Mutual Exclusion: It implies that a block of code may only be run by a single thread or process at once.
* Visibility: Other threads can see changes made to shared data by one thread.
* Synchronized: Both are guaranteed by this keyword. Just one thread can access a block of threads that edit a shared variable if we synchronise the blocks so that changes performed by that thread are reflected in the main memory. Any other threads will be blocked and put to sleep if they attempt to enter the block concurrently.

Note: In some cases we may desire visibility and not atomicity. The use of synchronized in those cases may overkill and cause scalability problems. Volatile variables have the visibility features of synchronized but not the atomicity feature. The values of the volatile variable will never be cached and all writes and reads to and from the same memory.

1. **After establishing the connection, SQL query needs to be executed:**

* PreparedStatement stmt = conn.prepareStatement(query)
* ResultSet rs = stmt.executeQuery();
* resultToCsv(rs,outputFile)

1. **ResultToCsv function:**

* PrintWriter: I utilised the PrintWriter class, which is found in the java.io package and used to print data in a format that is often legible. It broadens the Writer abstract class. In order to publish data to the output file that is sent into the function, we established a PrintWriter.
* .getMataData: It gives information from the data. I made use of the ResultSetMetaData interface which provided me with the no of columns, Column Names, Data Headers etc.. and I was able to format my csv file.

1. **Concepts of Public Key & Private Key in PGP Encryption**

The public/private key pair encryption scheme determines how usable PGP is. Each PGP user has a private key and a public key. The key that other users use to encrypt messages that only you can decrypt is known as a public key. A private key can be used to create file and message signatures and is the key that enables you to decrypt messages that have been encrypted using your public key and supplied to you. A private key should never, ever be shared, while a public key can.

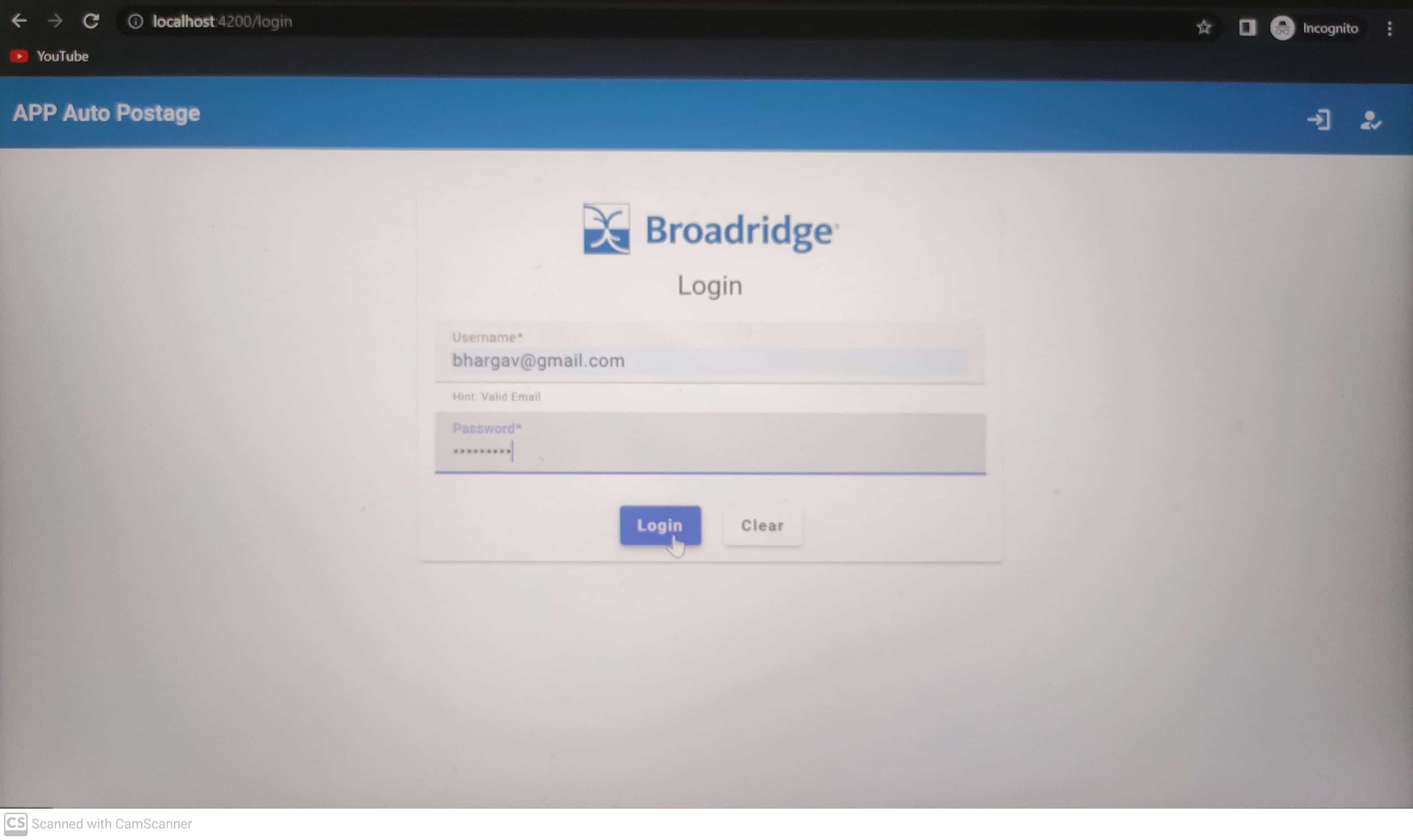
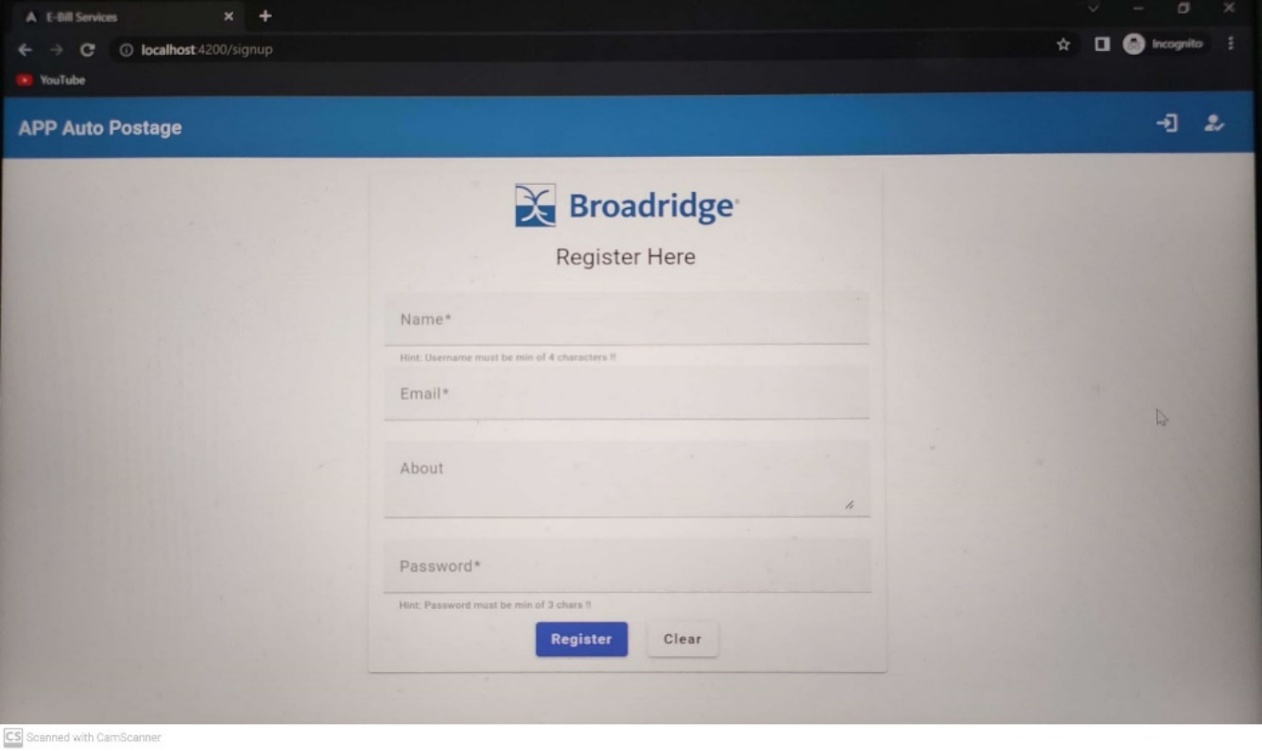
1. **Solid Design Principles:**

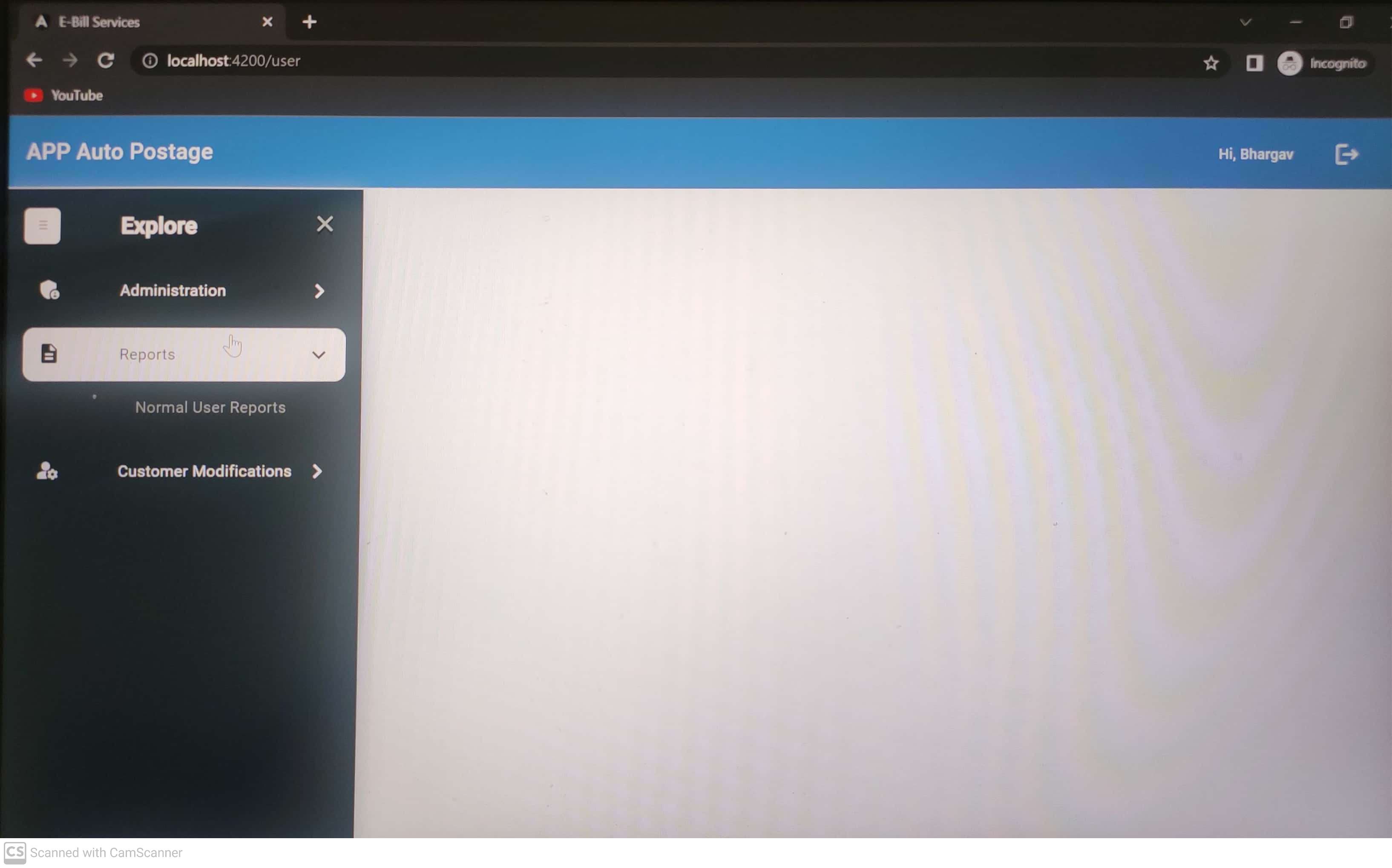
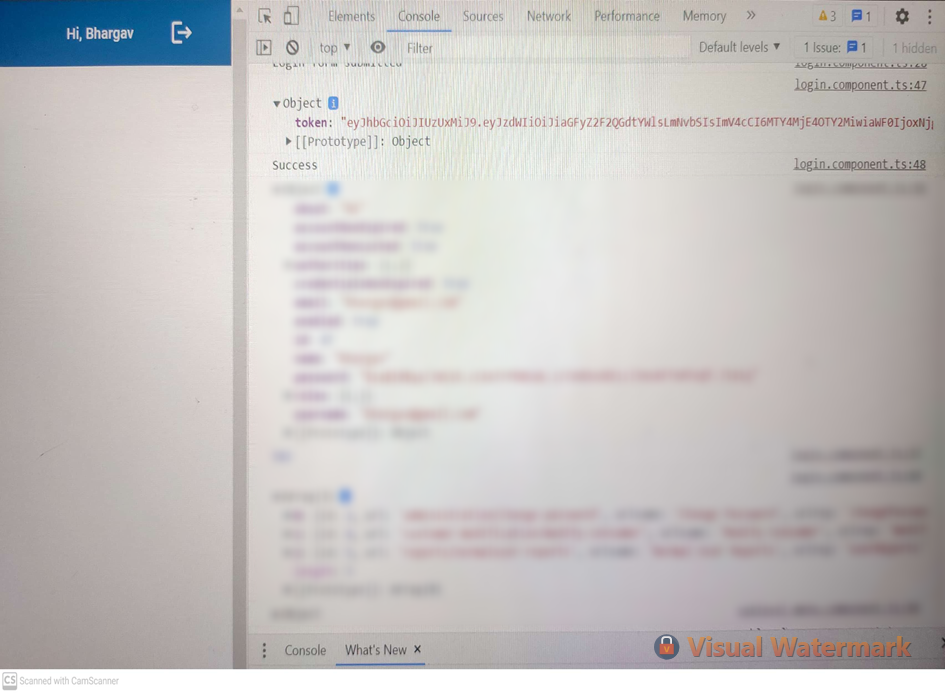
* Principle of a Single Responsibility (SRP): According to the single responsibility principle, each Java class must carry out just one function. The implementation of numerous functionalities in one class mashes up the code, and if any changes are needed, they may have an impact on the entire class.
* Open-Closed Principle (OCP): The module should be open for expansion but closed for modification in response to fresh requirements, per the open-closed paradigm.
* Principle of Liskov Substitution (LSP): The derived classes must be fully interchangeable with their source classes when it comes to inheritance. In other words, if class A is a subtype of class B, we should be able to substitute A for B without having the programme behave differently.
* Principle of Interface Isolation (ISP): The bigger interfaces break into smaller ones, according to the idea. because only the necessary methods are used by the implementation classes. The client should not be coerced into using techniques they do not want to utilise.
* Principle of Dependency Inversion (DIP): According to the idea, abstraction (abstract classes and interfaces) should be used instead of specific implementations.

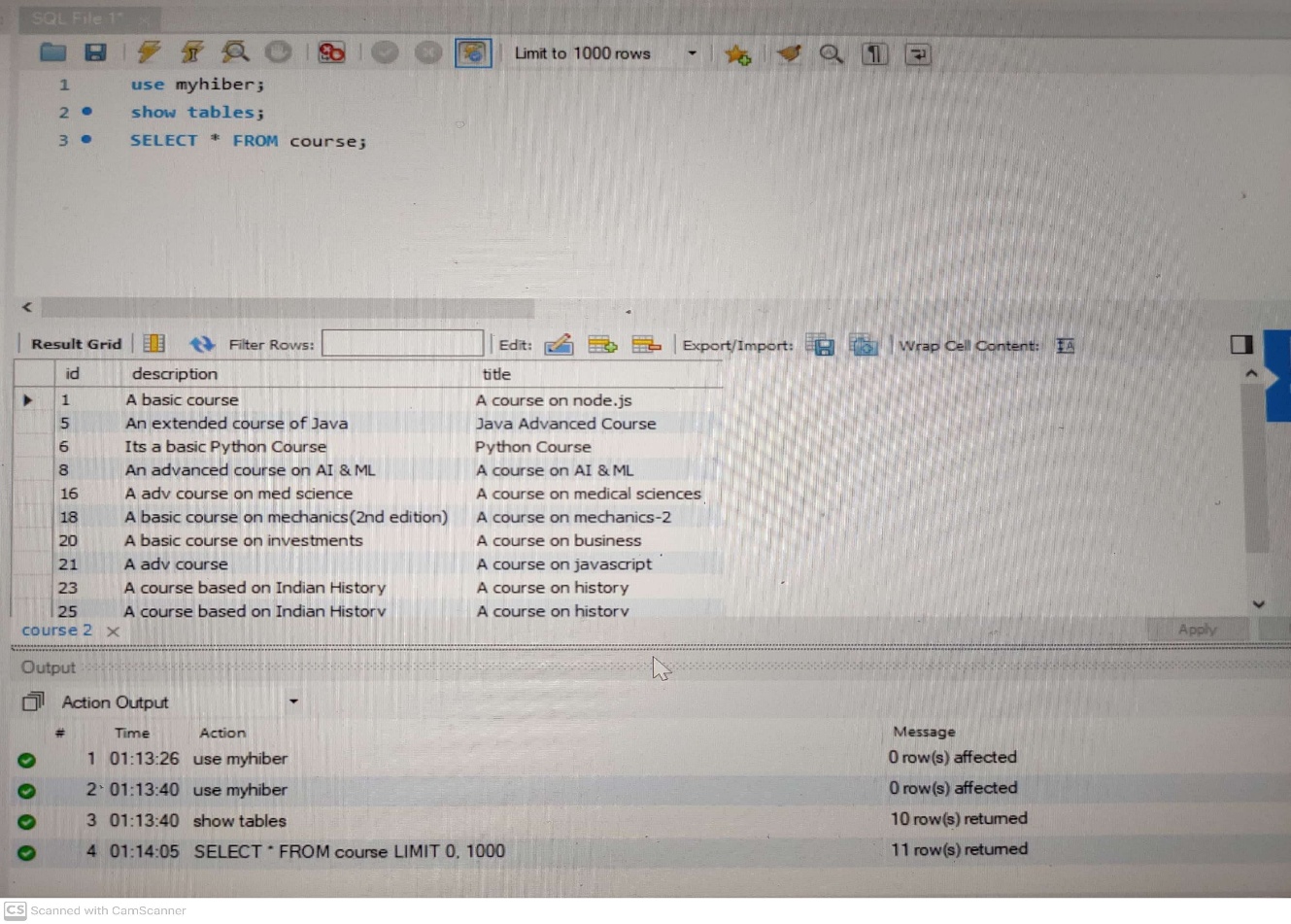
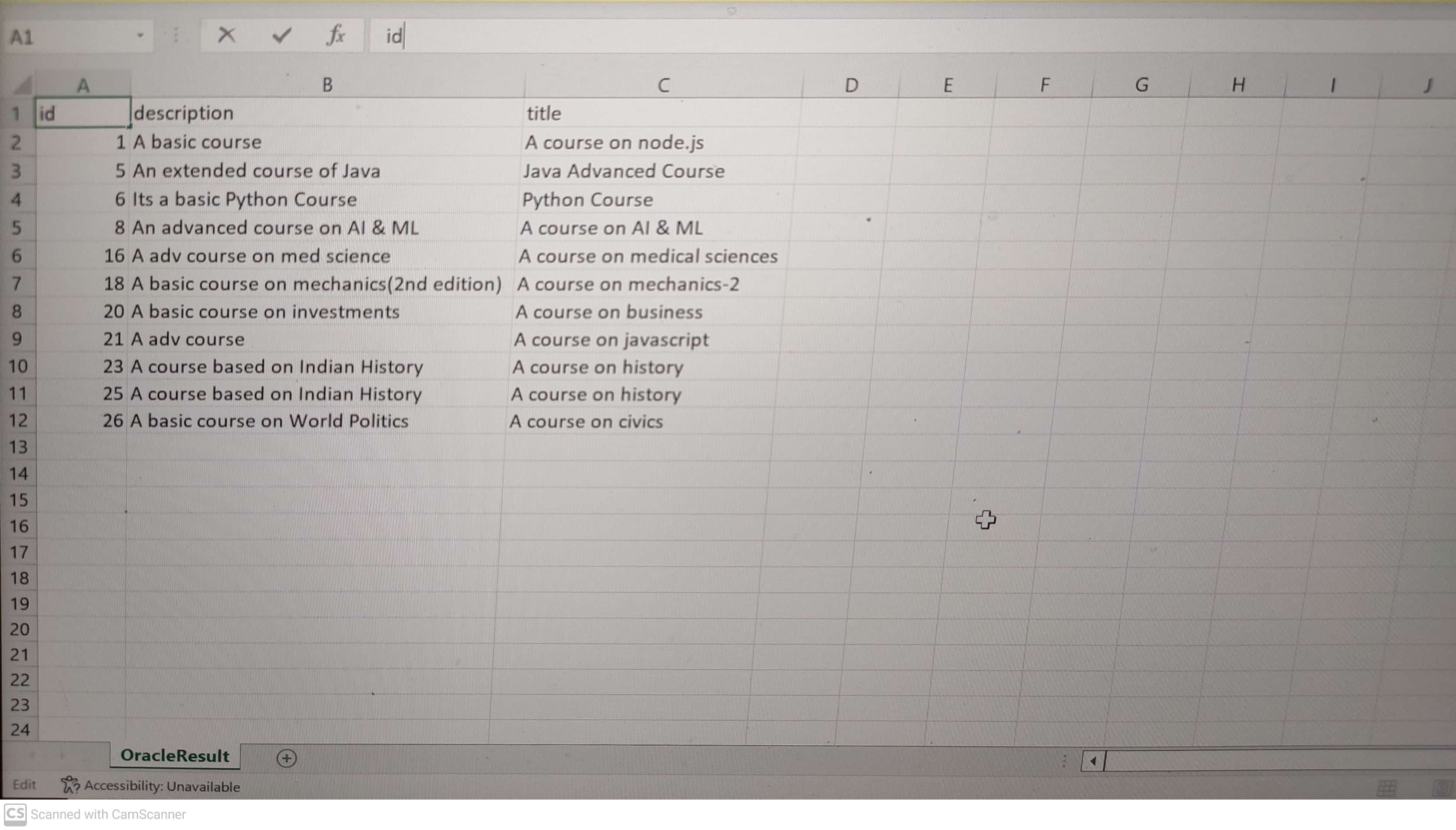
***CHAPTER 4***

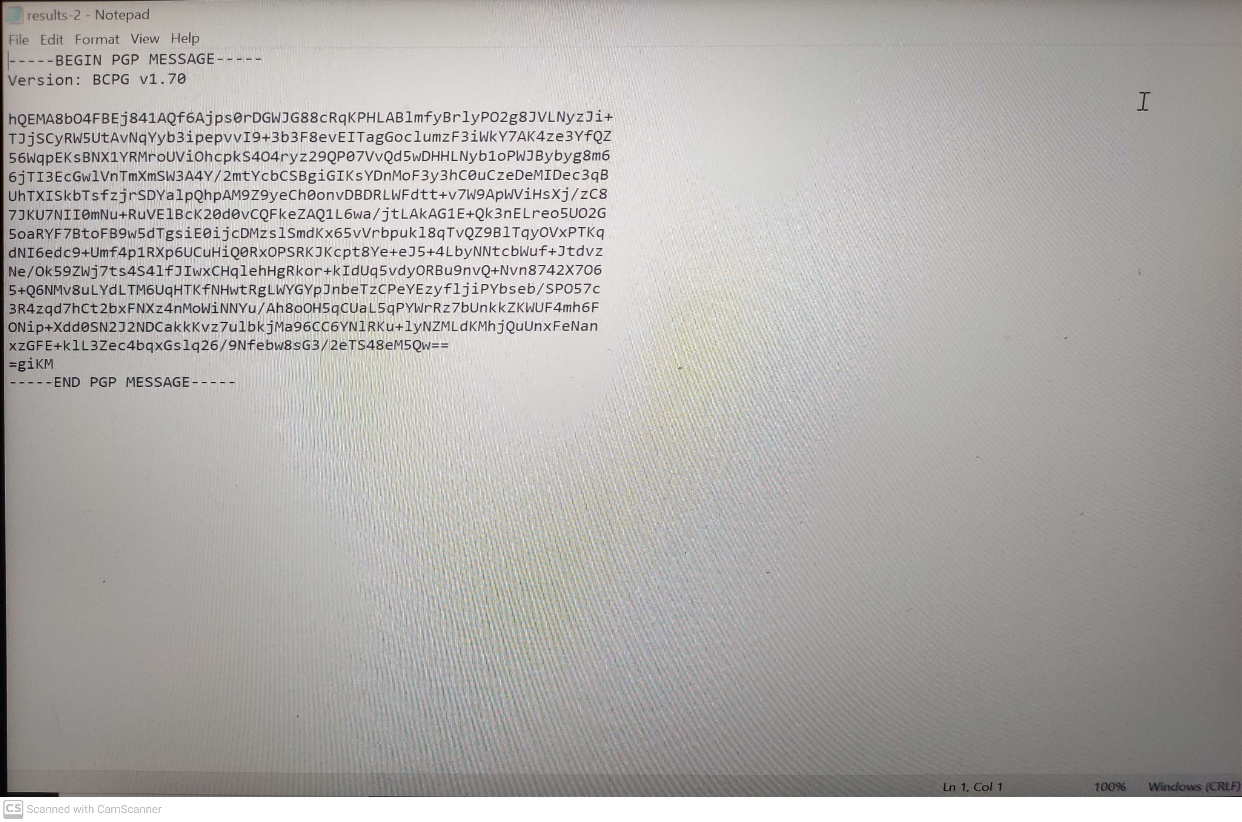
**EXPERIMENTAL RESULTS AND DISCUSSION**

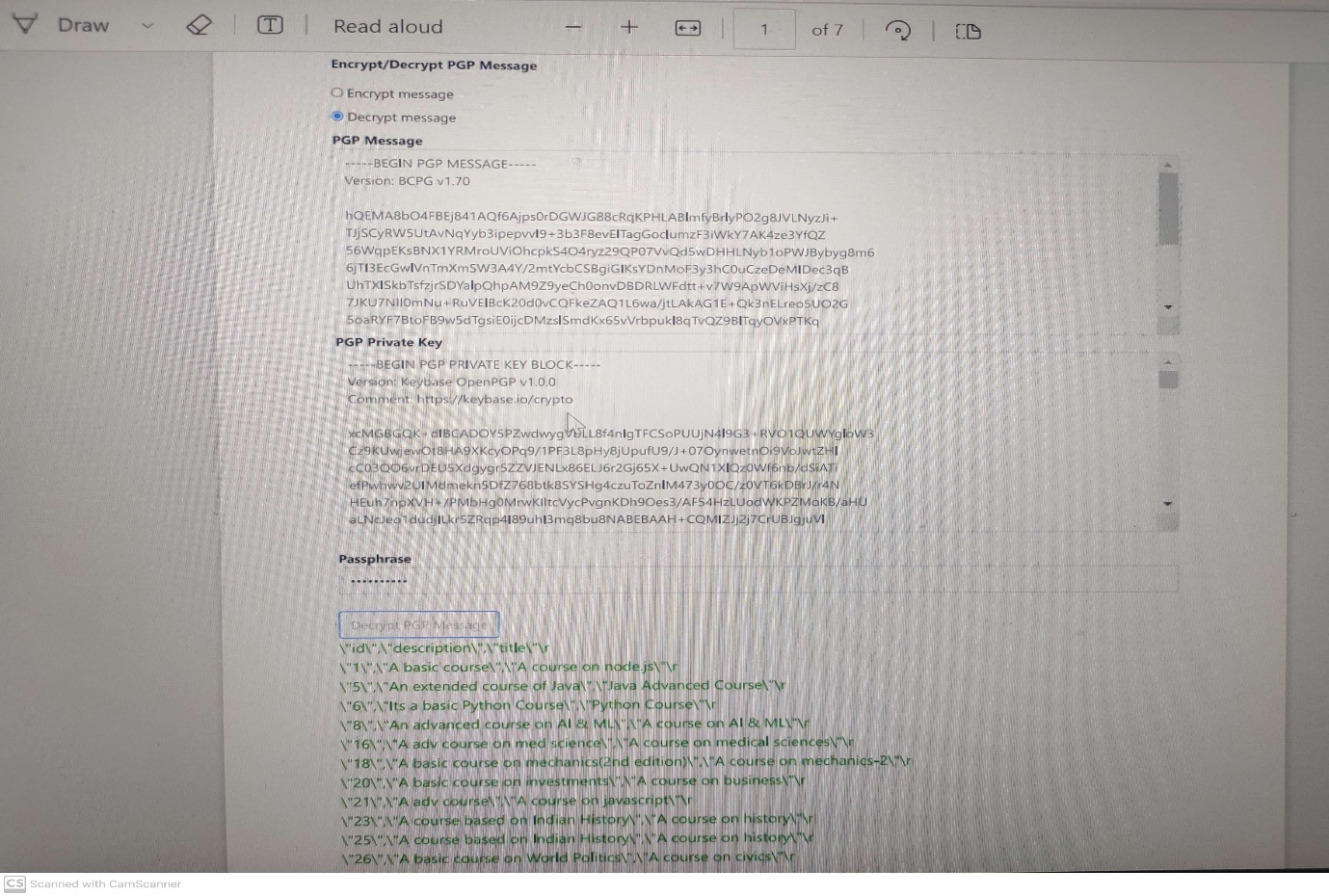
**4.1 1st Task Results :**

****



**4.2 2nd Task Results :**

****



***CHAPTER 5***

**CONCLUSION**

The frontend development of the application is actively in progress, with efforts focused on designing a user-friendly interface using Angular components, forms, and dynamic routing based on user roles. The goal is to create a seamless and responsive experience for end-users while ensuring that all functionalities integrate effectively with the backend services.

On the data management side, records from the source systems are being migrated to the central MySQL database on a daily basis. This regular data ingestion ensures that the most up-to-date information is available for processing, reporting, and future use by the application.

Following the completion of the migration phase, the development team plans to initiate the creation of backend APIs using Spring Boot. These APIs will allow clients to access their annual financial reports and other related data in real-time through secure endpoints. Furthermore, additional APIs will be implemented to manage and deliver various services such as data retrieval, filtering, transaction handling, and report generation.

To ensure that all transactions and communications between the client and server remain secure, the integration of Spring Security with JWT (JSON Web Token) authentication over HTTPS is being considered. This setup will help protect sensitive data in transit, enforce access control based on user roles, and prevent unauthorized access to critical resources.

Once the full stack (frontend and backend) functionalities are implemented and integrated, the entire application will be pushed to the main branch of the central repository. From there, the application will enter the Quality Assurance (QA) phase, where it will undergo rigorous testing. This includes functional, integration, security, and user acceptance testing to ensure that the application meets all specified requirements and performs reliably under expected workloads.

***REFERENCES***

[1] https://www.wikipedia.org/

[2] https://www.w3schools.com/

[3] https://www.javatpoint.com/

[4] https://www.geeksforgeeks.org/

[5] https://www.hackerrank.com/

[6] https://leetcode.com/

[7] https://www.baeldung.com/

[8] https://www.youtube.com/

[9] https://spring.io/

[10] https://www.javatpoint.com/spring-boot-tutorial