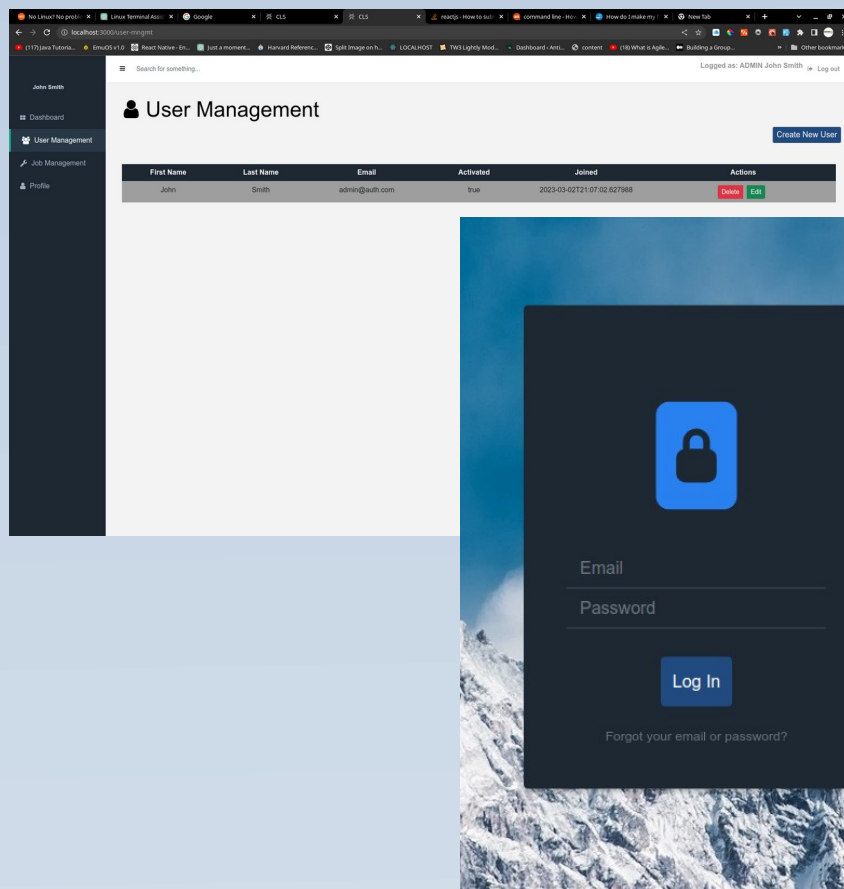




# Crew Logistic System Project Artefact Description



Name:	Arkadiusz Grudzien
ID Number	2001306
Date:	44988
Module Leader	Dr. Ela Homayounvala
First Supervisor	Victor Sowinski-Mydlarz
Second Supervisor	Sandra Fernando

**Table of Contents**

Features.....3

    Remaining Features.....3

Functionalities.....4

Technical details.....5

    Profile “dev” .....6

    Profile “deploy” .....9

Project’s structure.....9

## Features

Features which has been developed up to date:

- **Login:** Users can log in to the system by providing their email and password in the login fields provided. The system verifies the user's credentials and then grants access to the appropriate dashboard.
- **User Management:** The admin user can add new users to the system by filling out a specific form. The form collects basic user information such as name, email, and allows the admin to specify the user's role (either as an admin or a crew member). The admin can also edit or delete existing user accounts.
- **Admin Dashboard:** The admin user has access to a dashboard that displays information about the users in the system. The dashboard shows how many users have been created in total, as well as how many of those users are admin or crew members. The admin can also view a list of all the users in the system and edit or delete their accounts from the dashboard.
- **Profile:** The user can log in to the system and view their own dashboard. The profile section information such as the user's name, email, phone number, and address. The user can update their own profile information from the dashboard.
- **Job Management:** The admin user can create new job assignments for the crew members by filling out a specific form. The form collects details such as the job number, date, time, location, and the drivers and crew chief assigned to the job. The admin can also edit or delete existing job assignments.

## Remaining Features

- The program allows the admin to generate reports on various aspects of the system, such as user activity or job assignments. The reports can be customized to show specific data and can be exported to various file formats.
- Sending jobs to specific crew members

- Crew member needs to have the ability to reject or accept a job
- See accepted job
- Calendar with upcoming jobs
- Account recovery: If a user forgets their login credentials, the login feature may also include functionality to recover their account, such as password reset or account recovery options.

## Functionalities

Login has following functionalities:

**User identification:** The login feature first identifies the user by prompting them to enter their login credentials, such as a username and password. This information is then validated against the user's account information in the system's database.

**Credential verification:** The system verifies the user's credentials by checking if they match the records in the database. If the credentials are valid, the user is granted access to their account.

The authentication which is implemented in this project is JWT.

JWT stands for JSON Web Token, which is a standard for securely transmitting information between parties in the form of a JSON object.

In this system, when a user logs in, the server generates a JWT that contains the user's identity and any relevant metadata. This JWT is then signed with a secret key known only to the server. The server then sends the JWT back to the client, which stores it in local storage.

For every subsequent request that the client makes to the server, it includes the JWT in the request header. The server then verifies the JWT's signature and decodes the contents to identify the user and authorize the request.

Using JWTs has several advantages over other authentication methods, including:


- **Stateless:** The server does not need to store any session data, making it more scalable.
- **Secure:** JWTs are signed and can be encrypted, making them tamper-proof and ensuring that only authorized parties can decode the

The remaining functionality for this feature is:

Account recovery: If a user forgets their login credentials, the login feature may also include functionality to recover their account, such as password reset or account recovery options.

#### Register New User via Admin Account

The clients requests was that the user can only be created via admin account, not as usually is that the registration is available to everyone. This functionality require admin to log in to the system and go to section User Management and click on the button called “Create New User”. The admin have to fill the form which have been appeared and choose what type of user needs to be created: admin or crew member. Ones the user is created, the user receives an email to verify his email address. One this is done the user can normally log in.

 **Create New User**

Ones the user is created the system send a confirmation link.  
The user needs to click on it to activate his/her account.

Email	porters@mailinator.com
First Name	Porter
Last Name	Sellers
Role	CREW_MEMBER ▾
Phone Number	+1 (718) 675-5983
Address Line 1	10 West First Extension
Address Line 2	Id modi quisquam et
City	Velit nisi ipsum pe
State/Province	Excepturi quo quod p
Postal Code	Nesciunt iste facil
Country	Rerum dolores optio

SubmitCancel

Figure 1: Creating new user

For development purpose there has been setup email server called maildev where all emails are sent.

## Technical details

Spring Boot Profiles provide a way to configure an application for different environments such as development, testing, staging, or production. Profiles allows to define different configurations for each environment and to activate them at runtime. In Spring Boot, a profile is a set of properties and configurations that define the behaviour of the application for a specific environment.

There are two profiles that have been implemented for the software, namely the deploy and dev profiles. The dev profile, which stands for development, has been designed to facilitate making changes on the fly. To achieve this state, certain dependencies need to be installed, including:

OpenJdk version 17.0.6 (Java),  
NodeJS version 16.19.1,  
NPM 8.19 (which comes with Node),  
MySQL 8.0.32, and Apache Maven 3.6.3.

### Profile “dev”

To run the software in this profile (dev), several steps need to be followed. Needs to be sure that the profile is switched on in application.properties.

```
spring.profiles.active=dev
```

application.properties is a configuration to configure various properties of the application. It is a plain text file that is located in the src/main/resources directory of the application.

The file application.properties contains key-value pairs that configure various aspects of the Spring Boot application, such as the server port, database connection settings, logging configuration, security settings, and more. It allows to customize and configure the application without modifying the source code. In this case application.properties include common configuration for both profiles. The divided configurations are included in application-dev.properties and application-deploy.properties respectively. To switch between the profiles easily to change one value as mentioned before.

The profile “dev” has following configuration:

```
lex.js    JS Nav.js    # Style.css 1    application.properties 1, M    application-dev.properties
end > src > main > resources > application-dev.properties
You, now | 1 author (You)
# configuration for MySQL

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/cls
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# email configuration
spring.mail.host=localhost
spring.mail.port=8091
spring.mail.username=hello
spring.mail.password=hello
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
spring.mail.properties.mail.debug=true
spring.mail.properties.mail.smtp.ssl.trust=*

config-email.subject="Pinnacle Crew - Activate your account"
config-email.name="info@pinnaclecrew.co.uk"
logging.file.name=logs/cls.log

jwt.secret=HW7nsQ9prpQtvvBI9jLtIdHpiKGSrmCes

spring.security.filter.order=10

logging.level.web=DEBUGs
logging.logback.rollingpolicy.max-file-size=100MB
logging.logback.rollingpolicy.max-history=30
```

The first lines are configuring the database providing the way how the database need to behave, update that the data will be added there is also variable called create-drop. It does when spring boot reboots the data is lost usually used for testing when are created new entities.

The other line configuring the data source credentials to log in to the database.

Email configuration is for the mail server which has been implemented for developing purpose. Ones the user is created the email server receives the meail for confirm the email address of the user who is being registered.

The mailadev (<https://github.com/maildev/maildev.git>) can be run via docker with folowing configuration :



```

version: '3'

services:
  maildev:
    image: maildev/maildev
    restart: always
    environment:
      - TZ=Asia/Shanghai
      - MAILDEV_WEB_PORT=1080
      - MAILDEV_SMTP_PORT=1025
    ports:
      - "1080:1080"
      - "1025:1025"
    logging:
      driver: "json-file"
      options:
        max-size: "1m"

```

**command: docker-compose up**

To be able to run the program in mentioned profile the next step is to go to /backend folder and run a following command in terminal:

**mvn spring-boot:run**

The above command runs the Spring Boot application if all the configurations has been set up correctly following the guide above.

The next step is to run the frontend. To be able to do that need to “be” in the frontend folder and run command :

**npm install && npm start**

This command is responsible at first to download all dependences for the react framework and then it runs the web application at <http://localhost:3000>. It's important to have setup correct value in package.json:

**"proxy": "<http://localhost:8080>",**

The Spring Boot system creates a default user. The credentials are

**Email: admin@auth.com**

**Password: 1234**

## Profile “deploy”

This profile has been made for viewing the program without configuring the machine where will be run, however docker needs to be installed.

Once the docker is installed simply run following command:

**docker-compose up**

but needs to be sure that the profile is switched on in application.properties:

**spring.profiles.active=dev**

and in package.json for

**"proxy": "http://10.5.0.5:8080",**

**The delivered files should already have switched on the profile.**

The proxy value is the IP and port where the Spring Boot application runs. The frontend “knows” where to send all the requests.

## Project’s structure.

Spring Boot Backed - UML

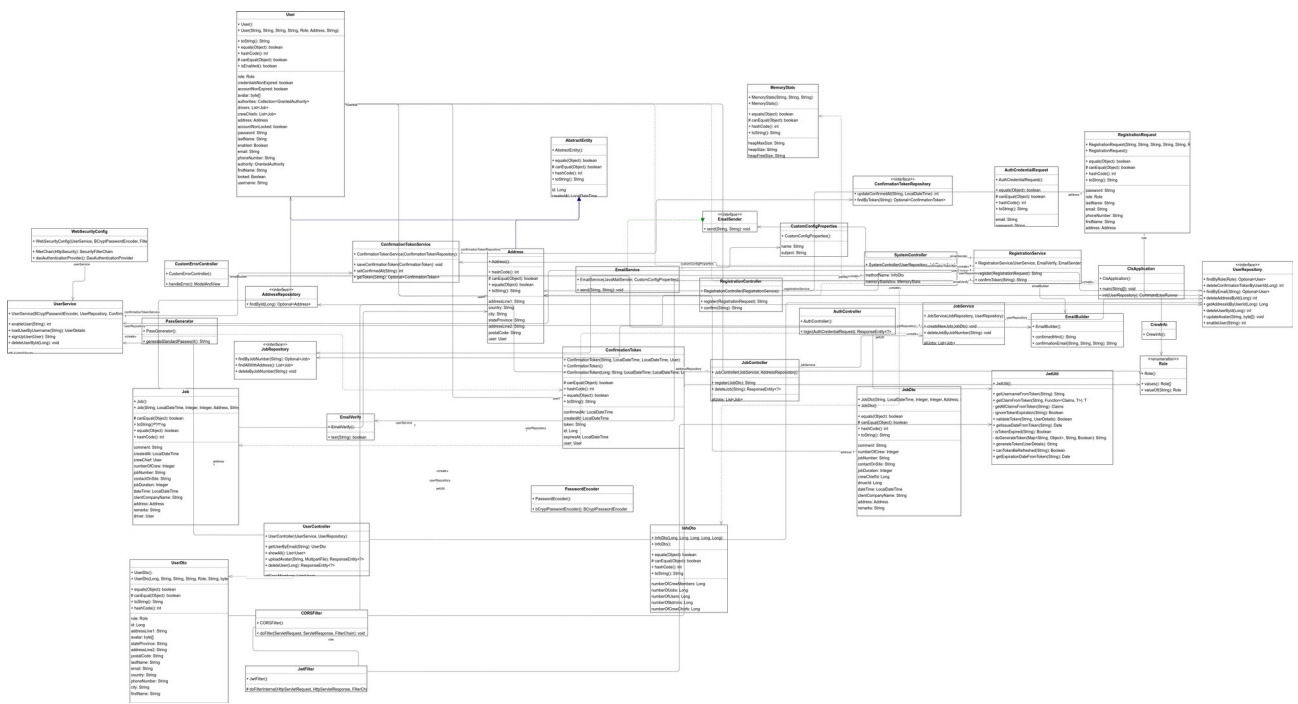


Figure 2: This show the structure of the web application - UML (can be zoomed)

The table below describes all the classes at this stage in the projects.

Class	Description
<b>Controller package</b>	
AuthController.java	<p>The AuthController is a Java class that serves as a RESTful web service controller for user authentication. It is annotated with <code>@RestController</code> to indicate that it handles HTTP requests and responses. The <code>@RequestMapping</code> annotation specifies the URL path prefix for all the endpoints in this controller. The <code>@CrossOrigin</code> annotation enables Cross-Origin Resource Sharing (CORS), which allows web applications from other domains to access this web service.</p> <p>This class has one endpoint: <code>/api/auth/login</code>, which is mapped to the HTTP POST method using the <code>@PostMapping</code> annotation. This endpoint expects an HTTP request body containing a JSON object with email and password properties.</p> <p>The login method in this class handles the <code>/api/auth/login</code> endpoint. It authenticates the user's credentials using Spring Security's <code>DaoAuthenticationProvider</code>, which is autowired into the class. If the credentials are valid, it generates a JSON Web Token (JWT) using the <code>JwtUtil</code> class, which is also autowired into the class. Finally, it returns an HTTP response containing the JWT in the Authorization header and the authenticated user's details in the response body. If the authentication fails, it returns an HTTP UNAUTHORIZED status code.</p>
AuthCredentialRequest.java	This class is probably used as a helper class to encapsulate the user's email and password in a single object, which can then be sent in the request body of an HTTP POST request to the <code>/api/auth/login</code> endpoint of the AuthController class.
CustomErrorController.java	This class handle <code>/error</code> endpoint. It allows to define a custom error page or error handling logic for all types of errors that may occur within the application.
JobController.java	This controller provides CRUD functionality for managing jobs in the API, with endpoints for creating, retrieving, and deleting jobs. The injected <code>JobService</code> and <code>AddressRepository</code> dependencies are used to provide the business logic for these operations.
SystemStatus.java	In this controller, the functionality of counting users, crew members, and admins is implemented. Additionally, a method is provided for retrieving information about the current status of the JVM, such as the heap size for developing puproses.
UserContoller.java	Is a key component of a this web application that provides functionality for user management It interacts with a user service to perform business logic related to user management and handles HTTP requests related to user authentication and profile management.
<b>DTO package</b>	

InfoDto.java JobDto.java MemoryStats.java UserDto.java	<p>DTOs are used to represent domain objects in a simplified form that is optimized for data transfer. They contain only the data that is necessary for the operation being performed, and may exclude sensitive or unnecessary data that is not required for the transfer. DTOs also provide data validation and transformation logic to ensure that data is consistent and correctly formatted.</p>
<b>Email package</b>	
EmailBuilder.java	<p>This class is designed to store HTML templates as strings, which can be used to generate emails that are sent to users of a Spring Boot application. Specifically, the HTML templates are used to generate emails that are sent to newly created users via an admin interface. The HTML templates contain a link that allows the user to confirm their email address.</p>
EmailSender.java	<p>This is an interface that defines a contract for sending emails. It specifies a single method called "send" that takes two parameters: a "to" address and an "email" message.</p>
EmailService.java	<p>This is a class that implements the EmailSender interface and provides a concrete implementation of the "send" method.</p> <p>The EmailService class uses the JavaMailSender interface provided by Spring Boot to send emails. The class has two constructor arguments: a JavaMailSender object and a CustomConfigProperties object, which are injected via constructor injection. The send method of this class is annotated with the @Async annotation, which indicates that the method should be executed asynchronously, in a separate thread. This allows the application to send emails in the background without blocking the main application thread.</p> <p>The send method creates a new MimeMessage object using the mailSender object and sets the email content and recipient using the MimeMessageHelper class. The email content is set to the "email" parameter of the method, which contains the HTML content of the email template.</p> <p>The "to" parameter of the send method represents the email address of the recipient of the email. This parameter is used to set the recipient of the email using the helper.setTo method.</p> <p>The CustomConfigProperties object is used to retrieve the subject and sender name for the email. These properties should be configured in a configuration file to allow for easy customization of the email content and sender details.</p> <p>If an exception occurs while sending the email, an error message is logged using the LOGGER object and an IllegalStateException is thrown.</p>
<b>Model package</b>	
AbstractEntity.java	<p>The AbstractEntity class defines an "id" property that represents the unique identifier of the entity. It also defines methods for getting and setting the id property, as well as methods for checking whether an entity is new (i.e. has not yet been persisted to the database) or not. It also have a filed createdAt so whenever is new entity crated it goes to the database. Not all calsses using this because not all class suing id as an integer. It helps to reduce code duplication.</p>

Address.java	<p>The Address class is an entity class that extends the AbstractEntity class. It represents a physical address with various properties such as addressLine1, addressLine2, city, stateProvince, postalCode, and country.</p> <p>The @OneToOne(mappedBy = "address") annotation is used to specify a bidirectional one-to-one relationship between the Address class and the User class. This means that each Address entity can be associated with only one User entity, and vice versa.</p> <p>The @JoinColumn(name = "user_id") annotation is used to specify the foreign key column name in the database table that maps to the User entity's primary key column.</p>
Job.java	<p>This is a class that represents a Job entity in the database. It is annotated with @Entity to indicate that it is an entity and @Table(name = "job") specifies the table name in the database. The class extends AbstractEntity, which means it inherits the properties of the abstract class.</p> <p>The class has instance variables such as jobNumber, dateTime, jobDuration, numberOfCrew, address, clientCompanyName, contactOnSite, driver, crewChief, remarks, and comment. These variables represent the attributes of a job. The class also has getters and setters for these variables.</p> <p>The class is also annotated with @JsonIdentityInfo and @JsonManagedReference to handle the JSON serialization and avoid infinite recursion issues when serializing relationships between entities. The class also uses @OneToOne and @ManyToOne annotations to specify the relationships between entities in the database schema.</p>
Role.java	<p>This is a Java enumeration representing the different roles that a user can have in the system. There are two roles defined in this enumeration: "ADMIN" and "CREW_MEMBER". These roles are used to determine the access level and permissions of users in the system. For example, an "ADMIN" user have access to more features and functionalities than a "CREW_MEMBER" user. The use of enums in this case makes it easier to manage and maintain the different roles in the system.</p>
User.java	<p>This class represents a User entity in a database. It is annotated with @Entity to indicate that it is an entity class, and @Table is used to specify the name of the table in which the entity is stored. The class extends AbstractEntity, which provides the primary key field and the equals and hashCode methods.</p> <p>The class has instance variables for email, first name, last name, password, role, address, phone number, avatar, and locked and enabled flags. It implements the UserDetails interface to provide authentication and authorization information for the user, including username (which is the email), password, authorities, and enabled status.</p> <p>The class also has relationships with other entities: One-to-one relationship with the Address entity using @OneToOne and @JoinColumn annotations, and one-to-many relationships with the Job entity as a driver or crew chief using @OneToMany annotation with the mappedBy attribute to indicate that the Job entity has the</p>

inverse relationship.

There is a constructor provided to create a new User with the required information, and getter and setter methods for all instance variables. The class also overrides several methods from the UserDetails interface to provide custom behavior.