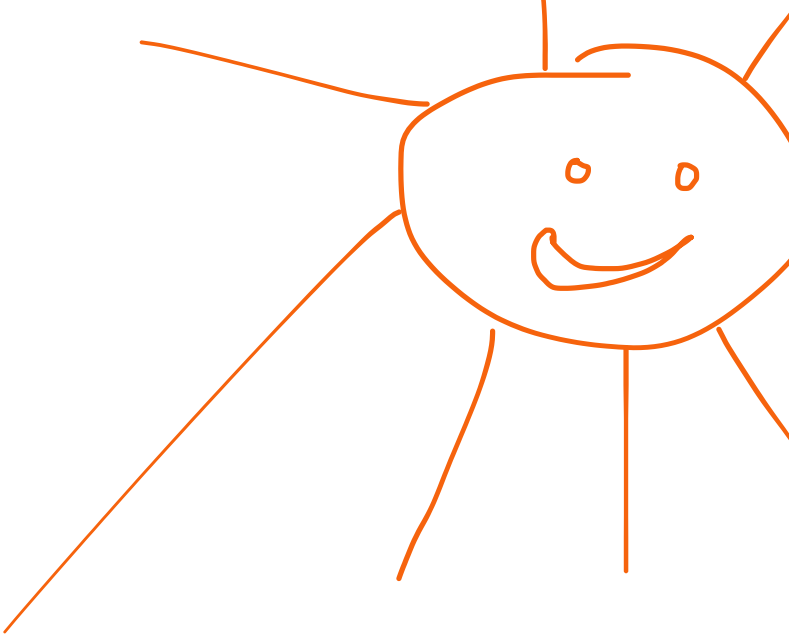
Well, this is my coursework



Developing an Online Booking System for Laundry Room Using Spring Framework

Aleksandr Timokhin, 2022

INDEX

ANALYSIS SECTION

Project Background

I have been living in a boarding house in the UK for almost two years now and I have encountered a problem: me and my fellow students couldn't make use of washing and drying machines efficiently. People had no way of figuring out whether machines are occupied or not. The only way to find that out so far was to travel through 4-5 floors with a basket of dirty clothing and see for yourself if one of the machines is vacant or not.

In order to solve it, boarding house assistants have set up a system where every floor of the house gets a separate day to do their washing. But this was also not working for me, as I didn't find the allocated day convenient.

Thus, I decided to solve this problem by designing a system that will help me and my peers to use the laundry room more efficiently.

Evidence of Analysis (Interview With End-Users)

I began by gathering some statistics on the issue from people with whom I share the boarding house. I conducted meetings with 5 students and 2 members of staff to see what the main issues with the currently-existing laundry system were. I have summarised their words into bullet-points. I didn't include any personal information those people have shared due to ethical reasons.

Students:

1) How often do you do laundry?
    a) 2 times a week on average
2) When do you do your laundry?
    a) During the dedicated day after school (from 5pm)
    b) On the weekends (Most of the respondents were doing laundry in the morning)
3) What is the problem with using the laundry room on your dedicated day?
    a) I am too late to put my things in, as some people come earlier than me to the boarding house.
    b) People leave their clothes over the school day so some machines are occupied when I arrive home.

    c) There is not enough time for the whole floor to do laundry (only about 4 hours)
4) How do you solve these difficulties?
    a) I wait until weekends to do my washings
    b) I call staff on duty to take the clothes out of the machines if machines have finished with their program
5) What is difficult with calling the staff member?
    a) There is no internet in laundry room
    b) They don't always pick up fast enough
    c) If I am gone to find the member of staff, someone might clear the machine and put other clothes in, so I lose my place in a queue.
6) What is difficult about using the machines on weekends?
    a) A lot of people do they washings on these days
    b) It is hard to know when machines are free to use
    c) I need to wash/dry my clothes few times, but there is not enough time for that
    d) Machines do not show the correct time of program termination, so I do not know if someone will take my clothes out or not.
    e) Some people occupy several machines at the same time
    f) I need to stay in the house for several hours so that I can pick up my clothes when they are done washing/drying.

Staff:

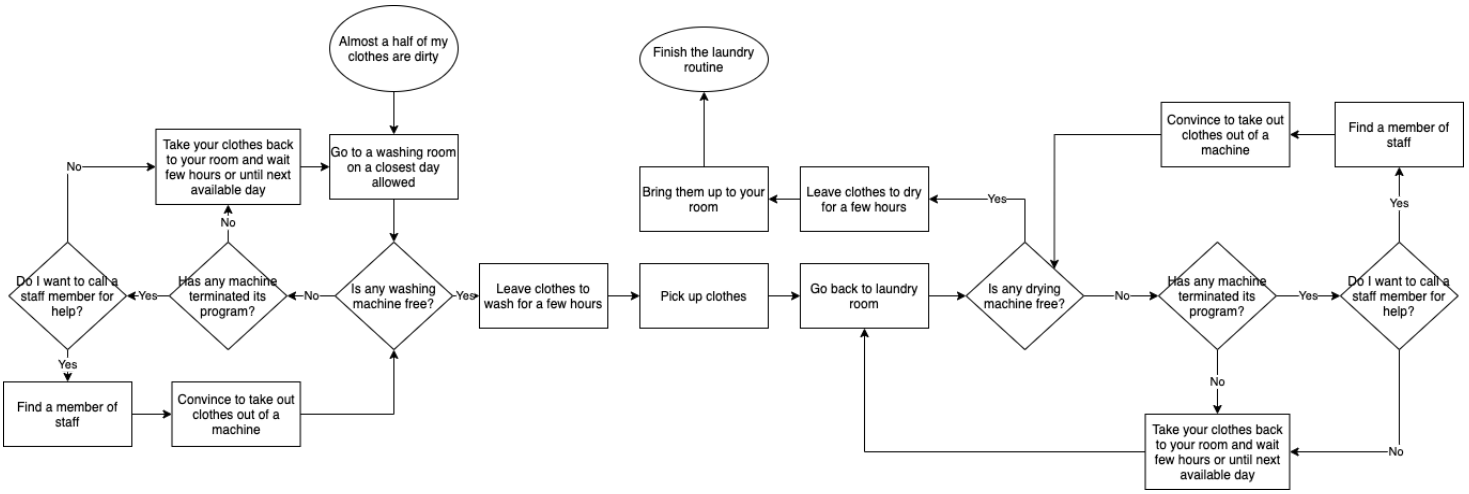
1) How often do you do laundry?
    a) Once a week we have a dedicated time when we can do the laundry.
2) Are there any problems you experience with using the laundry room?
    a) People don't always take their clothes out when they should.
3) How do you solve this difficulty?
    a) I take these clothes out.
4) Are there any difficulties with operating the current laundry system for students?
    a) On weekends I need to wake up earlier to take someone's clothes out of the machines.
    b) I need to constantly deal with people not taking their clothes out.

After conducting these interviews I have realised that I am not the only one who faces this problem.

Problem Modelling

As there are less boarding house staff than actual students I have decided to primarily focus on a solution for the students.

From the interview findings mentioned above I was able to model the context of using the laundry room and also to identify main problems that required immediate solution:



From the diagram we can spot a few problems that students face:

- A staff member is required to take out clothes (students are not allowed to do that themselves). As communication between staff and students is not that straight-foward without the internet, a bunch of problems arise.

- Students cannot control when they will wash their clothes. They might need to go through a waiting stage a few times, making the laundry process take up to a week to be completed. As more than a half of clothes require washing, that is a big issue.

From the interviews, I have identified the following problems as those that require a solution straight away:

- A lot of people do washings on the same days.

- Students have no way of knowing when they can use the machines (when they are free).

- It is hard to communicate with staff.

- People do not pick up clothes on time.

From that analysis I was able to come up with a shortlist of problems that my application must solve in order to be useful.

I have decided that an online booking system would be the most reasonable solution, as it allows to make bookings in advance and do it any time, anywhere.

Next step was to identify how to make sure that my system includes good features of other booking systems and what are the problems with the other systems.

Analysis of Existing Systems

I was not able to find online booking systems for laundry rooms. All of the solutions I found rely on monitoring when machines are available, which does not allow booking the machine at any convenient time. For instance, in Florida Institute of Technology they have cameras installed next to machine timers. Videos of these timers are streamed online and anyone can see when they can use a machine. Though this system works, it has some problems that I want to avoid:

- No one is sure whether when they walk into the room someone is not loading their clothes into the machine. That is possible because a person cannot control what is going on in the laundry room after they walk out of their room to get to the laundry room.

- People will cluster and do laundry at the same time. Obviously, some hours are more desirable for most people to use the machine (say, weekends). But that means that huges queues will be forming during rush hours. It would be much simpler if people would distribute their washings across the whole week.

With that in mind, I went to the web to search for online booking services that just provide a convenient booking.

System No. 1 - OpenTable

(https://www.opentable.com/)

Overview:

This is an online booking service to make restaurant reservations.

What I liked about the system:

+ Convenient navigation on the platform. It is easy to find a restaurant you wish to book.

+ Allows users to create accounts which prevents users from entering similar data over and over (like name, telephone number).

+ Can cancel bookings.

+ Can see if a restaurant is free on a given date (meaning that the booking works as intended).

+ Intuitive forms. That makes booking a table in a restaurant super easy.

+ The data is structured in a way that makes reading it very easy.

What I do not like about this system:

- In integrated APIs you cannot see when the restaurant is booked up. That means that you must send loads of requests.

System No. 2 - Trainline

(https://www.thetrainline.com/)

Overview:

This is a platform for booking train tickets online.

What I liked about the system:

+ Convenient navigation on the platform.

+ Allows users to create accounts which prevents users from entering similar data over and over (like name, telephone number).

+ Can cancel bookings.

+ Can compare multiple trains to see which you like more.

+ Can search for tickets with all relevant parameters (like destination, departure/arrival time, open/fixed return date, etc).

+ Have a mobile app. That means that accessing the service is very easy.

+ Can select a seat that you want. In my case, that can be a specific washing/drying machine selection.

Online booking systems of all sorts have existed since the creation of the internet. It is very possible to go online and to find such systems that are very poorly designed, and hence create a vast list of things that can be improved. However, I have done almost the same thing by identifying positives of the analysed systems.

Analysis of My System Features Based On the Previous Findings

After doing the analysis of the existing systems and gathering a list of main issues with the current laundry system at my boarding house, I was able to come up with a list of features my system must incorporate in order to be good.

Feature	What problem it aims to solve
Convenient navigation on the platform.	Makes it easier to navigate around the website
Providing each user with an account.	Allows to store data about a given user. Prevents data inconsistencies. Limits number of times user must enter a lot of data (like name, email address)
Can select which exact machine to book.	This should naturally even out the use of all machines. Plus, the user has a way to choose a machine of choice.
Provide users with an interface to see availability of given machines on a given date.	That would make the application easier. That allows users to see when machines are occupied and when not. This is also a way of showing data about all washing/drying machines on the same page.
Bookings can be cancelled	This is an extra functionality which is fair to implement. Users may not be happy with the time they have chosen and may wish to change it.
Provide a way to select the date for which to view the availability of the washing machines.	That allows users to make bookings in advance.
If the machine is not free when it should, that must be reported.	Reporting will help to optimize communication between staff and boarding students.
There should be a limit to the amount of booking time allowed to any boarding student per week.	That will limit the ability of students to book whole days for themselves, which is inefficient use of the machinery.

These features must provide users with a solution that will make the greatest difficulty go away - not knowing when machines are free to use (that was the most frequent complaint during the interview stage).

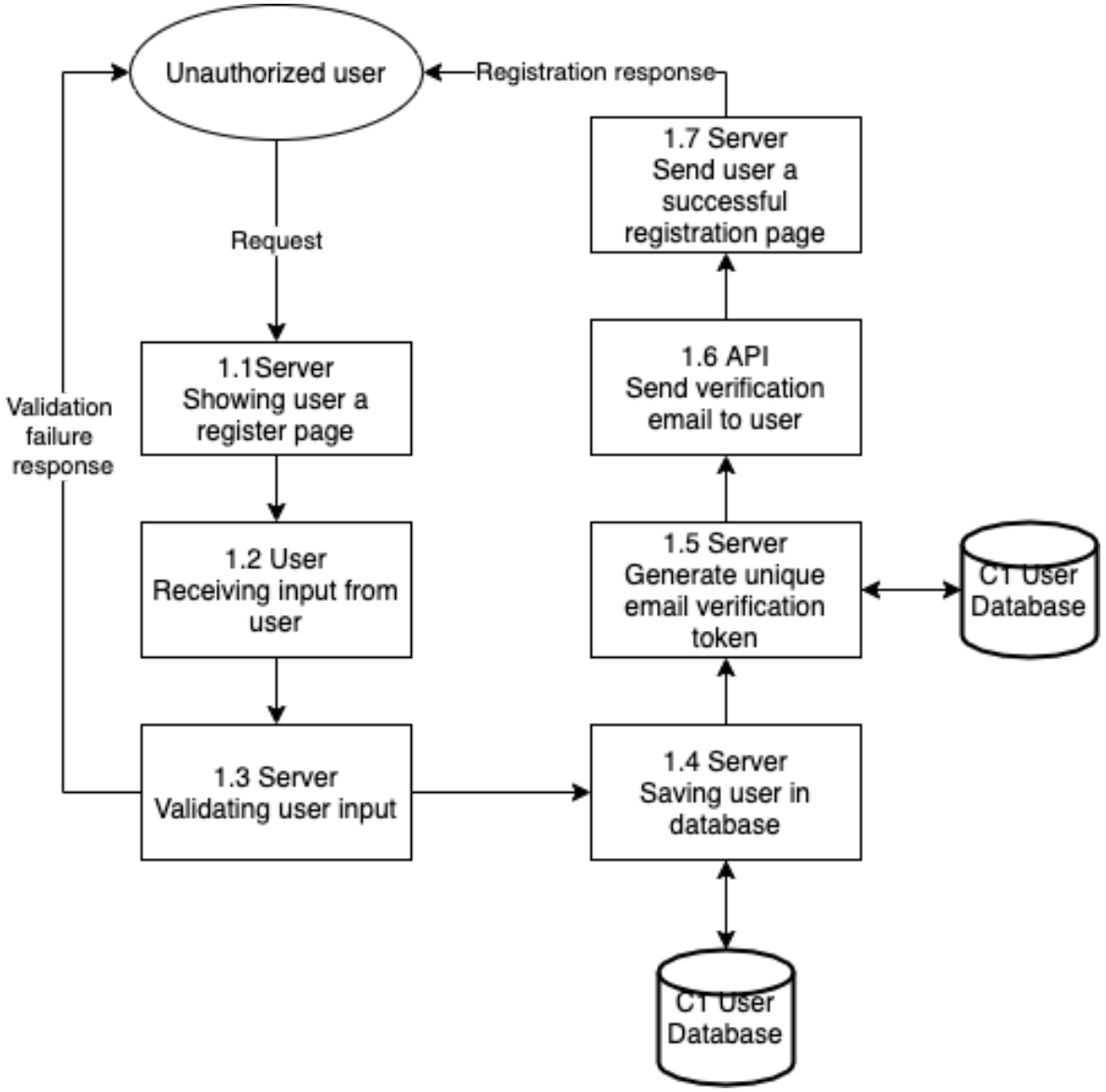
Initial Solution Modelling

With the features described above, the solution will simplify the algorithm for boarding students to use the washing and drying machines. The system will consist of the following main operations:

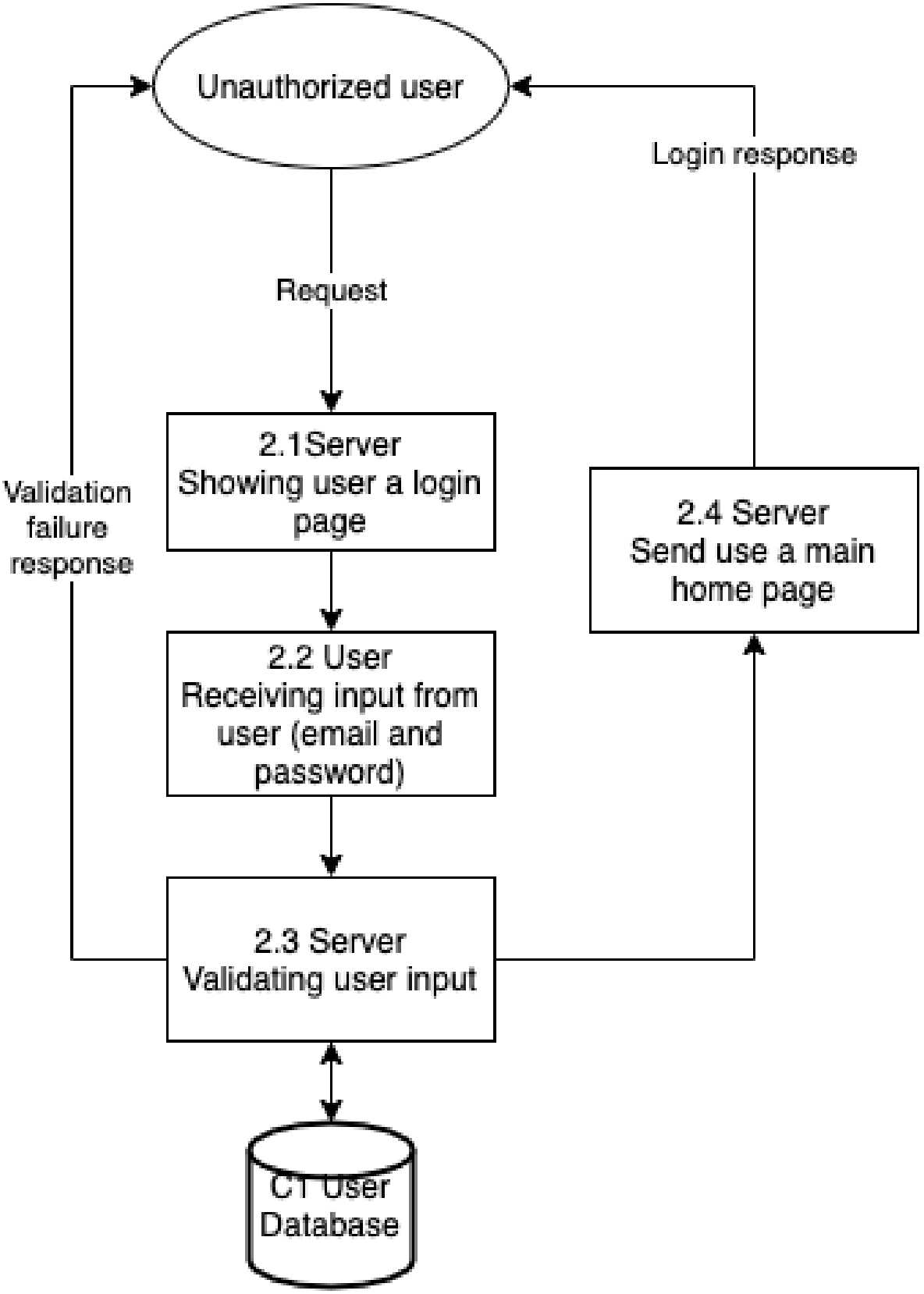
1) Registering on the platform
2) Login onto the platform
3) Viewing booking timetable for washing/drying machines on a specific date
4) Adding a booking
5) Viewing user's bookings
6) Cancelling a booking
7) Reporting a misuse of the laundry machines
8) Logging out from the platform

Here is the outline of how this actions would be designed (we will talk more about this in the design section):

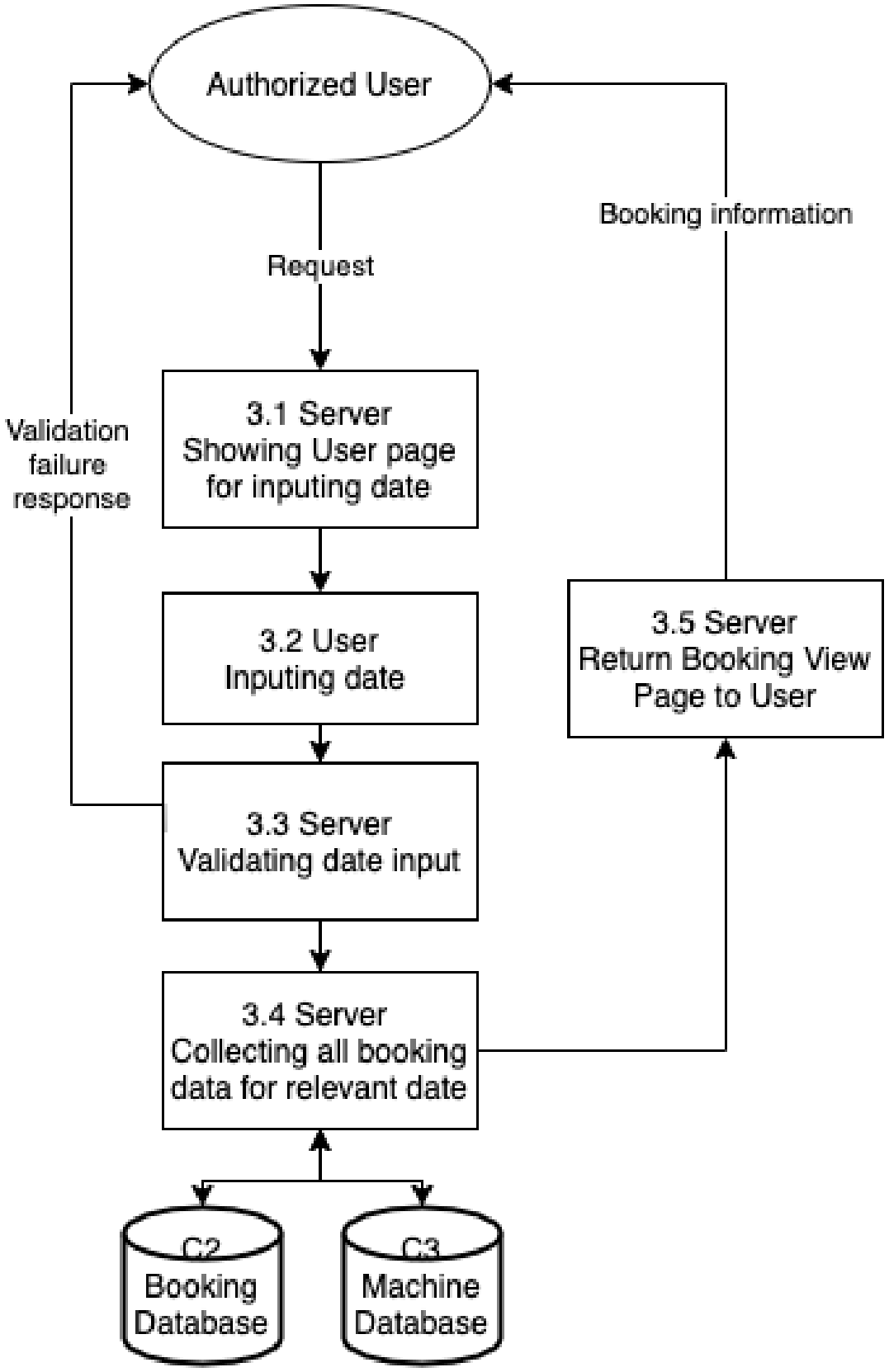
1) Registering on the platform



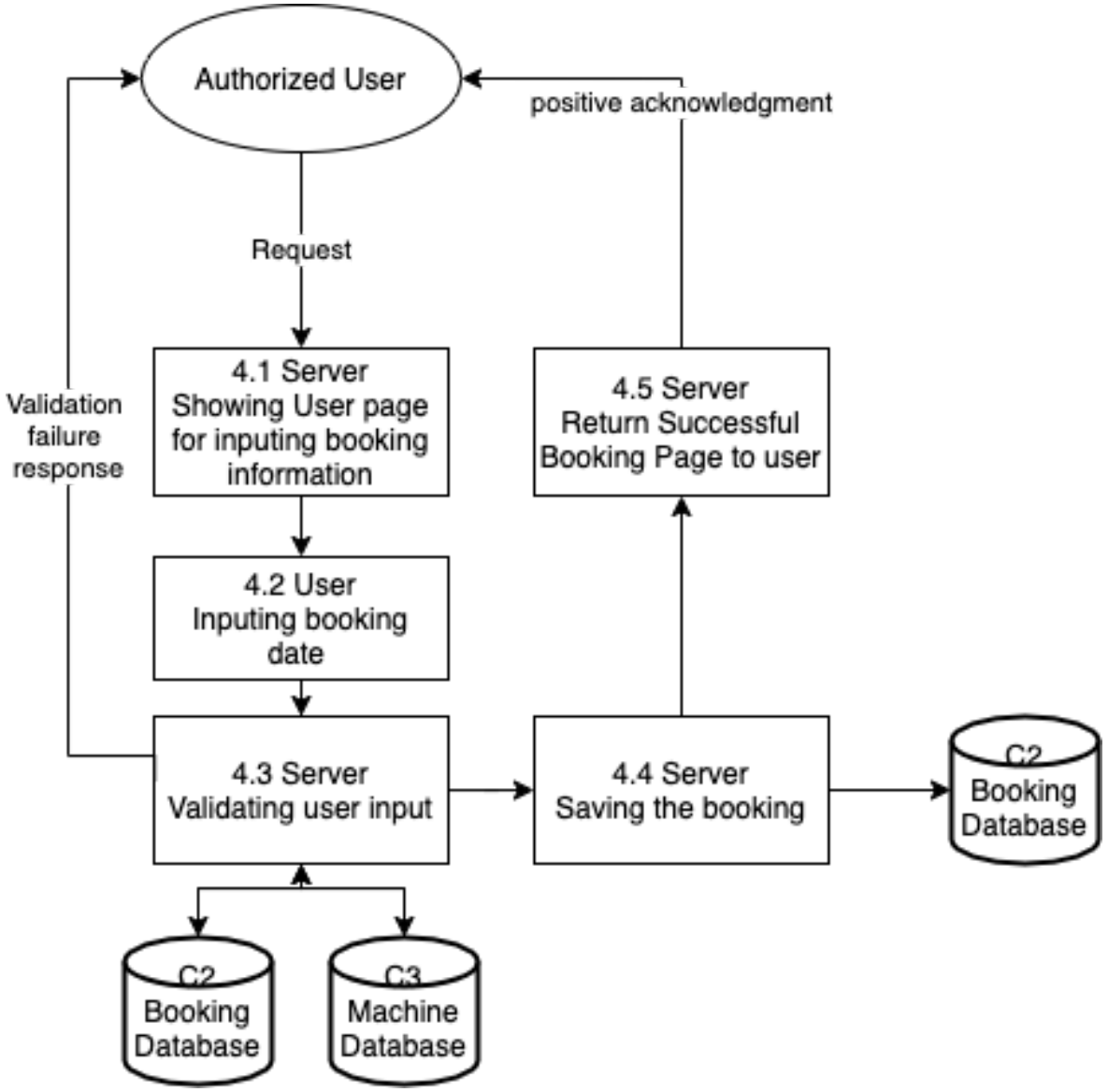
2) Login onto the platform



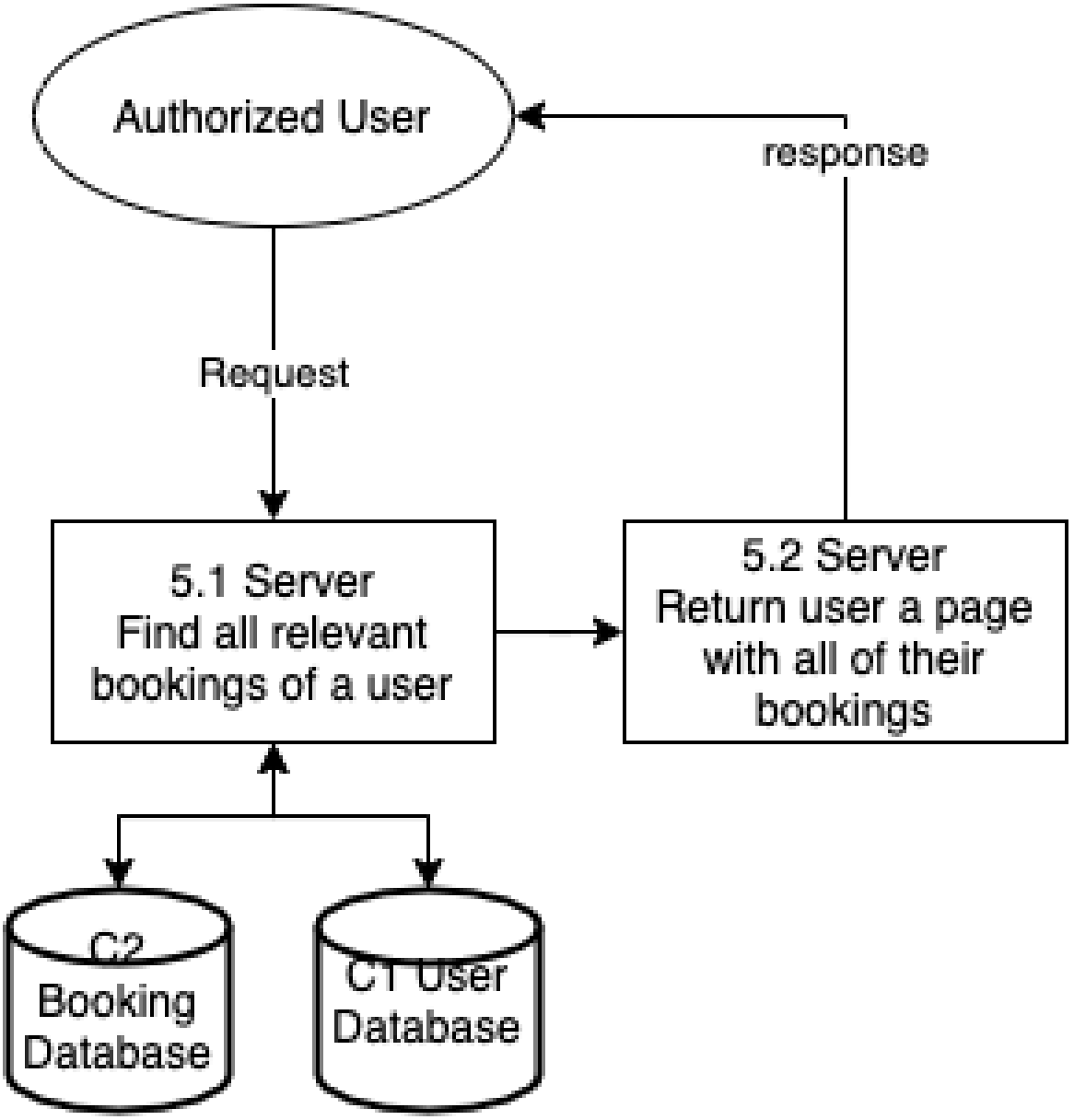
3) Viewing booking timetable for washing/drying machines on a specific date



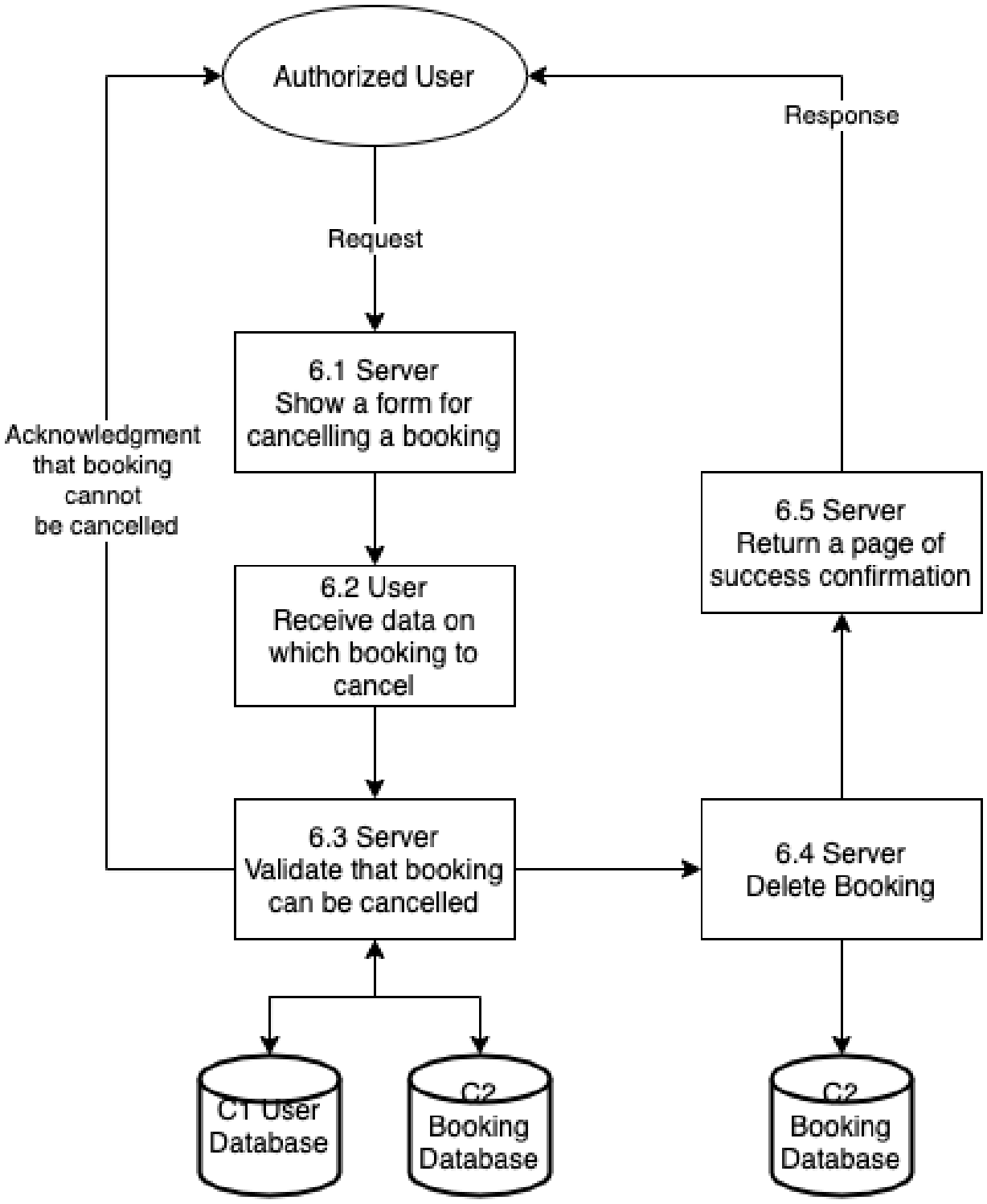
4) Adding a booking



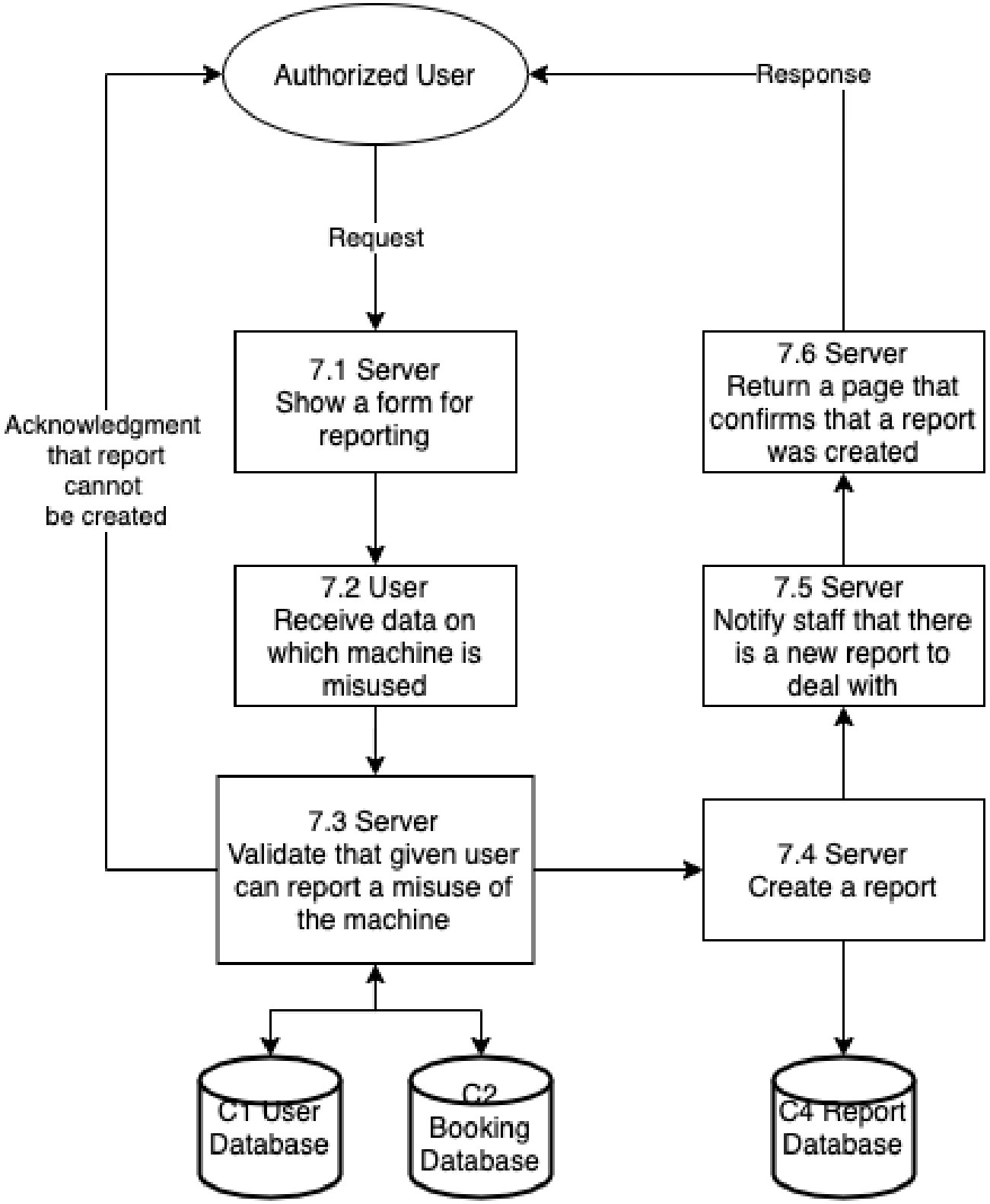
5) Viewing user's bookings



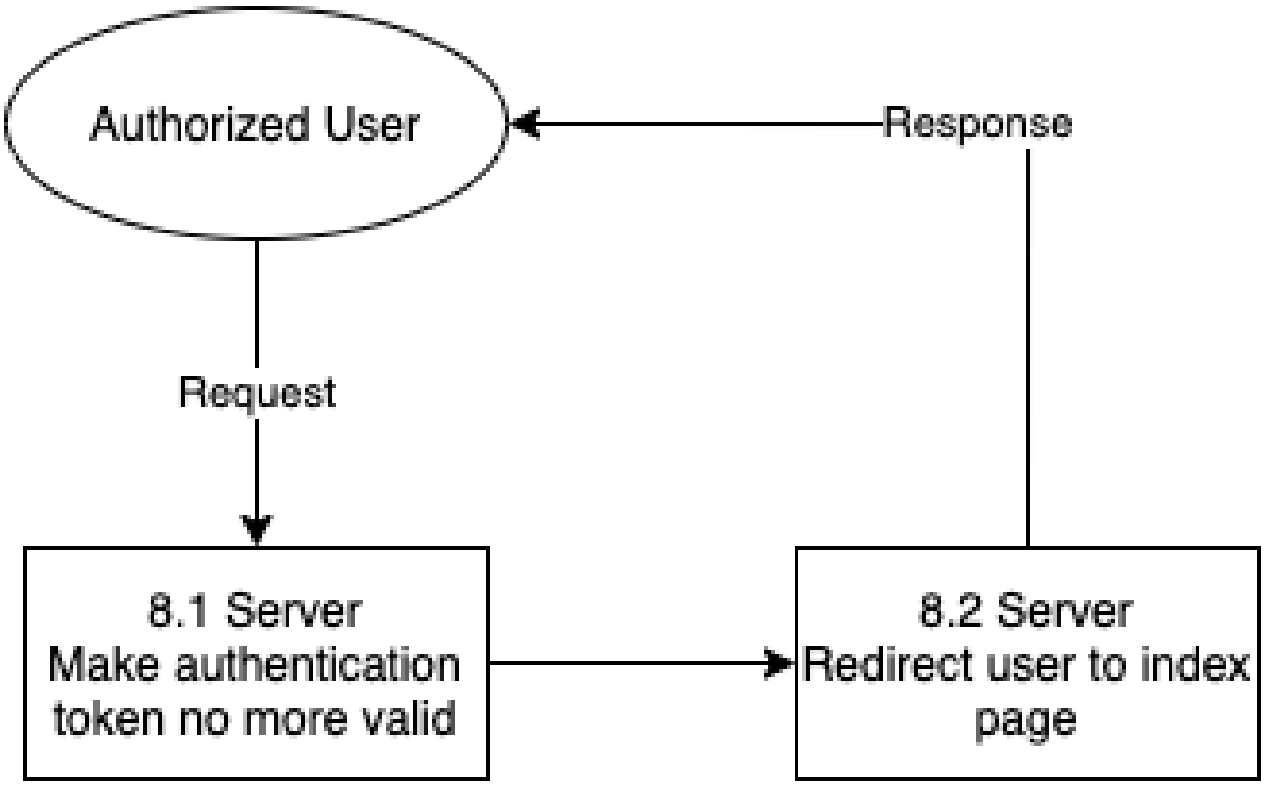
6) Cancelling a booking



7) Reporting a misuse of the laundry machines



8) Logging out from the platform



Important Note: On all of the diagrams above I have not specified how data gets to and from the user. That is because in all interactions in the client-server model we will be using HTTP. HTTP assumes that users send requests to a server that then handles it to produce valid responses.

Input validation in all of the steps above depend on the data volumes. Depending on what data about entities I will be storing, my validations would then also change. I will be going in greater depth into validations performed in the next subsection on this document.

While developing these diagrams, I have also noted a few other features that are not necessary for the operation of the program, but which would help to design a better application:

- Actions on the platform must be logged. That would ensure a swift process of debugging when problems would start to rise.

- There should be appropriate exception handling. Most errors should arise from incorrect user input, but as it would be validated that should not cause much of an

issue. However, there can be some unexpected actions, like entering random URLs, or some internal server errors. In those cases, users must be provided with an error page.

- There must be some protection enabled. Primarily I will focus on the following threats: CSRF, First-order SQL Injections, unencrypted data.

- Many actions that are performed by the server are “heavy” - they occupy a processor for too long. That limits the performance of the server dramatically. The solution to that would be ensuring that some actions are performed asynchronously.

- As time goes by a lot of booking data would become irrelevant. At 23:50 every day all booking details about that day should be removed from the database to decrease the amount of data stored in it.

Data Volumes

Based on the diagrams provided above I was able to build an initial understanding what kind of data must be stored in the database:

User-related data:

Attribute Name	Datatype	Description
id	Integer	Used to uniquely identify every user in a database. As all other fields about a user may vary, storing id as an integer would be the most efficient way.
name	String	Used to help staff to identify people that report and get reported.
email	String	Used to send emails to users and acts as a unique field for a user. Maybe later changed by a user, thus it is not an id field
password	String	Used to verify that a user owns an account associated with a given email.

Machine-related data:

Attribute Name	Datatype	Description
id	Integer	Used to uniquely identify every a machine in the system

type	String	Used to identify whether a machine is a washing or a drying machine

Booking-related data:

Attribute Name	Datatype	Description
user_id	Integer	Used to identify user that made the booking
machine_id	Integer	Used to identify machine to which booking is related
start_time	String	Used to specify starting time of the booking
end_time	String	Used to specify starting end of the booking

Time would be stored in String format because it would use less space in the database and would make it easier to use start_time and end_time in the system.

Report-related data:

Attribute Name	Datatype	Description
user_id	Integer	Used to identify user that made the report
booking_id	Integer	Used to identify the booking that came before, which then would be used to identify the user that didn't finish their washing on time.

As the amount of data stored about every entity is fixed and there are a lot of relations, it makes sense to use relational database architecture to store that data.

Validations

Based on the data stored I was able to come up with validations that would need to be performed to ensure that data is **added** to the database correctly. All other checks regarding user input would be discussed in the design section.

User-related validations:

Attribute Name	Check
id	Unique
name	Unique
email	In email format Unique
password	7+ characters

Machine-related validations:

Attribute Name	Check
id	Unique
type	One of the types provided (washing/drying)

Booking-related data:

Attribute Name	Check
user_id	Relates to user that actually exists
machine_id	Relates to machine that actually exists
start_time	Correct time format Comes after current time (cannot make a booking for the time that has already passed)
end_time	Correct time format Comes after start_time

Correct time format is given by: YYYY MM dd HH:mm

Extra validation: total booking time for a given user must not exceed a fixed value (e.g. 5 hours) per week.

Report-related validations:

Attribute Name	Check
user_id	Relates to user that actually exists
booking_id	Relates to booking that actually exists

End-Users and Prospective Users

My current end-users for which I create my program are all boarding students in my boarding house (~50 people). This system would also allow staff members to do washings, so I also include them as my end-users.

My prospective market includes all people that have to share laundry rooms but have no means of booking machines in advance. As described before, there are currently no systems in existence that allows booking washing machines online. People that would be the most interested in my program are students that live in big campuses (>20 people).

First place that comes into my mind are other boarding houses in my school. I know that they experience the same set of problems in regard to laundry as we do in my house.

Acceptable Limitations of My System

Right now I am planning to design a system only for a single laundry room. In the future my system might expand to include entities of laundry rooms, which would make it possible to use the system in different places.

As the system must be online 24/7, I cannot run it on my main computer that I use in school. Thus, I will run the program on my spare computer.

As for the minimum viable product it is not of the highest significance to obtain a casual and modern website, I will not spend a lot of time creating a nice-looking front-end for my service.

Current industry standard includes using JavaScript (or TypeScript) front-end in order to minimise the amount of operations that are performed by a back-end (like minimising the amount of times that a server must return views to users). Due to time constraints, I will have

only enough time to gain enough information on either back-end or front-end development. Thus, I will eliminate front-end JavaScript.

Due to time limitations I will be only to develop a website that would allow people to book, though a tablet application would be definitely more appropriate.

Proposed Solution Details

As I am familiar with Java it makes sense to focus on Java back-end development, rather than learning JavaScript to do back-end on Node.js

After initial research, I was able to come up with a list of technologies that I would use in my project.

Software (my stack of technologies):

1. Spring Core
2. Hibernate Framework
3. Log4J
4. SLF4J
5. Postgres database (and thus some PQL)
6. Spring Data JPA
7. Spring AOP
8. Apache Freemarker Template
9. XML
10. Maven
11. Git
12. Apache Tomcat Server
13. Java Mail API
14. HTML5
15. CSS3
16. Bootstrap 5

Hardware:

I will use my laptop to develop the application and then use another one to launch it online.

Set of Objectives

After my research was complete I was able to create a list of objectives that I need to accomplish in order to produce a good quality system.

Main objective: Build a web server that allows students to book washing and drying machines online.

Sub Objectives:

1. Users must be able to register on the platform.
    1.1. Users must get a register page.
    1.2. Users must be able to fill in an html form according to data that must be stored about a user.
        1.2.1. User must provide their name
        1.2.2. User must provide their email
        1.2.3. User must provide their password
        1.2.4. User must provide a second (check) password
    1.3. Input must be validated
        1.3.1. Email must be stored in the format of an email (******@***.***)
        1.3.2. Email must be uniquely stored in the database
        1.3.3. Passwords must match
        1.3.4. Password must be at least 7 characters long
        1.3.5. Name must be stored as two separate words
    1.4. After validation, the user must see an appropriate message which tells if they have passed validation or not.
    1.5. A new user entity must be saved in the database.
    1.6. Email verification token must be generated and stored in the database
        1.6.1. Email token must be of 5 characters and must be uniquely stored in the database
    1.7. Letter with an email verification link must be sent.
2. User must be able to verify their email
    2.1. After receiving the email with the token, the user must be able to follow the link which would automatically set the email as verified.
    2.2. Changes that indicate that email is verified must be saved in the database.
3. Users must be able to login onto the platform in order to see private content.
    3.1. Users must get a login page.
    3.2. Users must be able to fill in an html form according to data that must be stored about a user.
        3.2.1. User must provide their email
        3.2.2. User must provide their password
    3.3. Input must be validated
        3.3.1. Email must appear in the database

3.3.2. Email must be verified.
3.3.3. Password provided must match the password of the account associated with the provided email.
3.4. After validation, the user must see an appropriate message which tells if they have passed validation or not.
3.5. If a user has passed the validations, then they must get an authentication token assigned to them.

4. Users must be able to check washing and drying machine availability on any chosen day.
4.1. Authenticated users must have access to a page with washing and drying machine timetables.
4.2. Users must be able to select the date at which they want to see the machines' availability.
4.3. Date selected must be validated
4.3.1. Date must be provided in the format YYYY MM dd HH:mm

5. Users must be able to add a booking.
5.1. Users must have access to a form to make a booking.
5.2. Users must enter all relevant information about the booking.
5.2.1. Users must specify the machine type which they want to book.
5.2.2. Users must specify the machine id which they want to book.
5.2.3. Users must specify the starting time of the booking.
5.2.4. Users must specify the finishing time of the booking.
5.3. User's inputs must be validated
5.3.1. Machine type selected exists
5.3.2. Machine with the id specified exists
5.3.3. Starting time is after the current time
5.3.4. Starting time is in the time format
5.3.5. Finishing time is after the starting time
5.3.6. Finishing time is on the same day as the starting time
5.3.7. Finishing time is in the time format
5.3.8. Time slot selected is not yet booked
5.3.9. Combined booking time for a specific machine type in the given week does not exceed the maximum booking time for that machine type
5.4. After validation, the user must see an appropriate message which tells if they have passed validation or not. Appropriate error message must be displayed.
5.5. If a user has passed the validations, then a new booking must be saved in the database.

6. Users must be able to view their bookings.
6.1. Users must have an access to a page with their bookings
6.2. Server must identify the user and display currently active bookings (for the days that have not yet passed).

7. Users must be able to cancel a booking.
7.1. Users must be able to click on a "Cancel" button next to their booking on the "my bookings" page to cancel their booking.

7.2. Appropriate checks must take place to see if the booking can be cancelled.
    7.2.1. Check that booking exists
    7.2.2. Check that the booking was made by the user
    7.2.3. Check that the booking starts at most in 2 hours
7.3. After validation, the user must see an appropriate message which tells if they have passed validation or not. Appropriate error message must be displayed.
7.4. If the validation was successful the booking must be removed from the database.

8. Users must be able to report a misuse of the laundry machines.
    8.1. Users must have access to a page where they can access the form for reporting.
    8.2. Users must be able to enter the data required
        8.2.1. Users must enter machine id
        8.2.2. Users must select machine type
    8.3. Appropriate validations must take place
        8.3.1. Machine id refers to a machine that exists
        8.3.2. Machine type exists
        8.3.3. Currently there is a booking for current time for the given machine made by a current user
    8.4. After validation, the user must see an appropriate message which tells if they have passed validation or not. Appropriate error message must be displayed
    8.5. If the validation stage was passed then there must be a report added to the database. The report must also be displayed on a special page for staff to take notice
9. Users must be able to logout of the platform.
    9.1. Users must have a link to “logout” action
    9.2. On click their authentication token must be no longer valid
    9.3. Users must be redirected to the home page.
10. Users must be able to access relevant pages
    10.1. Authenticated users must have a way to access these pages:
        10.1.1. “View my bookings” page
        10.1.2. “View timetables” page
        10.1.3. “Report” page
        10.1.4. “index” page
    10.2. Unauthenticated users must be able to view these pages:
        10.2.1. “Login” page
        10.2.2. “Register” page
        10.2.3. “index” page
    10.3. Users must not be allowed on the pages that they should not be able to enter
    10.4. Users must be redirected to the relevant page if they attempt to enter a URL that they cannot access.
11. Actions on the platform must be logged.
    11.1. All web exceptions must be logged
    11.2. Booking creation/cancelation must be logged

11.3. Methods that carry out important functionality (that might be the cause of errors).

12. There should be appropriate exception handling.
    12.1. Users must see special page for internal server errors
    12.2. Users must see special page for “Page Not Found” error (HTTP Response: 404)
13. There must be some protection enabled.
    13.1. All forms must be protected with CSRF protection keys
    13.2. Passwords must be encrypted
    13.3. First order SQL-Injections must not be possible
14. Actions must be asynchronous when possible.
    14.1. Emails must be sent asynchronously
    14.2. Data must be deleted from the database asynchronously
15. Database content that is no longer relevant must be deleted automatically.
    15.1. Every day a scheduled function must be called that deletes all bookings and complaints for the passed day.
16. A database must be organised so that all data required is stored there.
    16.1. The following data must be stored in the database about users:
        16.1.1. Id (primary key)
        16.1.2. Name (unique field, not null)
        16.1.3. Email (unique field, not null)
        16.1.4. Password (not null)
    16.2. The following data must be stored in the database about machines:
        16.2.1. Id (primary key)
        16.2.2. Machine_type
    16.3. The following data must be stored in the database about bookings:
        16.3.1. User_id (foreign key, part of a composite primary key)
        16.3.2. Machine_id (foreign key, part of a composite primary key)
        16.3.3. Start_time (not null)
        16.3.4. End_time (not null)
    16.4. The following data must be stored in the database about complaints:
        16.4.1. User_id (foreign key, part of a composite primary key)
        16.4.2. Booking_id (foreign key, part of a composite primary key)

DESIGN SECTION

Deciding on Programming Language

The most common options for back-end development nowadays include the following frameworks:

- Laravel (PHP)
- Django (PYTHON)
- Node.js (JAVASCRIPT)
- Ruby on Rails (RUBY)
- Spring (JAVA)

I have chosen Spring framework for the following reasons:

- Java is a language that I am familiar with and have been using for many years.
- Spring is powerful and secure
- Spring is widely used in modern development

Overview of Technologies Used

- Spring Framework
    - Spring MVC
    - Spring Security
    - Spring AOP
    - Spring JPA
    - Spring Web
    - Spring ORM
    - Spring Context
    - Spring Core
    - Spring Beans
- Hibernate
    - Hibernate Validator
    - Hibernate ORM
- Logging
    - SLF4J
    - LOG4J
- Postgres
- Java Mail API
- Apache Freemarker Template Engine

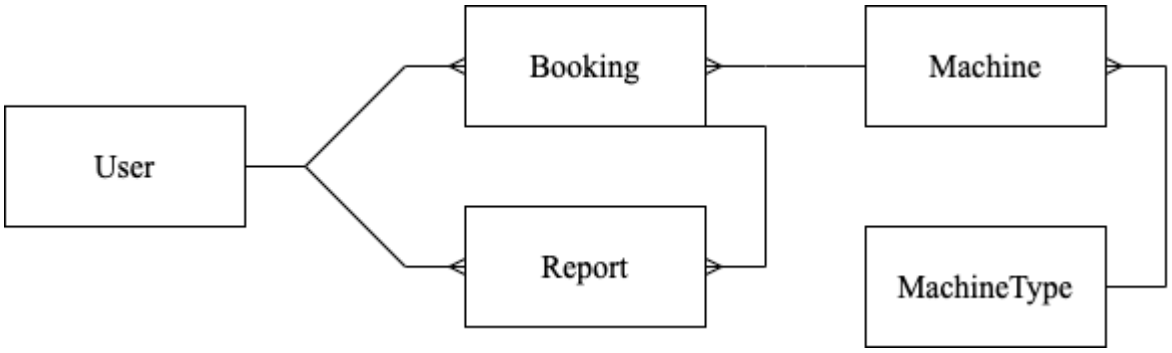
- Apache Tomcat Server
- Lombok
- Maven
- HTML5
- CSS3
- Bootstrap 5

Database Layout

There would be the following tables:

User(id, name, email, password, email_token)
Machine(id, type_id)
MachineType(type_id, description)
Booking(booking_id, user_id, machine_id, start_time, finish_time)
Report(user_id, booking_id)

Relational diagram:



Because booking is accessed in the Report table, we cannot create a composite key for the Booking table, though it would make sense to create one (made out of user_id, machine_id and start_time).

Accessing Database

In order to make database be accessible from the server's code I need to do a certain setup:

1) Download database onto the machine
2) Generate tables using the SQL commands listed on pages 28-30
3) Download Driver

Driver is a Java class that is designed to handle requests to the database. It can be thought of as an API.

As it is with many Java APIs, you do not need to use any other special syntax (JSON, XML) rather than java code to send requests. In this case, I do not need to send SQL requests personally. There are classes that would convert parameters I would pass into their methods into SQL and send it to the database. In fact, I have described JPA repositories and JPQL notation on pages 30-33 .

Particalary, JPA repositories make use of the Driver I would install to get the connection with the database and to create proxy connections.

I would specify Driver and data required to access the database in a special file: persistence.properties. It looks like this:

```
# Database username
user=atimokhina
# Database password
password=
# Link to access the database (could very well be on a separate computer)
url=jdbc:postgresql://localhost:5432/washing_queues_database
# name of the pre-downloaded Driver class
driver=org.postgresql.Driver
```

To establish connection using this data we need a list of objects:

1) DataSource - object that represents the actual database that we can access. This object would need data from the persistence.properties file.
2) TransactionManager - object that opens transactions (environments in which the server interacts with the database) to obtain data from the tables.
3) EntityManagerFactory - object that would map models (page 34) to an appropriate ORM equivalent in the database.

These objects would be singletons (page 34) and thus would be generated before the launch of the actual website.

Database Table Generation SQL

The following SQL commands are executed to create the tables:

```
DROP TABLE IF EXISTS user_table;
DROP TABLE IF EXISTS machine_types_table;
DROP TABLE IF EXISTS machine_table;
DROP TABLE IF EXISTS booking_table;
DROP TABLE IF EXISTS report_table;

/*user_table*/
CREATE TABLE user_table
(
   id   SERIAL       NOT NULL,
   name VARCHAR(255) NOT NULL UNIQUE,
```

```
   email VARCHAR(255) NOT NULL UNIQUE,
   password VARCHAR NOT NULL,
   Email_token CHAR(5) NOT NULL
);

CREATE UNIQUE INDEX user_table_id_uindex ON user_table (id);

ALTER TABLE user_table ADD CONSTRAINT user_table_pk PRIMARY KEY (id);

/*machine_types_table*/
CREATE TABLE machine_types_table
(
   type_id   SERIAL       NOT NULL,
   description VARCHAR(255) NOT NULL UNIQUE,
);

CREATE UNIQUE INDEX machine_types_table_type_id_uindex ON machine_types_table
(type_id);

ALTER TABLE machine_types_table ADD CONSTRAINT machine_types_table_pk PRIMARY KEY
(type_id);

/*machine_table*/
CREATE TABLE machine_table
(
   id   SERIAL       NOT NULL,
   type_id   INT      NOT NULL,
   CONSTRAINT machine_table_machine_types_table_id_fk                FOREIGN KEY
(type_id) REFERENCES machine_types_table       (type_id) ON DELETE DELETE
);

CREATE UNIQUE INDEX machine_table_id_uindex ON machine_table (id);
ALTER TABLE machine_table ADD CONSTRAINT machine_table_pk PRIMARY KEY (id);


/*booking_table*/

CREATE TABLE booking_table
(
   booking_id   SERIAL       NOT NULL,
    user_id   INT      NOT NULL,
    machine_id   INT      NOT NULL,
    start_time CHAR (16) NOT NULL,
    end_time CHAR (16) NOT NULL,
);


CREATE UNIQUE INDEX booking_table_id_uindex ON booking_table (booking_id);

ALTER TABLE booking_table ADD CONSTRAINT booking_table_booking_id_pk PRIMARY KEY
(booking_id);


/*report_table*/

CREATE TABLE report_table
(
    user_id   INT      NOT NULL,
    CONSTRAINT report_table_user_table_id_fk FOREIGN KEY (user_id) REFERENCES
    user_table (id) ON DELETE DELETE,

    machine_id   INT      NOT NULL,
    CONSTRAINT report_table_machine_table_id_fk FOREIGN KEY (machine_id) REFERENCES
    machine_table (id) ON DELETE DELETE
);
```

```
ALTER TABLE booking_table ADD CONSTRAINT report_table_user_id_pk PRIMARY KEY
(user_id) ADD CONSTRAINT report_table_machine_id_pk PRIMARY KEY (machine_id);
```

Communication with the Database

In order to work with data inside of the database I will be using the ORM paradigm. It allows me to work with data inside the tables as if they were JAVA objects.

Specifically, I will be using Hibernate JPA - a sort of implementation of the ORM paradigm. Due to specific syntax JPA uses (JPQL), I will list the queries that I would be performing in my database in PQL down here.

User-related Queries:

1) Save User in database

```
SQL (PQL):
INSERT INTO user_table(id, name, email, password,
email_token)
VALUES (id, name, email, password, email_token);

JPQL:
*This query is replaced by save() method in JPA and
Hibernate ORM*
```

2) Find User by email:

```
SQL (PQL):
SELECT * FROM user_table WHERE email = email;

JPQL:
SELECT u FROM User u WHERE u.email = :email
```

3) Find Users with not null email verification token:

```
SQL (PQL):
SELECT * FROM user_table WHERE email_token IS NOT NULL;

JPQL:
SELECT u FROM User u WHERE u.emailToken IS NOT NULL
```

4) Find User by specific email verification token:
   SQL (PQL):

```
SELECT * FROM user_table WHERE email_token = emailToken;
```

JPQL:

```
SELECT u FROM User u WHERE u.emailToken = :token
```

Machine-related Queries:

1) Find All machines :

SQL (PQL):

```
SELECT * FROM machine_table;
```

JPQL:

```
SELECT m FROM Machine m
```

2) Find machine by id:

SQL (PQL):

```
SELECT * FROM machine_table WHERE id = id;
```

JPQL:

```
SELECT m FROM Machine m WHERE m.id = :id
```

Booking-related Queries:

1) Save Booking in database

SQL (PQL):

```
INSERT INTO booking_table(machine_id, user_id,
start_time, end_time)
VALUES (machine_id (SELECT id FROM machine_table WHERE id
= machine_id),
user_id (SELECT id FROM user_table WHERE id = user_id),
start_time, end_time);
```

JPQL:

```
Booking booking = new Booking (User, Machine, start_time,
end_time);
save(booking);
```

2) Delete Booking from database

SQL (PQL):

```
DELETE FROM booking_table WHERE booking_id = booking_id
```

JPQL:

```
delete(booking);
```

3) Find Booking for a given machine from database:

SQL (PQL):

```
SELECT booking_table.user_id, booking_table.machine_id,
booking_table.start_time, booking_table.end_time FROM
booking_table WHERE booking_table.machine_id = id AND
INNER JOIN machine_table ON booking_table.machine_id =
machine_table.id ORDER BY booking_table.machine_id ASC;
```

JPQL:

```
SELECT b FROM Booking b WHERE b.machine.id = :id ORDER BY
b.machine.id ASC
```

4) Find Booking for a given user from database:

SQL (PQL):

```
SELECT booking_table.user_id, booking_table.machine_id,
booking_table.start_time, booking_table.end_time FROM
booking_table WHERE booking_table.user_id = id AND INNER
JOIN user_table ON booking_table.user_id = user_table.id
ORDER BY booking_table.user_id ASC;
```

JPQL:

```
SELECT b FROM Booking b WHERE b.user.id = :id ORDER BY
b.user.id ASC
```

5) Find Booking for a given user and for time that starts with [] from database:

SQL (PQL):

```
SELECT booking_table.user_id, booking_table.machine_id,
booking_table.start_time, booking_table.end_time FROM
booking_table WHERE booking_table.user_id = id AND date
LIKE booking_table.start_time AND INNER JOIN user_table
ON booking_table.user_id = user_table.id ORDER BY
booking_table.user_id ASC;
```

JPQL:

```
SELECT b FROM Booking b WHERE b.user.id = :id AND :date
LIKE(b.startDate)
```

Report-related Queries:

1) Save Report in database

```
SQL (PQL):
INSERT INTO report_table(machine_id, user_id)
VALUES (machine_id (SELECT id FROM machine_table WHERE id
= machine_id),
user_id (SELECT id FROM user_table WHERE id = user_id));

JPQL:
Report report = new Report (User, Machine);
save(report);
```

2) Find All Reports in database

```
SQL (PQL):
SELECT report_table.machine_id, report_table.user_id FROM
report_table, machine_table, user_table WHERE
report_table.machine_id = machine_table.id AND
report_table.user_id = user_table.id;

JPQL:
findAll();
```

3) Delete All Reports in database

```
SQL (PQL):
DELETE report_table.machine_id, report_table.user_id FROM
report_table, machine_table, user_table WHERE
report_table.machine_id = machine_table.id AND
report_table.user_id = user_table.id;

JPQL:
deleteAll();
```

Note: Usually parameters in SQL that would be passed in are given by the "?" symbol. I have decided to use words in **bold** instead to make it easier to comprehend what data would be passed into each of those fields.

As you can see, many queries are already predefined by JPA and I do not need to implement them myself.

Benefits of Hibernate and JPA Repositories

Using JPQL takes care of SQL-injections. As parameters are used, they would be stored as strings and not part of actual commands. There are many benefits of using higher level methods of communication with databases.

Another advantage of Hibernate is that it allows lazy data fetching - instead of loading all data related to a single object, it would load that data from related tables when the data is needed to be accessed.

Hibernated also forces the use of the ACID pattern of working with databases. That means that data can be accessed efficiently by the help of transactions.

Transactions are scopes in which the server “talks to” the database. All requests related to a given object are executed in one transaction.

MVC

MVC stands for Model-View-Controller. It is a software design pattern that describes how to create web applications scalable and manageable.

Code responsible for web interactions can be split into three main sub-categories: models, view and controllers.

Model is a part of the application that represents and interacts with data stored at the backend. Model sets the rules for how the data would be stored, updated and accessed.

In my application there are a number of Models: there are classes that represent an object stored in the database; repositories that handle the connection and queries between the database and the rest of the program; service layer that sets the rules for what kind of operations can be performed with every kind of object stored in the database.

I have the following objects: Booking, Report, Machine, MachineType, User. Each of these classes has its own repository and service with its implementation. Classes listed above would have the following attributes:

Booking

Parameter name and type	Use	Sample value stored	Limitations
id: integer	Uniquely identifies a booking object in the database for referencing.	3	Up to 2147483647 (4 bytes used to store this value in the database)
machine: Machine	Reference a machine for which the booking was made	{id:1; machineType: {id: 1; description: "WASHING_MACHINE"}}	There is a many to one relationship. Hence, each booking can reference only a single Machine object.
user: User	Reference a user that made the booking.	{id:12; email:"example@gmail.com"; name: "John Smith"; password: "hsgb2688d7hh2gh#hshh616twtggv"; emailToken: ""}	There is a many to one relationship. Hence, each booking can reference only a single User object.
startDate: String	Represents the date and time when the booking time starts	"2021 18 18 15:00"	String must come in the format: YYYY MM dd HH:mm. Other formats would not be accepted.
endDate: String	Represents the date and time when the booking time ends	"2000 02 17 16:30"	String must come in the format: YYYY MM dd HH:mm. Other formats would not be accepted
complaints: List<Report>	Stores references to all reports that are associated with this booking	[{id:1; repostingUser:{id:12; email:"example@gmail.com"; name: "John Smith"; password: "hsgb2688d7hh2gh#hshh616twtggv"; emailToken: ""}; booking: this }]	This relations are stored as a foreign key column in report_table

Report

Parameter name and type	Use	Sample value stored	Limitations
id: integer	Uniquely identifies a report object in the database for referencing.	10209	Up to 2147483647 (4 bytes used to store this value in the database)
reportingUser: User	Reference a user that reported misuse of the system.	{id:12; email:"example@gmail.com"; name: "John Smith"; password: "hsgb2688d7hh2gh#hshh616twtggv"; emailToken: ""}	There is a many to one relationship. Hence, each report can reference only a single User object.
booking: Booking	Reference a booking that caused the report to be created. If report is initiated then the booking that happened before the time of the current one, for the given machine, would be stored as the one that caused the report	{id: 400; machine: {id:1; machineType: {id: 1; description: "WASHING_MACHINE"}}; user: {id:12; email:"example@gmail.com"; name: "John Smith"; password: "hsgb2688d7hh2gh#hshh616twtggv"; emailToken: ""}; startDate: "2021 18 18 15:00"; endDate: "2021 18 18 18:00"}	There is a many to one relationship. Hence, each report can reference only a single Booking object.

Machine

Parameter name and type	Use	Sample value stored	Limitations
id: integer	Uniquely identifies a machine object in the database for referencing.	9	Up to 2147483647 (4 bytes used to store this value in the database)
type: MachineType	Identifies type of machine.	{id:1; typeDescription: "DRYING_MACHINE"}	Limited to WASHING_MACHINE and DRYING_MACHINE (also known as

			dryer)
bookings: List<Booking>	References all bookings that are registered for current machine	[{id: 400; machine: {id:1; machineType: {id: 1; description: “WASHING_MACHINE”}}; user: {id:12; email:”example@gmail.com”; name: “John Smith”; password: “hsgb2688d7hh2gh#hshh616twtggv”; emailToken: “”}; startDate: “2021 18 18 15:00”; endDate: “2021 18 18 18:00”}, {id: 401; machine: {id:1; machineType: {id: 1; description: “WASHING_MACHINE”}}; user: {id:12; email:”example@gmail.com”; name: “John Smith”; password: “hsgb2688d7hh2gh#hshh616twtggv”; emailToken: “”}; startDate: “2021 18 18 15:00”; endDate: “2021 18 18 18:00”}]	This relations are stored as a foreign key column in booking_table

MachineType

Parameter name and type	Use	Sample value stored	Limitations
id: integer	Uniquely identifies a machineType object in the database for referencing.	2	Up to 2147483647 (4 bytes used to store this value in the database)
typeDescription: string	Description of machine type. Used for displaying	“WASHING_MACHINE”	The value must be unique in the database. The values

	machine type in the program for users.		that can be created must be a part of a special enumerated list. I called this list Types.
machineList: List<Machine>	References all machines that use this machineType	[{Booking object}, {Booking object}, …]	This relationship is stored as a foreign key column in machine_table

User

Parameter name and type	Use	Sample value stored	Limitations
id: integer	Uniquely identifies a User object in the database for referencing.	10289	Up to 2147483647 (4 bytes used to store this value in the database)
name: String	Used to give user more personalised experience of using the application by displaying their name	Rober Child	Must consist of two words.
email: String	Used to identify a user's account and to send them emails.	example@example.com	Must be unique and be in the format of the email (more on this in the Validation section (page 66))
emailToken: String	Used for validation of the user's account. (More on this in the Email API section (page 62))	71625	Consists of 5 digits. Must be unique
bookings: List<Booking>	References all bookings that are registered for a given user	[{id: 400; machine: {id:1; machineType: {id: 1; description: “WASHING_MACHINE”}}; user: {id:12; email:”example@gmail.com”; name: “John Smith”; password:	This relations are stored as a foreign key column in booking_table

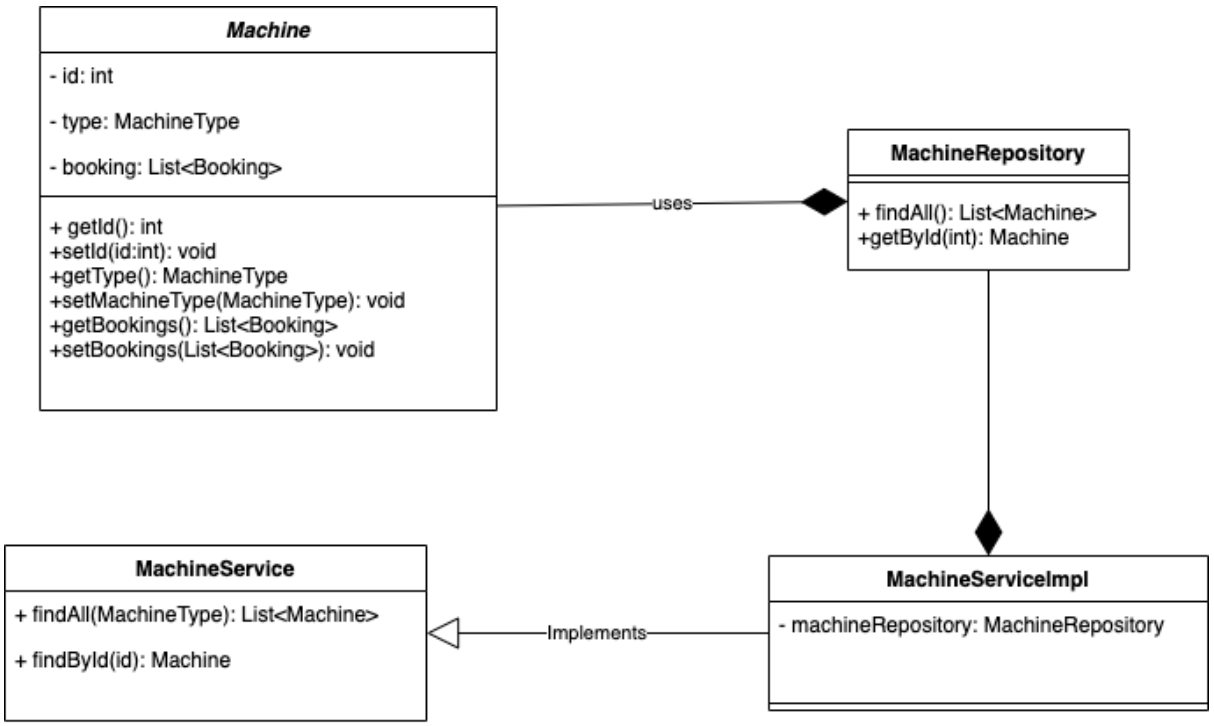
		“hsgb2688d7hh2gh# hshh616twtggv”; emailToken: “”}; startDate: “2021 18 18 15:00”; endDate: “2021 18 18 18:00”}, {id: 401; machine: {id:1; machineType: {id: 1; description: “WASHING_MAC HINE”}}; user: {id:12; email:”example@g mail.com”; name: “John Smith”; password: “hsgb2688d7hh2gh# hshh616twtggv”; emailToken: “”}; startDate: “2021 18 18 15:00”; endDate: “2021 18 18 18:00”}]	
complaints: List<Report>	Stores references to all reports that are associated with this user.	[{id:1; repostingUser:{id:12 ; email:”example@g mail.com”; name: “John Smith”; password: “hsgb2688d7hh2gh# hshh616twtggv”; emailToken: “”}; booking: this }]	This relations are stored as a foreign key column in report_table

Types enum is designed to store two parameters:

WASHING_MACHINE, DRYING_MACHINE

No particular values are associated with these parameters - their names are the values we are interested in.

Repositories, services and service implementations for each of the models have the same kind of relationship structure. The example below is given for Machine model:



Second part of the design pattern is views. They are files that are responsible for visualising data for users. In the case of this application, this is done through FTL pages. Those are covered in greater detail in the section that goes over FTL (page 41).

Last fundamental part of the user-server interactions are controllers. They are JAVA classes that take data from the user to update models or to get some views returned.

In this project all files that act as controllers are located in the “controllers” directory. In essence, all controller classes are organised in a similar fashion. They contain methods that are mapped to certain HTTP requests from the user. Inside these methods program calls services and other back-end-oriented classes like validation classes. Classes that are responsible for handling TCP/IP protocol and invoking correct methods in the controllers are known as servlets. Due to the fact that I do not require any unusual setup, I would be using the servlet that Spring provides - dispatcher servlet.

Inversion Of Control and Dependency Injections

I am using a singleton pattern for this application. This allows me to use the same instance of the class in all other objects that I desire. For instance, in the example in the MVC section (page 34) I have indicated that MachineRepository is used inside MachineServiceImpl. In practice, I may end up using that repository in other classes. I want to make sure that it is the same object that I will be using. Therefore I need to create a MachineRepository object and store it in some container from which I can retrieve it when needed to use in other objects. That is how IoC and DI work.

In terms of code that example with the MachineRepository can be visualised as follows:

```
public class MachineRepository{
        // variables …
        public MachineRepository(){
                // constructor
        }

        // other methods…
}

public class Container{
    public static final MachineRepository machineRepository = new MachineRepository();
    // OTHER OBJECTS…
}
public class MachineServiceImpl{
        // to get MachineRepository object we need to call the Container class as follows:
        MachineRepository = Container.machineRepository;
}
```

Hopefully this example explains this technique better.

Custom implementation of IoC and DI has its drawback - you need to put all objects into the container class prior to program running. That can be problematic as the solution that I am creating has about 70 different singletons. Each singleton can have inner dependencies that must be fulfilled prior to saving that object into the container. That might cause issues where two objects must be initialised at the same time but they are referencing each other.

These problems are solved by Spring Framework. Before running the code that I have written, Spring would start by creating a so-called Spring container and would then identify what singletons to store and what dependencies they must get before being able to be accessed by other singletons. Spring identifies classes that would be stored in the container if the class is annotated with one of the following annotations: **@Bean, @Component, @Service, @Repository, @Controller, @Configuration**.

To identify where to inject those singletons @Autowired annotation is used over the attribute in the class.

FTL

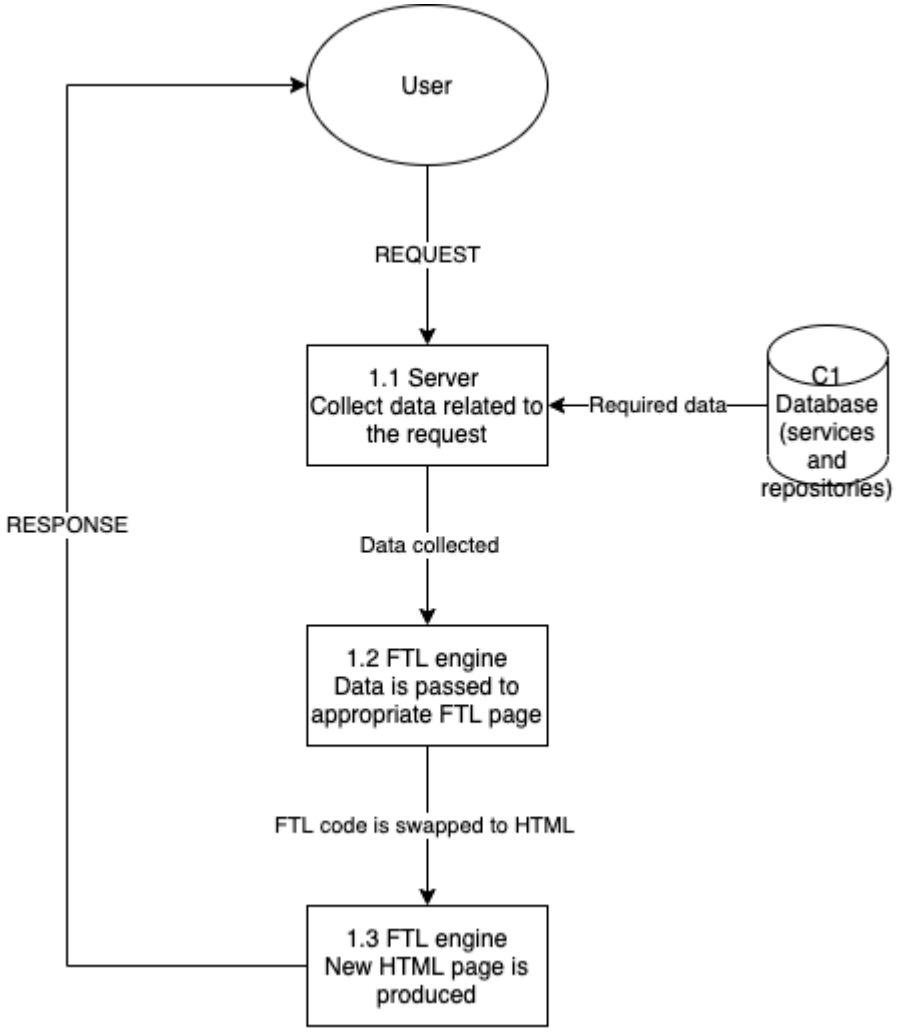
There exists an alternative tool for creating dynamically-updating websites. Instead of creating RESTful servers and using PHP or JavaScript front ends, you can use a template engine.

Template engines allow to create a code that would be executed before the user would receive a requested HTML page. That code would be executed to change the HTML, CSS codes that are sent to the user.

This method has its downside - it requires a lot of processing power from the server to create these pages for every of its users. However, as the project would serve a limited number of users simultaneously that is not of a concern. The greatest benefit of using the template engine, in my case, is its relative simplicity. Powerful Web-oriented languages like JAVA have their own template engines with a syntax that is not too different from that language's.

I would be using the Apache FreeMaker Template engine to generate dynamic content on the website.

The scenario in which I would be using FTL is the identical to all pages, and it is summarised in the diagram below:



To pass data collected from a controller to the appropriate FTL page two steps are required:

1) As a return value of the controller specify the FTL page location

2) In the controller add the key-value pair (that then can be accessed in the FTL page) to a Model object

Here is an example:

Controller:

```
public String signUpPost(@RequestParam(name = "email") String email,
                         @RequestParam(name = "name") String name,
                         @RequestParam(name = "password") String password,
                         @RequestParam(name = "passwordTwo") String passwordTwo,
                         Model model) {

   User user = new User(name, email, password);

   List<UserError> errorList = userValidator.validate(user, passwordTwo); // making our
validations

   // We decide whether there are any errors and then either return a fail or a success
validation message. Objective 1.4
   if (!errorList.isEmpty()) {
       model.addAttribute("errors", errorList);
       return "signUp";
   } else {
       // ...
   }

}
```

signUp.ftl page:

```
...
<#if errors??>
   <div class="row">
       <div class="col-12">
           <#list errors as error>
               <h4 class="error-msg">${error.getMessage()}</h4>
           </#list>
       </div>
   </div>
</#if>
...
```

FTL syntax starts with the hash (#) symbol. In the example given you can see that in the signUpPost() method a list of errors is passed as a parameter (named “errors”) through the model object. In the signUp.ftl page we can access that “errors” element (‘??’ - means “exists?”)

Human-Computer Interactions

This program is a web-application. All of the interaction, from the clients' side, would happen through a web browser.

Users will input data into HTML-forms and receive a response in the form of a new web page.

Here are some of the most significant forms that user can send to the server:

Sign up form:

Data inputted by a user:

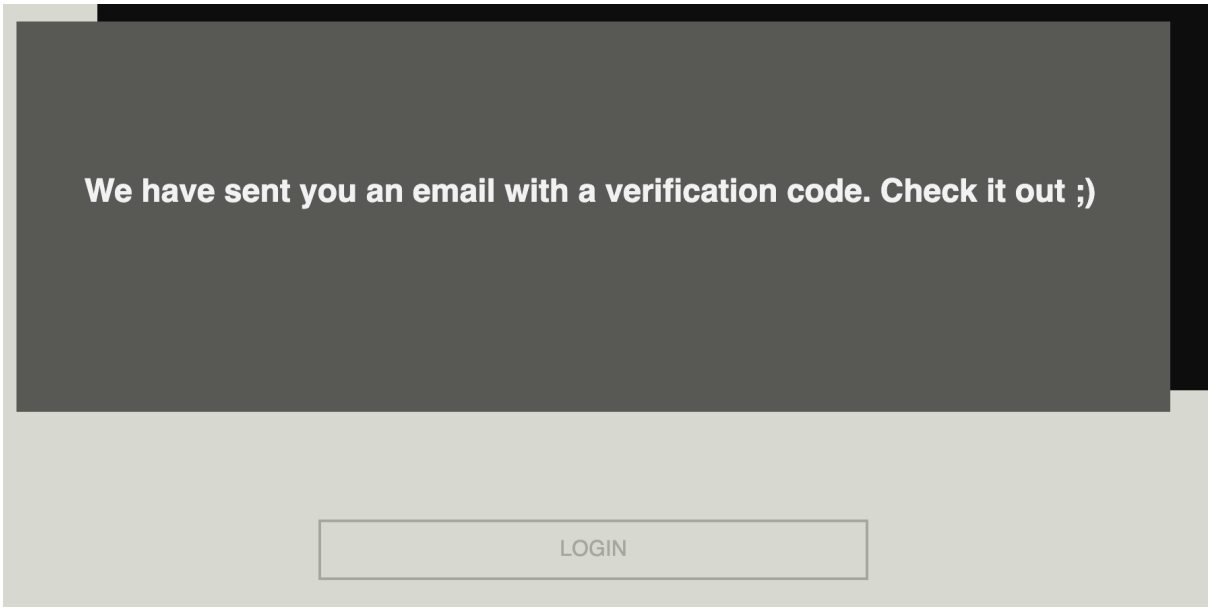
- Email
- Name
- Password
- Confirm password

Response from a computer:

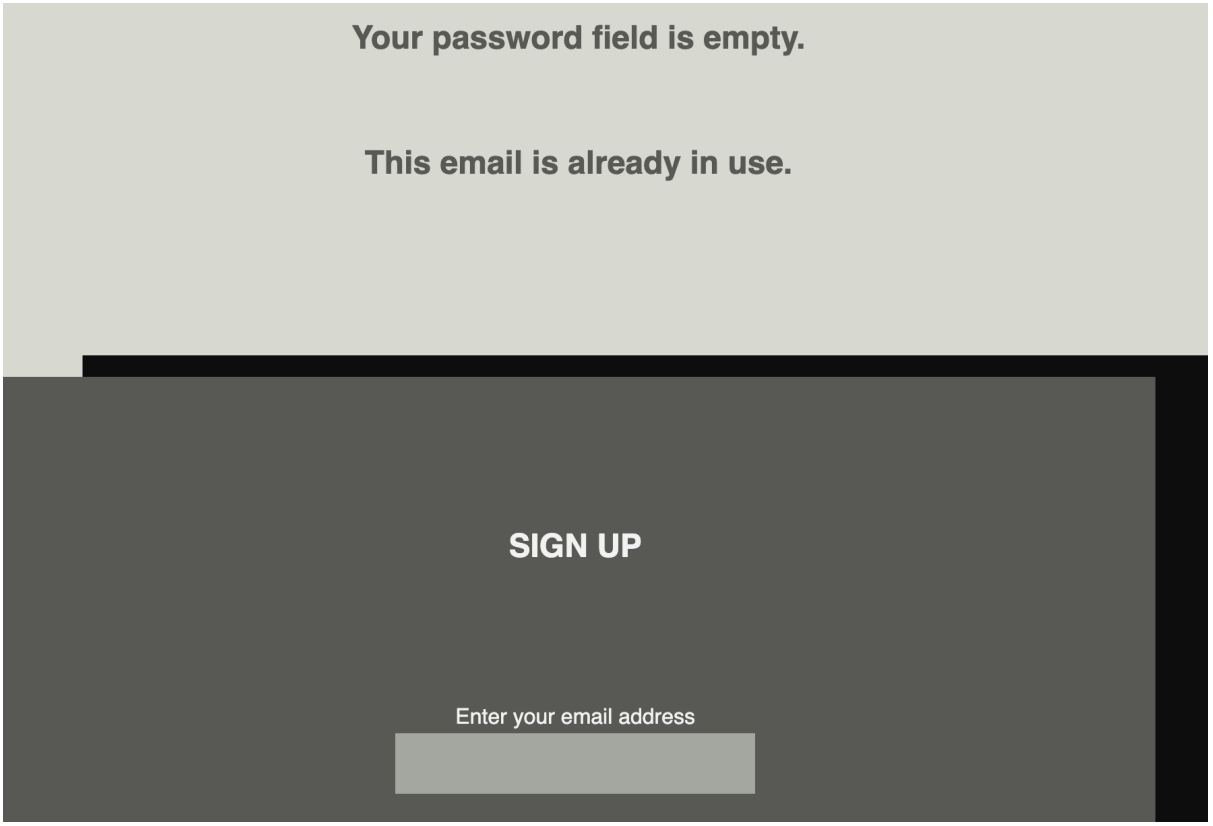
- Redirected to a "email verification letter is sent" message-page if all validations were passed successfully.
- Redirected to the sign up page with an error message on it.

Images of the given HCI:

The actual form.



The “email verification letter is sent” message-page.



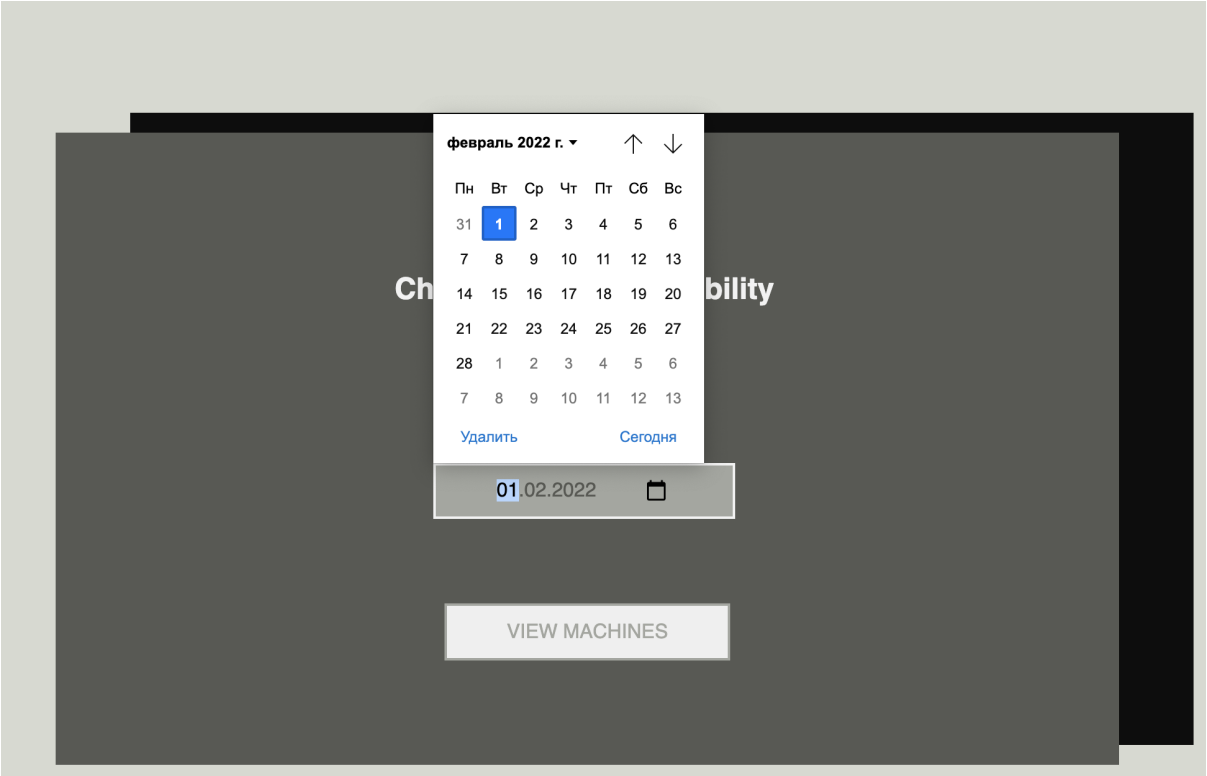
The sign up page with the error messages.

Washing machine booking form:

This is rather an unusual form. Each washing machine gets its own table that represents all possible time slots for a given day. Next to each vacant time slot there is a checkbox. When

the user selects all slots they need, they must press a “submit” button at the bottom of the form to send the request for a booking.

Before being able to view the available slots for a washing machine, the user must select a date. The date is selected on a separate page and the information about the selected date would be stored inside the washing machine booking form. It looks like this:



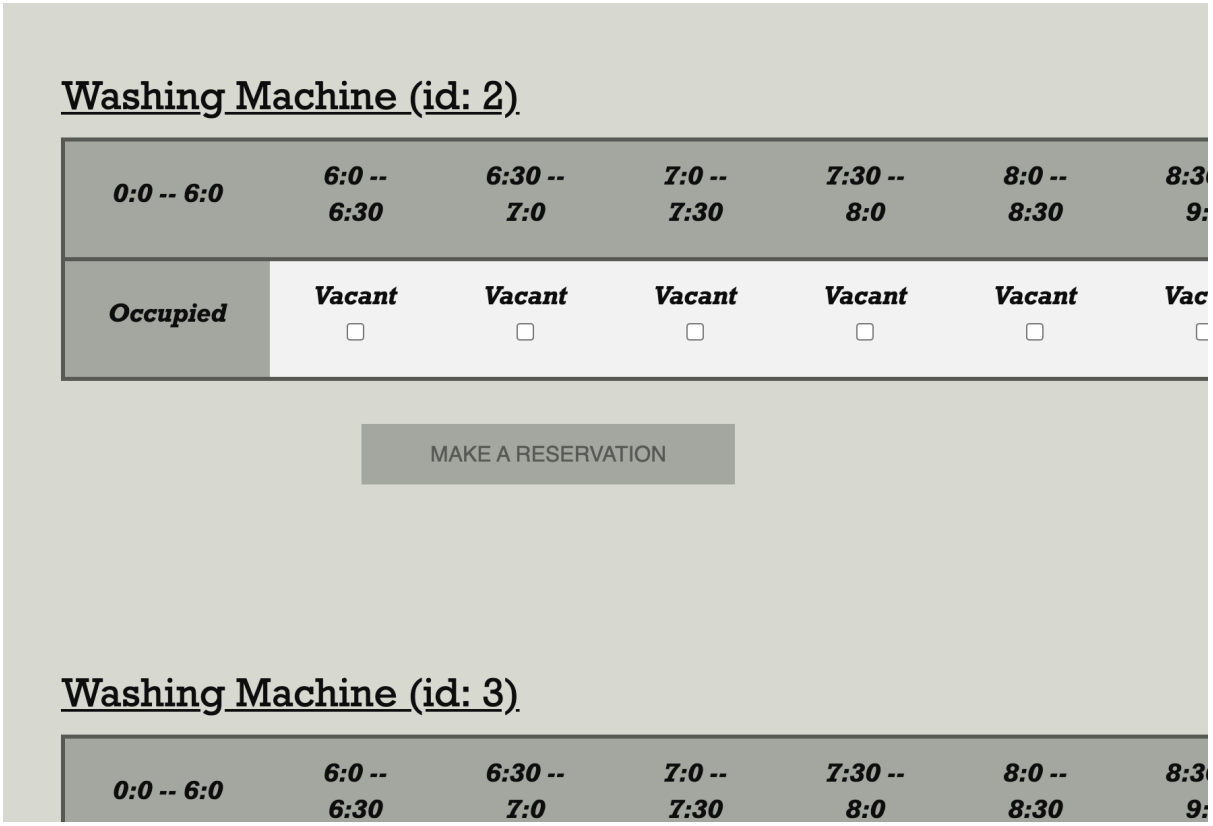
Instead of selecting starting time, finishing time and day manually, the user would need to tick the right boxes. That prevents many possible errors that can happen due to incorrect input of data by the user.

Data inputted by a user:

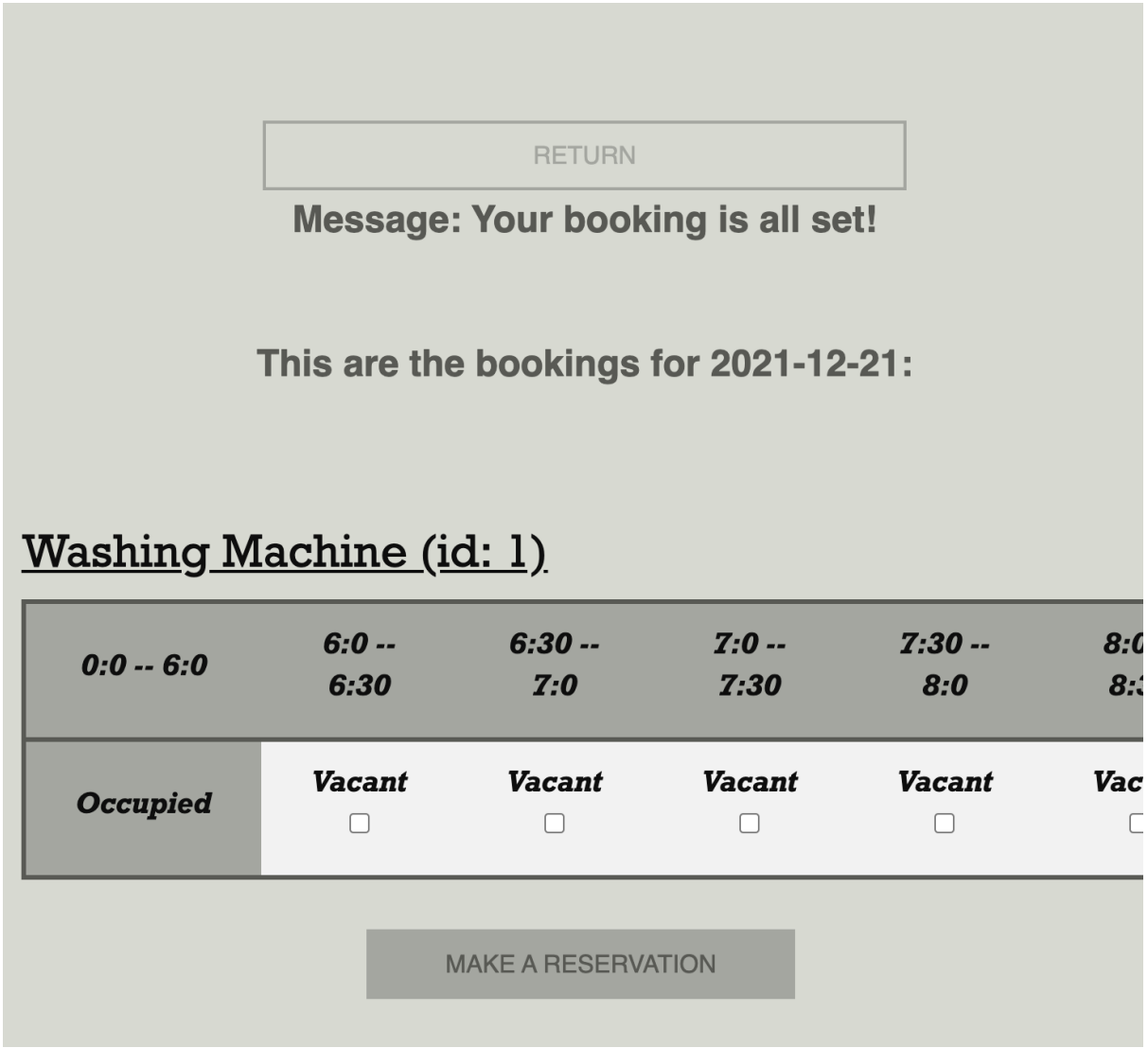
- Selected checklist buttons

Response from a computer:

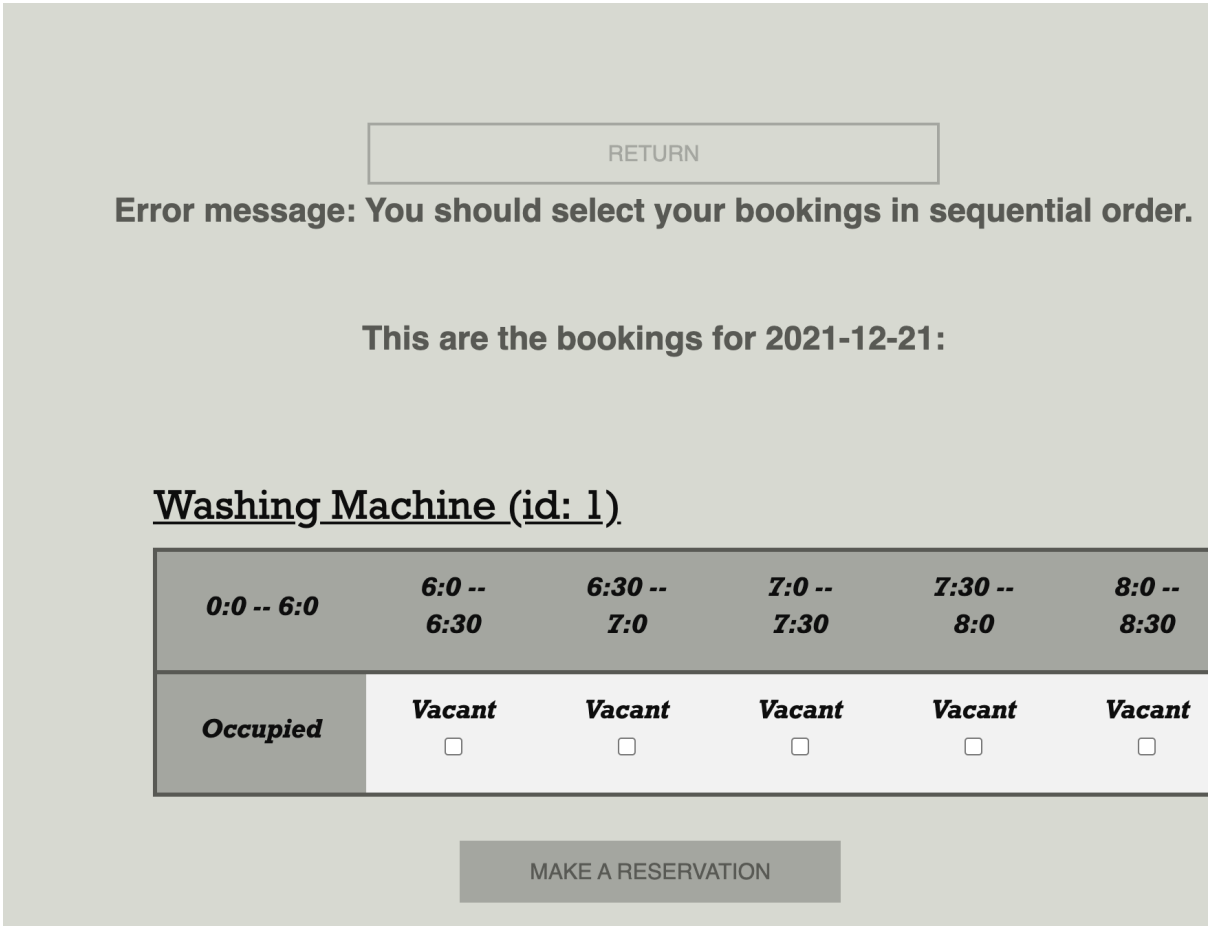
- Redirected to the booking page with a succession message on it if the validations were all passed.
- Redirected to the booking page with a failure message on it if failed to validate.



The booking form.



The booking page with a succession message on it.



The booking page with a failure message on it.

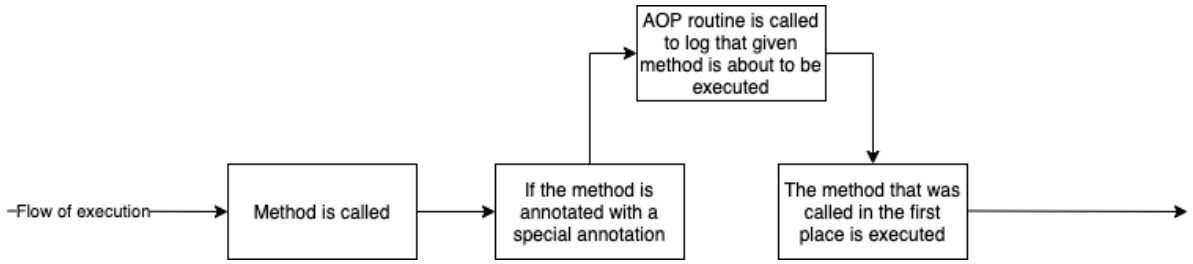
In terms of a design this solution is poorly optimised as the server must send a new web page to every single user, no matter what action they are performing. The right solution would be to create a JavaScript front-end for the users to interact with. JavaScript would allow users to display new pages on users' screens without sending a request to a server. When that is implemented, a web-product becomes more scalable as less operations are carried directly by the server itself.

However, due to time limitations, I will be unable to learn JavaScript well enough to create a fully-functioning front-end.

AOP

AOP - is a paradigm of programming that allows to insert functionality into specified methods without writing code inside these methods. That is useful for adding temporary (and even permanent) identical code into many methods simultaneously.

To visualise AOP paradigm, I have created the following visualisation:



AOP routine can be executed after the method was executed, or even around the method's execution.

The AOP routine (officially referred to as "aspects") can be called before/after/during execution of certain methods ("joipoints"). The set of joinpoints that trigger an aspect's execution is called pointcut.

I find it useful to configure pointcuts to custom annotations, as then adding and removing joinpoints from the influence of a certain aspect becomes simpler.

I find it useful to have the following aspects in my code:

validateArgumetsNotEmpty() aspect is called to check if none of the parameters of the method that is annotated with @NotEmptyArguments annotation are set to null. If some are, that information would be recorded into log files.

testExecution() aspect is responsible for logging that execution of certain methods was initiated and terminated. That is useful information for debugging to keep a track of what methods threw exceptions (which are logged in a special exception-handling class). This aspect would be initiated around the execution of any method annotated by @DebugMethod. Such a name is chosen as this aspect should only be executed if the logger is set to the debug mode.

Main reason why I want to implement AOP is to contribute to code efficiency - decrease the amount of code written.

Logging

Recording internal activity of the application is essential for many reasons, but the main ones are debugging the application and collecting data for future analysis.

It is necessary to log data related to invocation of methods, exceptions being thrown.

In Java there are different ways to log, but there exist dedicated tools for this task. Instead of using a basic text-file writer, I would be using a combination of two technologies: SLF4J and LOG4J.
LOG4J is an instrument that makes logging as simple as calling a .log() method.
SLF4J is a tool that acts as a medium between loggers and the code. Using SLF4J I would be able to switch to different loggers (like logback, or JDK logging).

In essence, all debug and error information would be stored in the following format:

```
2021-11-12 08:39:19 TRACE
[com.adtimokhin.controller.AuthController.index(AuthController.java:104)]: message
```

I would be using a rolling logger. That is a setup of a logger that has a maximum number of files it can create. When the first file is filled it would move to the second one and so on. When the last log file is filled, the data inside the oldest file would be erased and rewritten.

That helps to limit the amount of redundant data stored on the server.

As I have mentioned, LOG4J is an implementation of a logger and thus only set up is required. This configuration is listed in a separate file called log4j.properties and that is how it looks:

```
# Tells Log4j to use the rolling logger
log4j.appender.trace=org.apache.log4j.RollingFileAppender
# Tells where to store the log files
log4j.appender.trace.File=/Users/atimokhina/Desktop/washingDisasterSolver/log/trace.log
# Tells the file size of a single log file
log4j.appender.trace.MaxFileSize=10MB
# Tells the number of log files that would be created before rolling over existing data
log4j.appender.trace.MaxBackupIndex=10
# Tells the instrument to create a layout
log4j.appender.trace.layout=org.apache.log4j.PatternLayout
# Tells the pattern to use
# I use the following format: year-month-day hour:minute:second category [location that called
the logger]: message
log4j.appender.trace.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p [%l]: %m%n
# Tells what category to use
log4j.category.com.adtimokhin= TRACE, trace
```

Category of a logger is a so-called “level of logging”. All logging messages are split into 5 levels (trace, debug, info, warn, error). When you create a message in a log file you must indicate the level of logging to ease the navigation through the entries.

Depending on the category, only messages with the certain logging levels would be recorded in the log files. This table summarises this information:

Category selected	Will the message be recorded in the log file?					
		Trace	Debug	Info	Warn	Error
	Trace	✓	✓	✓	✓	✓
	Debug	✗	✓	✓	✓	✓
	Info	✗	✗	✓	✓	✓
	Warn	✗	✗	✗	✓	✓
	Error	✗	✗	✗	✗	✓

Saving Data About Previous Bookings

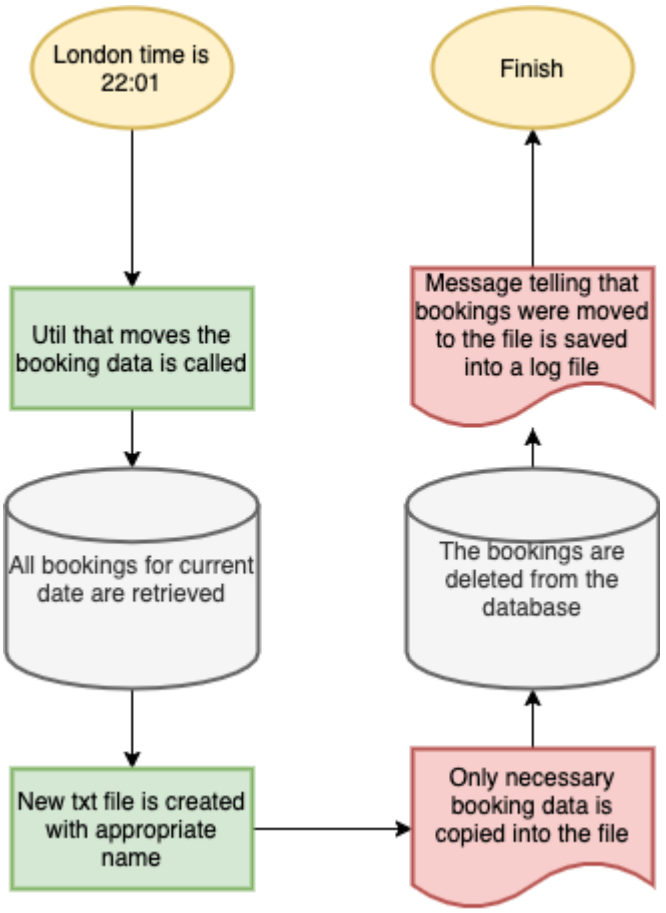
For future potential analysis I want to store data about bookings that are made during the week. However, storing this data inside the database with pending bookings is not a good practice - it would slow the process of retrieving useful information from the tables.

Instead every day, at a given time, all bookings for the current date would be removed from the table and stored inside text files. Each day gets its own file so that it is easier to manipulate data about particular dates (for example, collecting information about bookings on Tuesdays).

Text file was chosen as the format because it allows easier manipulation of data stored. In comparison to JSON, text files allow me to store only certain information about the bookings and not all of the fields (for example, I do not store ids of bookings). However, I used a structure similar to one in JSON to enable swift data manipulations.

Each file is a given name in the format: **booking_YYYY_MM_DD.txt**

Here is a flowchart that summarises the process of moving booking information into booking files:



The routine happens in a separate thread - thus asynchronously.

22:01 is not a random time. By 22:00 the laundry room is locked and no one is able to use it. This routine must be called while the room is not operating. I have chosen to call it at 22:01.

Example data that can be stored inside the booking_2021_10_20.txt file:

Booking{ userId=4, machineId=50, startDate='2021 10 20 09:20', endDate='2021 10 20 11:40'}
Booking{ userId=100, machineId=3, startDate='2021 10 20 18:00', endDate='2021 10 20 20:00'}
Booking{ userId=1, machineId=1, startDate='2021 10 20 10:00', endDate='2021 10 20 12:00'}

Each line stores a single booking entity.

Working with Time

Start and finishing times in the database are stored as strings in the format:

YYYY MM dd HH:mm

Unlike date datatype, the string stores only the symbols in the format above and no metadata (like timezone, milliseconds measurements).

Because I would be using strings to represent time, I would need a custom class for performing operations on these time-strings. I named this class DateFormatResolver as it resolves and performs operations on strings that are formatted as dates.

This class is used throughout my code because controllers, services and other utilities are required to perform tasks on dates. The list of functions in this class is large, and there is no point to go into too much depth on how every single of them operates. Intead, I would summarise the methods in the table below:

getDatePart()		
Input	Process	Output
• Date: string • Part of date for extraction: int	Part of the date should be equal to one of the options provided: YEAR, MONTH, DAY, HOUR, MINUTE. It is then extracted from the date	• Numerical value of the part requested. (int)
datesMatch()		
Input	Process	Output
• startDate: string • endDate: string	Compares whether HOUR and MINUTE values of the strings are equal. This methods assumes dates have the same YEAR, MONTH and DAY components	• Boolean value that indicates whether HOUR and MINUTE values of the strings are equal. (boolean)
isTimeBigger()		
Input	Process	Output
• startHour: int • startMinute: int • endHour: int • endMinute: int	Checks if the start time is larger than the end time (works if the DAY part of the dates is the same).	• Boolean value that indicates whether the start time is larger than the end time. (boolean)

isTimeBigger()		
Input	Process	Output
• startDate: string • endDate: string	Checks if the start time is larger than the end time (works if the DAY part of the dates is the same).	• Boolean value that indicates whether the start time is larger than the end time. (boolean)
resolveTimeForDate()		
Input	Process	Output
• hour: string • minute: string • date: string	Assigns hour and minute values to the date passed.	• Date with new HOUR and MINUTE settings (string)
extractYearMonthDay()		
Input	Process	Output
• date: string	Extracts a string which contains only YEAR, MONTH and DAY parts.	• New string
isDateBeforeAnother()		
Input	Process	Output
• dateOne: string • dateTwo: string	Compares all parts of the dates to see whether the dateOne happens in time before the dateTwo	• Boolean value that indicates whether the dateOne happens in time before the dateTwo. (boolean)
isDateBefore()		
Input	Process	Output
• dateOne: string • dateTwo: string	Compares YEAR, MONTH and DAY parts of the dates to see whether the dateOne happens in time before the dateTwo	• Boolean value that indicates whether the dateOne happens in time before the dateTwo. (boolean)
today()		
Input	Process	Output

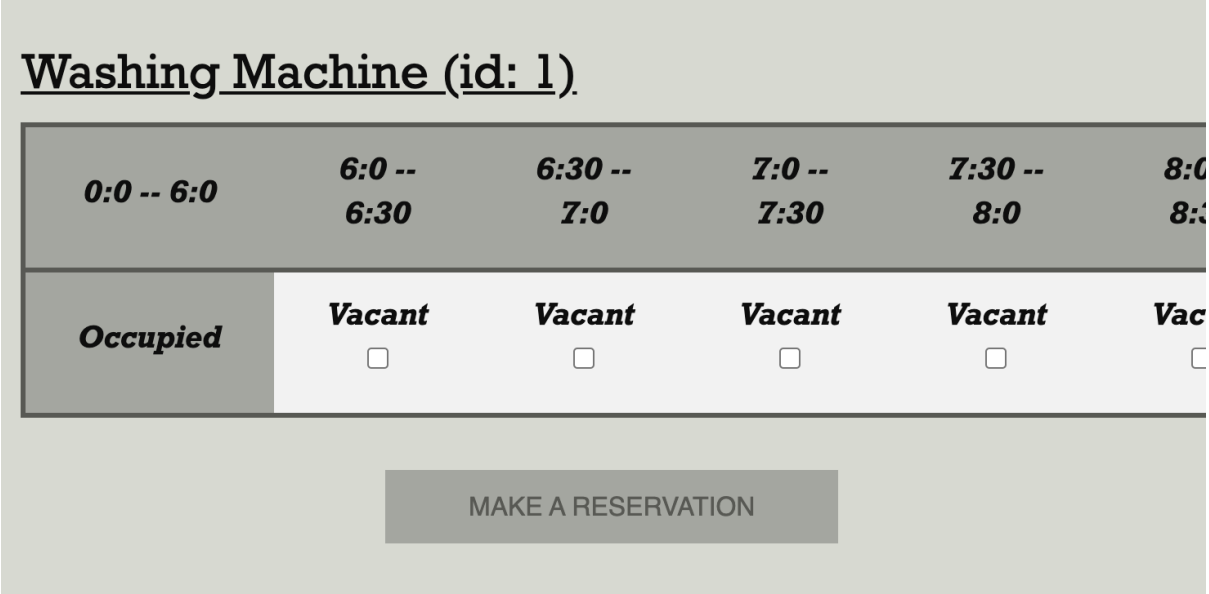
	Finds a string-date representation of the current time.	• String in format YYYY MM dd HH:mm that represents current moment in time (string)
onTheSameDay()		
Input	Process	Output
• timeOne: string • timeTwo: string	Compare the YEAR, MONTH and DAY components of the two dates to see whether the two dates point to the same day.	• Boolean value that represents whether the two dates point to the same day. (boolean)
areFarEnoughInTime()		
Input	Process	Output
• dateOne: string • dateTwo: string • distance: int	Compares the two dates to see whether the distance in time between them is greater than the distance passed as a parameter	• Boolean value that represents whether the distance in time between them is greater than the distance passed as a parameter. (boolean)
appropriateFormat()		
Input	Process	Output
• date: string	Checks whether the string passed follows the pattern: “YYYY MM dd HH:mm”	• Boolean value that represents whether the string passed follows the pattern: “YYYY MM dd HH:mm” (boolean)
appropriateFormatWithDash()		
Input	Process	Output
• date: string	Checks whether the string passed follows the pattern: “YYYY-MM-dd-HH:mm” This method is needed	• Boolean value that represents whether the string passed follows the pattern: “YYYY-MM-dd-HH

	because the date obtained from a page where the user selects time for the timetables is passed in the format given.	:mm” (boolean)
getWeekDays()		
Input	Process	Output
• date: string	Gets dates in the correct format for all seven days that happen on the same week as the date passed	• A list of seven dates
getMinutesBetweenDates()		
Input	Process	Output
• dateOne: string • dateTwo: string	Calculates the number of minutes that separate the two dates.	• A number of minutes that separate the two dates. (int)

Timetables For Machines

As it was discussed previously, booking times are saved as a string in format YYYY MM dd HH:mm (page 55). Such format is also explained by how the bookings are stored in the database.

In my design, the users can select only distinct time slots (of length 30 minutes) for their reservations. When user wants to select or simply view available time slots for their booking, they need to access the booking data in a form of a timetable like on the picture below:



The timetables should be separate for every machine and date.

When user sends request to view a timetable two actions might happen;

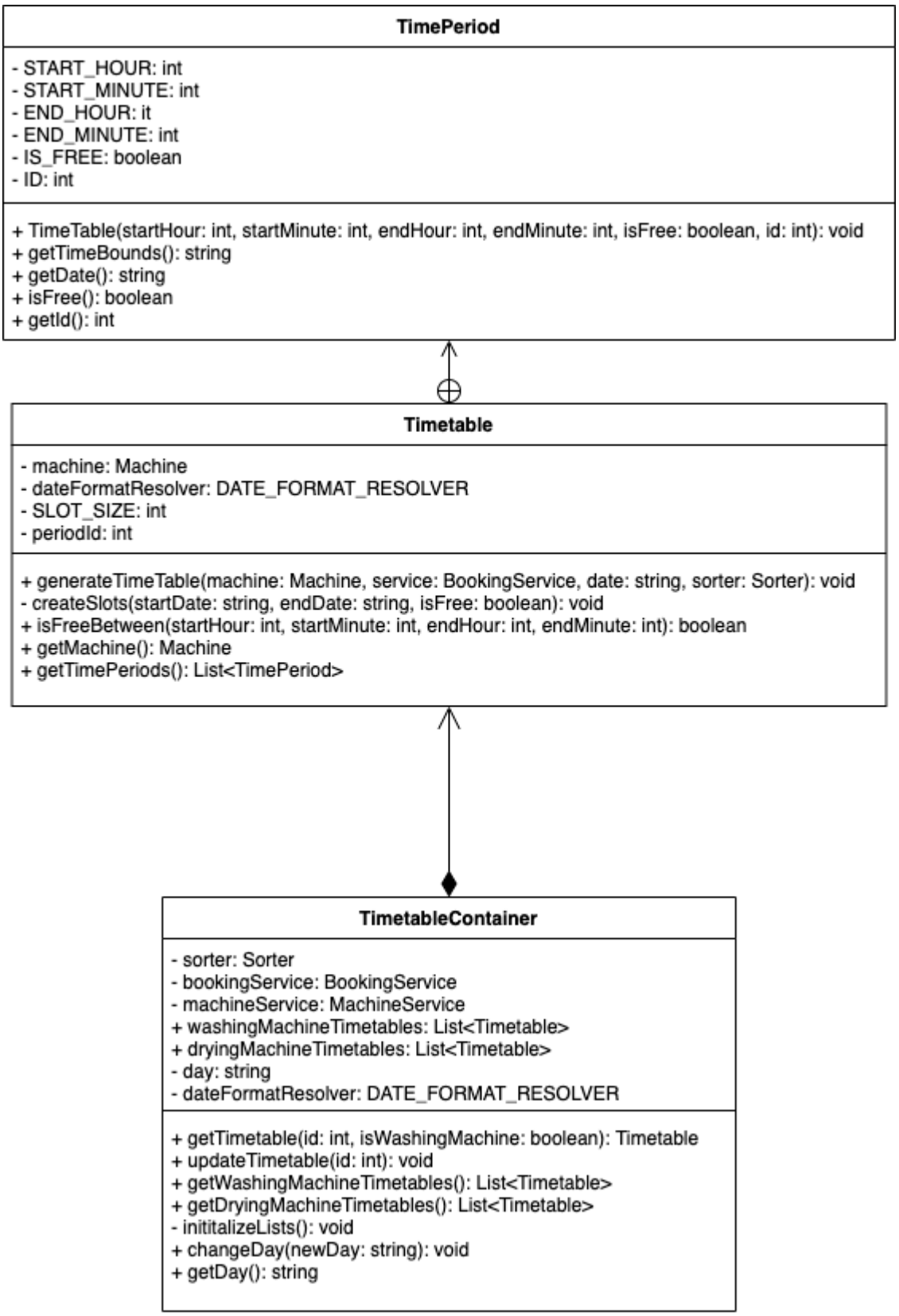
- New timetable for that machine and date would be generated
- Existing timetable should be returned

As the current system would operate in a single location, there is no point in selecting particular machines to generate timetables for. Instead, the user would have access to all timetables for that date.

When a user requests timetables for a date for which the timetables are not yet generated, all other timetables would be erased. That is done out of the reasons of memory efficiency. Storing timetables for dates that are not visited currently is redundant. Hence, dedicating the server's power to calculate timetables for 8 machines (each made out of 34 slots) seems more efficient.

The obvious problem occurs when multiple users wish to view timetables for different dates. The server would need to recalculate the tables multiple times to complete even a single request. That is an issue caused by me not using a front-end. If front-end I could simply send data about bookings on a particular date to the clients' machines which would create timetables themselves. But, due to limitations of my system, that is an issue that I must ignore.

The timetables would be stored in a special container that contains timetables for the given date. Here is a class diagram of classes that are responsible for the creation and storing timetables:


TimePeriod
- START_HOUR: int
- START_MINUTE: int
- END_HOUR: it
- END_MINUTE: int
- IS_FREE: boolean
- ID: int
+ TimeTable(startHour: int, startMinute: int, endHour: int, endMinute: int, isFree: boolean, id: int): void
+ getTimeBounds(): string
+ getDate(): string
+ isFree(): boolean
+ getId(): int
Timetable
- machine: Machine
- dateFormatResolver: DATE_FORMAT_RESOLVER
- SLOT_SIZE: int
- periodId: int
+ generateTimeTable(machine: Machine, service: BookingService, date: string, sorter: Sorter): void
- createSlots(startDate: string, endDate: string, isFree: boolean): void
+ isFreeBetween(startHour: int, startMinute: int, endHour: int, endMinute: int): boolean
+ getMachine(): Machine
+ getTimePeriods(): List<TimePeriod>
TimetableContainer
- sorter: Sorter
- bookingService: BookingService
- machineService: MachineService
+ washingMachineTimetables: List<Timetable>
+ dryingMachineTimetables: List<Timetable>
- day: string
- dateFormatResolver: DATE_FORMAT_RESOLVER
+ getTimetable(id: int, isWashingMachine: boolean): Timetable
+ updateTimetable(id: int): void
+ getWashingMachineTimetables(): List<Timetable>
+ getDryingMachineTimetables(): List<Timetable>
- inititalizeLists(): void
+ changeDay(newDay: string): void
+ getDay(): string

TimePeriod is an inner class that represents a single time slot that can either be available or occupied. That class is made out of methods that perform only functions of getters or return a prettified data that is stored in the object.

I want to focus on how the timetable is actually generated by writing a pseudocode:

```
PROCEDURE generateTimetable(machine, bookingService, date, sorter)
        // sorter is a class that sorts data.
        bookings ← sorter.sortBookings(sorter.clearData(
        bookingService.getBookingsForMachine(machine)), date)

        size ← bookings.length()
        timePeriods ← []
        timePeriods.add(NEW TimePeriod(0,0,6,0, FALSE, -1)) // an interval between 0:00 -
6:00
                                                            // cannot be accessed by anyone.
        FOR i ← 1 TO (size + 1)
                currentBooking ← bookings[i]
                startDate ← current.getStartDate()
                endDate ← current.getEndDate()
                oldEnd ← timePeriods[timePeriods.length()].getDate() // oldEnd - endDate of
the
                                                                    // previous booking in
                                                                    the // timePeriods list.
                IF DATE_FORMAT_RESOLVER.datesMatch(startDate, oldEnd) THEN
                        CALL createSlots(oldEnd, startDate, TRUE)
                END IF
                CALL createSlots(startDate, endDate, FALSE)
        END FOR
        oldEnd ← timePeriods[timePeriods.lenght()].getDate()
        startDate ← "0000 00 00 22:00" // dummy date with the time when the laundry room
                                       // closes
        IF DATE_FORMAT_RESOLVER.datesMatch(startDate, oldEnd) THEN
                CALL createSlots(oldEnd, startDate, TRUE)
        END IF
        timePeriods.add(NEW TimePeriod(22,0,23,59, FALSE, -1))
END PROCEDURE

PROCEDURE createSlots(begin, finish, isFree)
        // HOUR and MINUTE are constants that represent part of the string-date of hour and
        // minute respectively.
        startHour ← DATE_FORMAT_RESOLVER.getDatePart(begin, HOUR)
        startMinute ← DATE_FORMAT_RESOLVER.getDatePart(begin, MINUTE)

        hourDiff ← DATE_FORMAT_RESOLVER.getDatePart(finish, HOUR) - startHour
        minuteDiff ← DATE_FORMAT_RESOLVER.getDatePart(finish, MINUTE) - startMinute

        timeSlots ← (hourDiff * 60 + minuteDiff) / SLOT_SIZE

        // SLOT_SIZE is a constant that represents a length of a single slots in minutes
(30
        // minutes in this case).

        finishMinute ← -1
        finishHour ← startHour

        FOR i ← 1 TO timeslots + 1
                finishMinute ← startMinute + SLOT_SIZE
                IF finishMinute = 60 THEN
                        finishMinute ← 0
                        finishHour ← finishHour + 1
                END IF
```

```
                timePeriods.add(NEW
TimePeriod(startHour,startMinute,finishHour,finishMinute,
                                isFree, periodId++))
                startHour ← finishHour
                startMinute ← finishMinute
        END FOR
END PROCEDURE
```

Timetables would be used mostly for decorative purposes in FTL pages (to display data in a way that enables simplified user-computer interactions).

Email API

In order to verify that email exists, users must receive a letter with verification code. In order to generate that code the following algorithm might be implemented:

```
// This function is initially called to generate the token.
FUNCTION generate()
        // userService is a service class that works with the ORM Representation of User in
        // the database.
        tokens ← userService.findAllEmailTokens()
        IF tokens.length() = TOKEN_LENGHT ** symbols.length THEN
                // TOKEN_LENGHT is a number constant for the length of the token
                // symbols is a list of all symbols that can be used to create a token
                RETURN “”
        ENDIF
        RETURN CALL generateSymbol(symbols, 1, tokens, “”)
END FUNCTION

FUNCTION generateSymbol(symbols, position, viableTokens, currentToken)
        IF position = TOKEN_LENGHT THEN
                RETURN currentToken
        END IF
        symbols ← shuffle(symbols) // shuffle() randomises the order of elements in the
array
        FOR i ← 1 TO (symbols.length() + 1)
                charToTry ← symbols[i]
                matchDigit ← []
                FOR j ← 1 TO (viableTokens.length() + 1)
                        IF viableTokens[j][position] = charToTry THEN
                                matchDigit.append(viableTokens[j])
                        END IF
                END FOR
                IF matchDigit.length() = 0 THEN
                        currentToken ← currentToken + charToTry
                        RETURN currentToken + CALL generateRest(currentToken.length, symbols)
                ELSE
                        IF matchDigit.length() <> symbols.length ** (TOKEN_LENGTH - pos)THEN
                                potentialToken ← CALL generateSymbol(symbols, pos + 1,
                                matchDigit, currentToken + charToTry)// recursion

                                IF potentialToken <> NULL THEN
                                        RETURN potentialToken
                                END IF
```

```
                    END IF
                END IF
        END FOR
        IF pos = 1 THEN
                THROW TokenOverflowException // Exception that indicates that there are no
more                                                    // possible tokens to use.
        END IF
        RETURN NULL
END FUNCTION

FUNCTION generateRest(startPosition, symbols)
        remainingToken ← ""
        FOR i ← 1 TO (TOKEN_LENGTH - startPosition + 1)
                remainingToken ← remainingToken + symbols[i]
        END FOR
        RETURN remainingToken
END FUNCTION
```

My application should have no more than 70 active users in a year. Thus, email verification tokens do not need to be long. I have chosen the length of these tokens to be 5 digits. That gives 10^5 possible tokens.

In the worst case scenario only 70 of those would be used, meaning that there is a chance of guessing the code equal to 0.07%.

Having only 0.07% of possible tokens be in use means that it is easy to generate a unique token using a random number generating function (with the range of 00000 to 99999).

In addition, after the user has activated their email, the token is disassigned from the user and can be reused if needed. That means that the total chance of randomly generating a unique token is even higher than 99.3%.

The pseudocode for the method above, however, is designed to provide optimisation on scale. This is a quick summary of how that code achieves that optimization:

- Each character is generated on its own.
- After generation, it is calculated how many existing tokens have that exact digit on that spot.
- All tokens that do have that digit on the given spot would be stored in the array. Others would be discarded.
- If the number of remaining tokens is zero - the new token would be auto-filled with random digits until there are 5.
- If the number of remaining tokens is 10^n, where n is 5 — (position of the current character), then the previous digit is changed and the algorithm continues running.
- If the number of remaining tokens is between 0 and 10^n, then the same algorithm is repeated for the next digit.

Complexity of such an algorithm is O(n log(n)). Due to arguments described above, this algorithm is not the optimal solution, because the chance of recalculating the token is 0.07%. However, the advantages of this algorithm are the following:

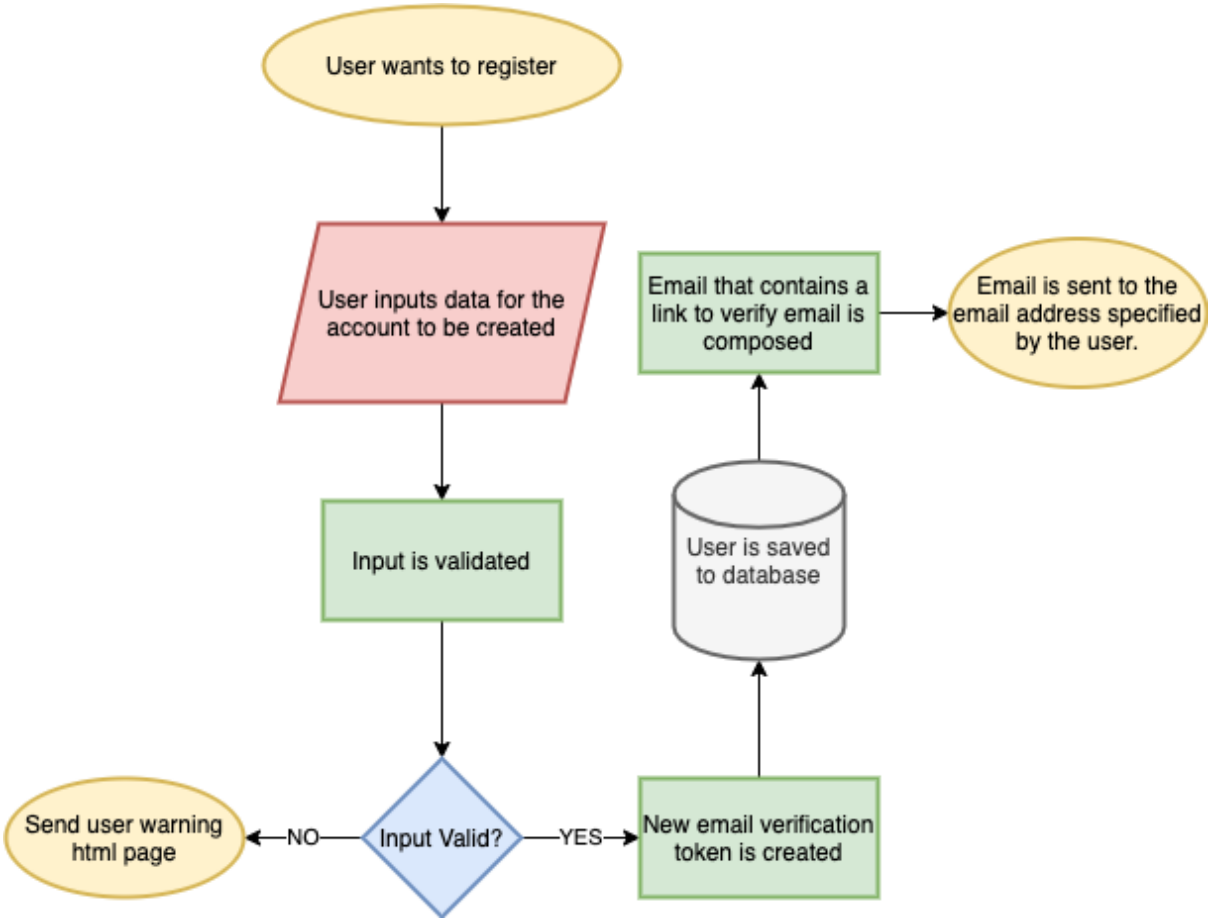
- It increases the efficiency if there are many more tokens. Due to the fact that this application might not only be used in my dormitory, but in other schools, making sure that the process of generating tokens is swift is essential to decrease the amount of work the server must do.
- Implementation of the algorithm described above reduces the risk that the server would generate tokens in use multiple times. That means that users would not need to wait for minutes (depending on the number of tokens) to get their unique token.
- This implementation limits the randomness of the process as far as possible.

After generating the token, it must be sent to the user. I use the JavaMail API for achieving such results. In order to send the email, I must first provide configuration to the API. I store it in a file email.properties so that I can update the connection data easily. This file looks like this:

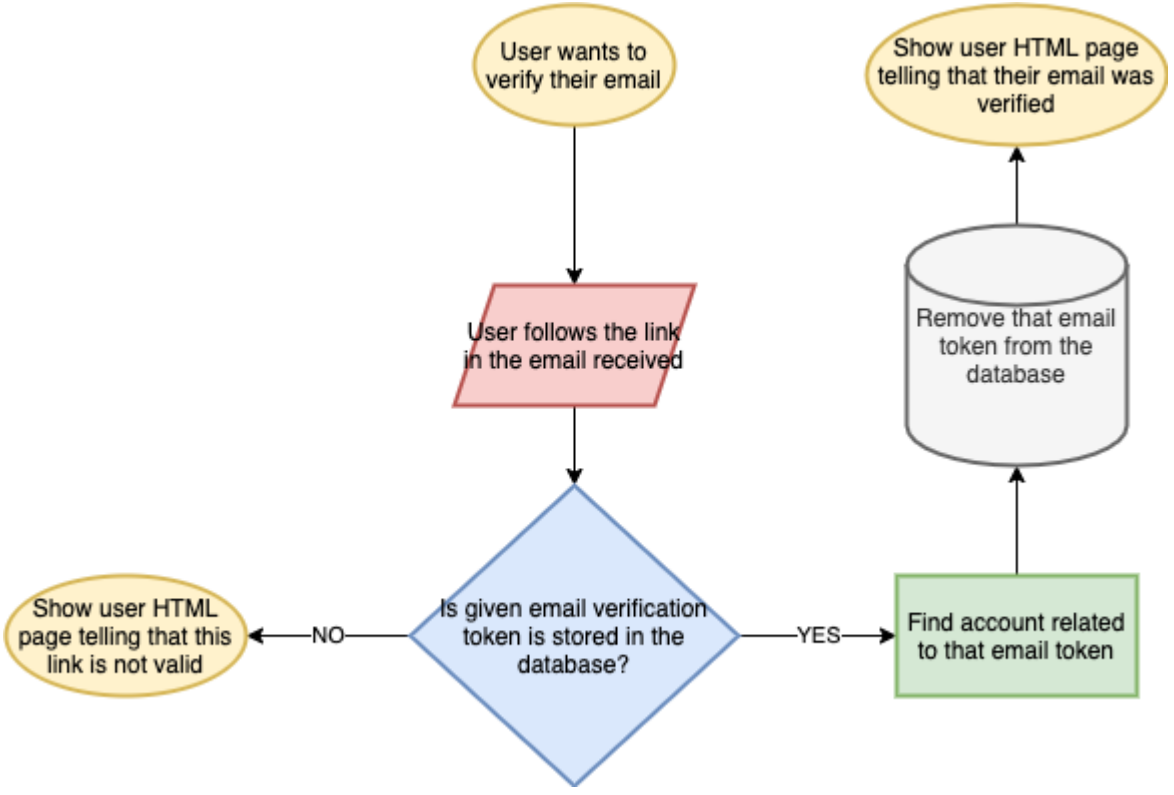
```
mail.host=smtp.yandex.ru
mail.username=kolestnitsa-maria@yandex.ru
mail.password=
mail.port=465
mail.protocol=smtps
mail.smtp.auth=true
mail.smtp.ssl.enable=true
```

I use the SMTP server provided by Yandex to send my verification emails. The server uses port 465 and protocol SMTPs. I use SSL to send secure requests to the server.

The connection is established before the launch of the actual web-application. Here is a flowchart that illustrates a process of sending user their verification letter:



And here is the flowchart of how the user can validate their email:



To understand why removing the token marks the email as valid navigate to the validation section.

Data Validation

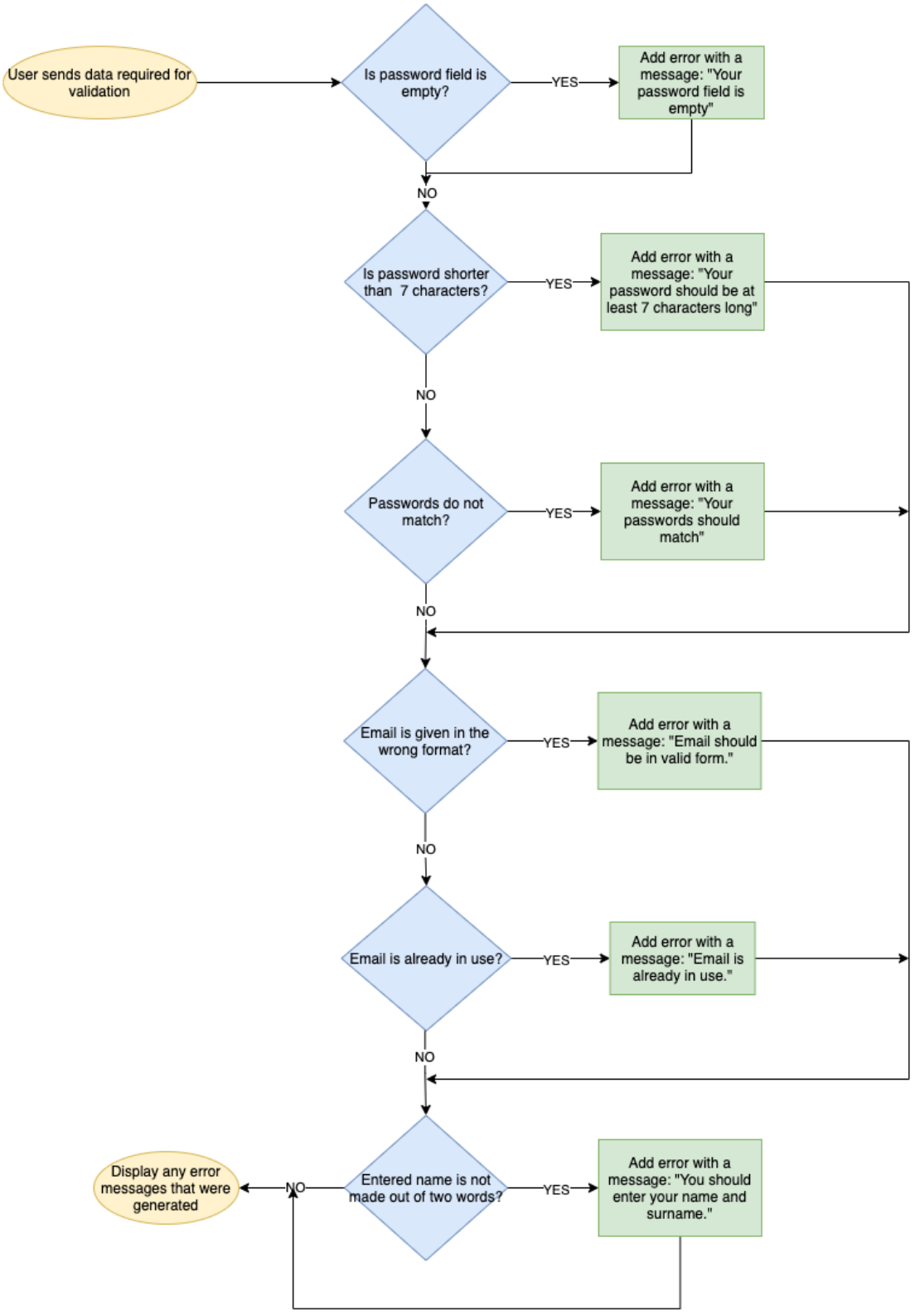
In the analysis section I have mentioned the validations that would be needed prior to adding objects into the database.

After discussing MVC and the models that I would be using in my application, I believe it is essential to outline the process of validation that I would follow.

The users of the software would be able to add objects of three classes: User, Booking, Report.

User Validation:
This validation would require the following input:

- Password - represents password that user would use to verify the ownership of the account.
- PasswordTwo - re-entered password. Used to validate that the user has not made a mistake while typing their password for the first time.
- Email - represents email of the user.
- Name - used to address the user in the application.



By “correct email format” I mean an email that follows the following regular expression:

```
(A-Z | a-z | 0-9 | '+' | '_' | '.' | '-')+ '@' (A-Z | a-z | 0-9 | '-')+ '.' (A-Z | a-z | 0-9 | '-')+
```

To validate whether the email is not yet in use, we need to search for a user with the email entered in the database. If there is none, then the email is not in use.

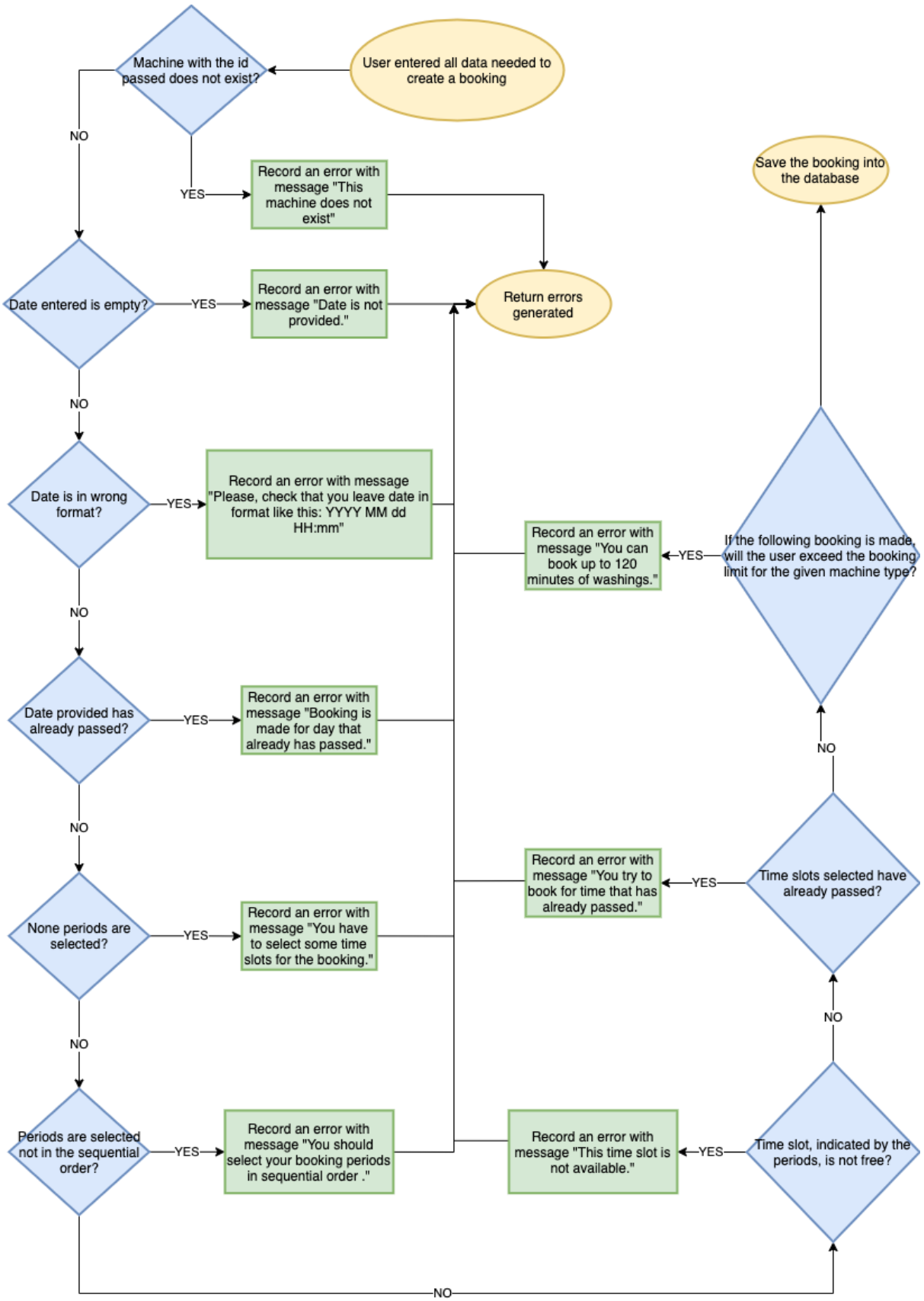
Here is a pseudocode of the validation process outline in the flowchart:

```
FUNCTION validate(user, passwordTwo) // user is an object that contains all user-relevant
       // information that requires validation. PasswordTwo is a string that contains a
       // confirm  password for the user.password
       errors ← [] // list of errors that occur during the validation
       IF user.password = "" THEN
              errors.append( "Your password field is empty.")
       ELSE
              IF user.password.length < MIN_PASSWORD_LENGTH THEN
                     // MIN_PASSWORD_LENGTH is constant that indicates the minimal
                     // size of the password
                     errors.append( "Your password must be at least " +
                     MIN_PASSWORD_LENGHT + " characters long.")
              ELSE IF user.password <> passwordTwo THEN
                     errors.append( "Your passwords do not match.")
              END IF
       END IF

       emailRegex ← (A-Z | a-z | 0-9 | '+' | '_' | '.' | '-')+ '@' (A-Z | a-z | 0-9 |
'-')+ '.' (A-Z | a-z | 0-9 | '-')+ // regular expression that matches the email pattern
       IF user.email DOES NOT MATCH emailRegex THEN
              errors.append("Email is in invalid form.")
       ELSE
              IF userService.getUserByEmail(user.email) <> NULL THEN
                     errors.append("This email is already in use.")
              END IF
       END IF

       IF user.name = "" THEN
              errors.append("You must enter your name.")
       ELSE IF user.name.split(" ").length <> 2 THEN
              errors.append("You should enter your name and surname.")
       END IF

       RETURN errors // receiving end will check if there are any errors inside the list.
If
                     // there are none, then the validation is passed.

END FUNCTION
```

Booking Validation:

This validation would require the following input:

- MachieId - id of machine that would be associated with the booking.
- Date - String in format YYYY MM dd HH:mm. It is used to set year, month and day parts of the start and end dates for the booking.
- MachineType - should match one of the machine types that the system can recognise (currently limited to washing and drying machine types). This input is needed to then update appropriate timetables.
- Periods - time slots selected during which the booking would be active.


User entered all data needed to create a booking
Machine with the id passed does not exist?
YES
Record an error with message "This machine does not exist"
NO
Date entered is empty?
YES
Record an error with message "Date is not provided."
Return errors generated
NO
Date is in wrong format?
YES
Record an error with message "Please, check that you leave date in format like this: YYYY MM dd HH:mm"
NO
Date provided has already passed?
YES
Record an error with message "Booking is made for day that already has passed."
NO
None periods are selected?
YES
Record an error with message "You have to select some time slots for the booking."
NO
Periods are selected not in the sequential order?
YES
Record an error with message "You should select your booking periods in sequential order ."
NO
Time slot, indicated by the periods, is not free?
YES
Record an error with message "This time slot is not available."
NO
Time slots selected have already passed?
YES
Record an error with message "You try to book for time that has already passed."
NO
If the following booking is made, will the user exceed the booking limit for the given machine type?
YES
Record an error with message "You can book up to 120 minutes of washings."
NO
Save the booking into the database

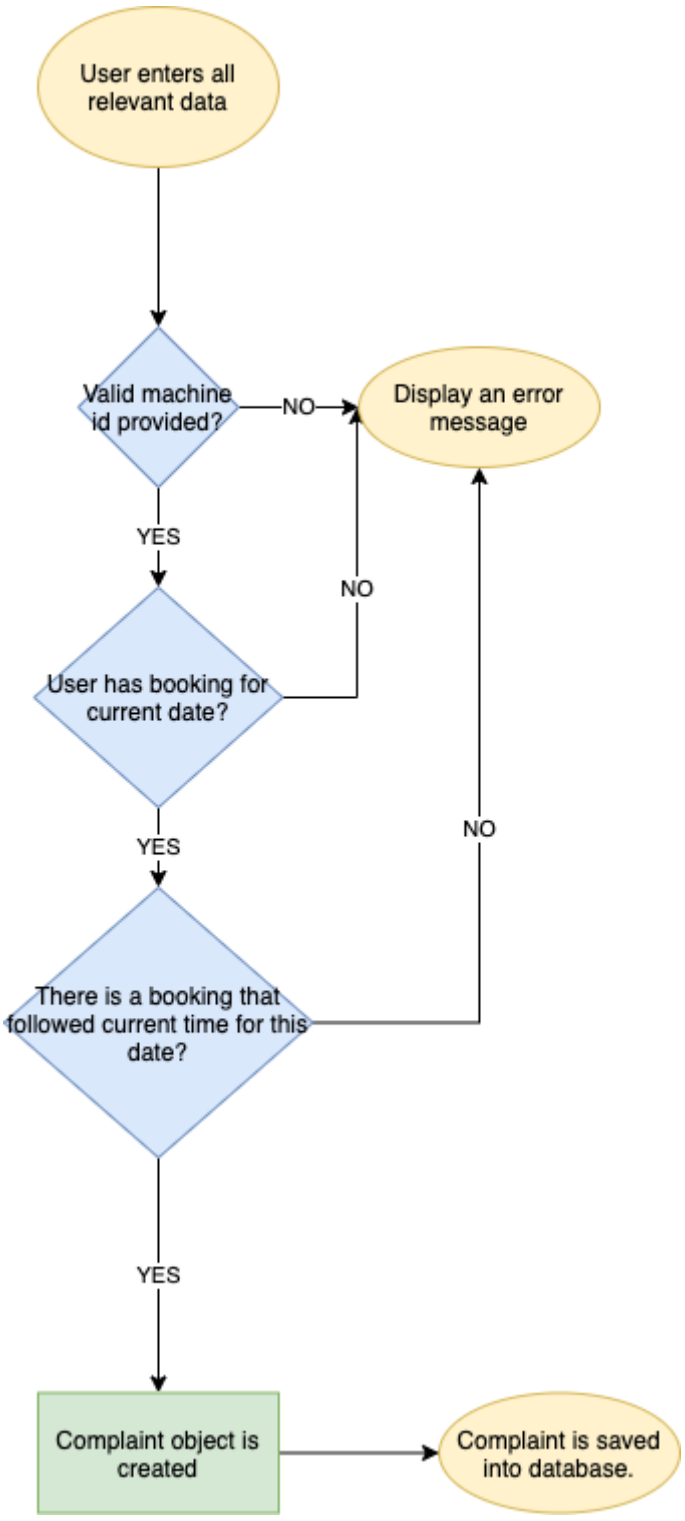
“Correct data format” is a format of a string that follows: “YYYY MM dd HH:mm”. This validation is necessary against people that would make requests through custom forms - users of my application would not enter data manually.

“Time slots selected have already passed?” Means that some time periods have already passed. That is different from the “Correct data format”.

Complaint Validation:

This validation would require the following input:

- MachieId - id of machine that would be associated with the complaint.


User enters all relevant data
Valid machine id provided?
NO
Display an error message
YES
NO
User has booking for current date?
YES
NO
There is a booking that followed current time for this date?
YES
Complaint object is created
Complaint is saved into database.

Requests

Here is a list of all requests that users can make to the server and a brief description what would the server do in response:

Request	HTTP Method	Action
account/bookings	GET	Returns a page that contains all bookings (that are currently active) made by the requesting user.
account/	GET	Returns a page that welcomes a user to their account section of the website.
login/	GET	Returns a page that user may fill in to request to login on the website
login/process	POST	Validates user's input and redirects the user to the main page on successful validation pass. Otherwise the login page with an error message is returned.
sign_up/	GET	Returns a page that user may fill in to request to register on the website
sign_up/	POST	Validates user's input and saves the user to the database on successful validation pass. Otherwise the login page with an error message is returned.
/	GET	Returns an index page.
verify/{token}	GET	Returns the user a page that tells whether the token passed has successfully validated their account. When this request is sent, the token would also be removed from the database.
booking/actions	GET	Returns a page that allows a user to navigate on the website.
booking/view/{type}	GET	Returns a page that allows a user to make a booking and view timetables (for machines of particular type).
choose/date/{type}	GET	Returns a page with the timetables but with updated timetables.
cancel/{type}	DELETE	Deletes a booking of choice from the database (if validations have been passed successfully).

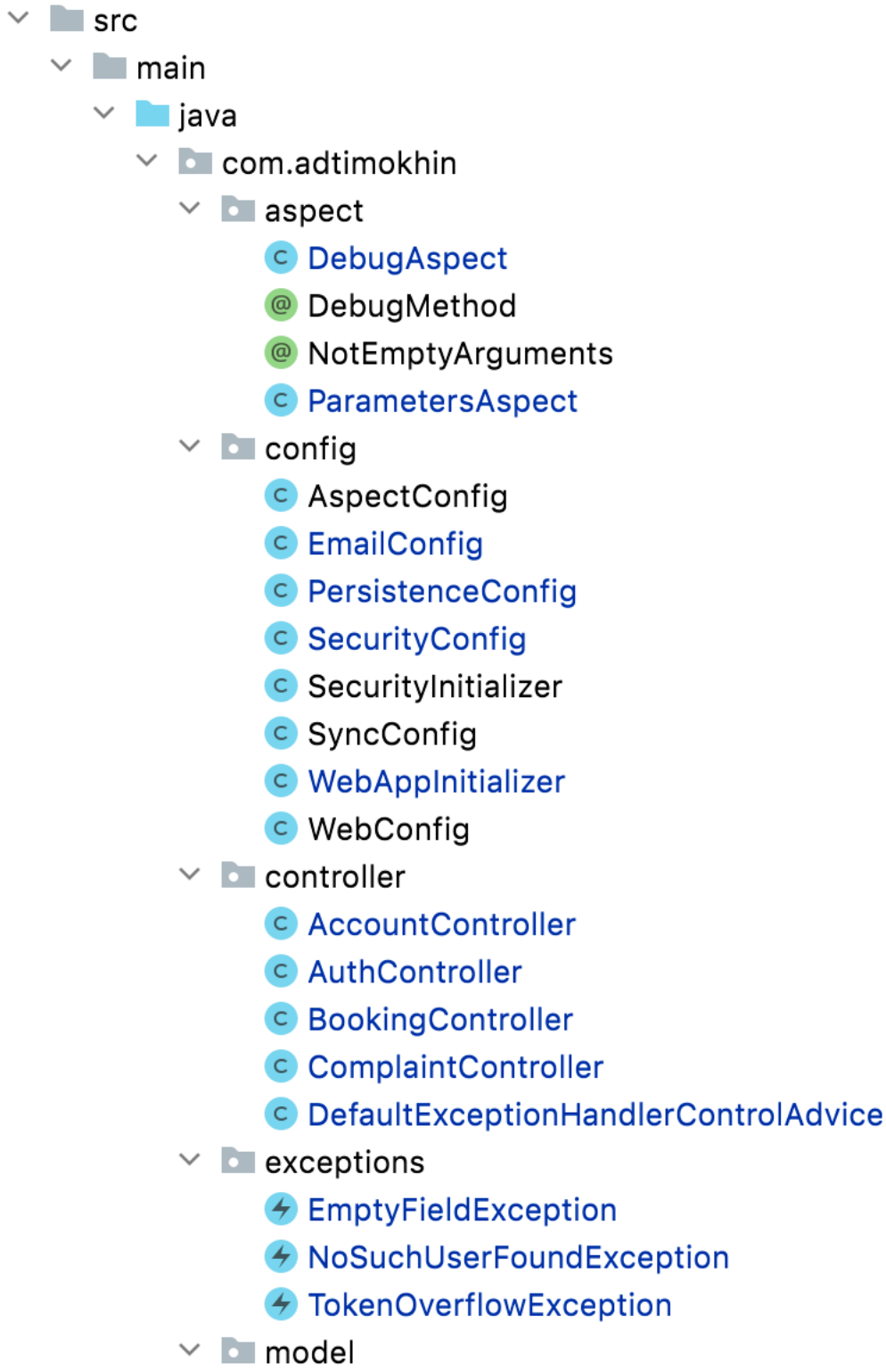
add/{type}	PUT	Saves a booking to the website if validations are passed. Otherwise an error message would be returned.
complaint/add	PUT	Adds a complaint if all validations are passed.
complaint/	GET	Returns a page that displays current complaints (necessary for the staff to know that there are some problems that require fixing).

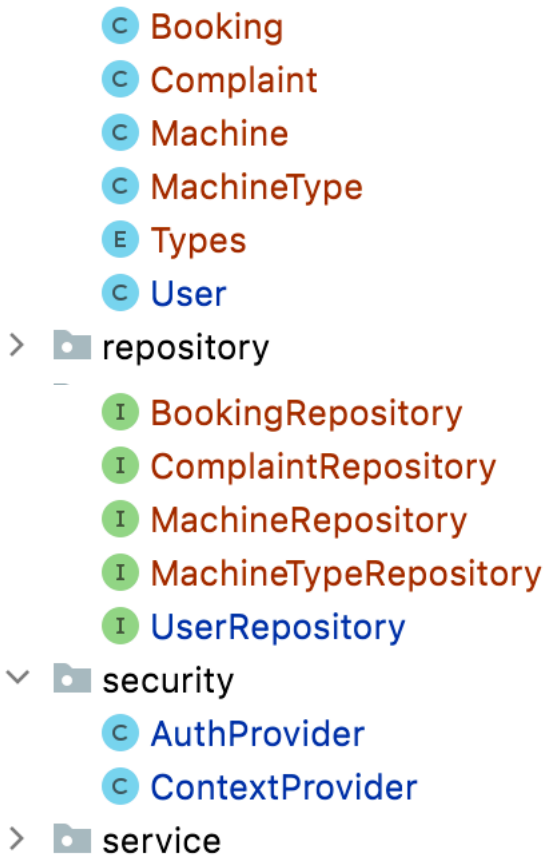
These requests are in no particular order.

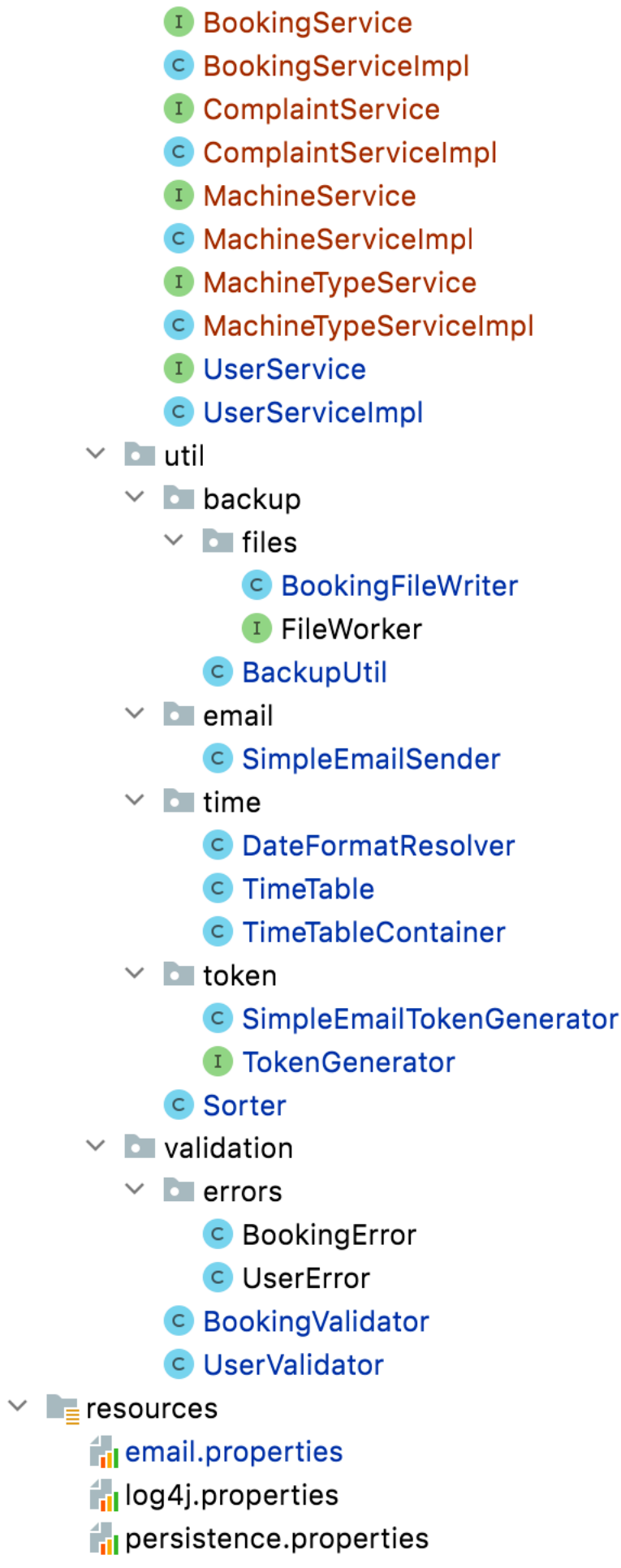
In the code you would not see that I have used PUT, DELETE HTTP methods. The reason for that is that html forms support only the POST method.

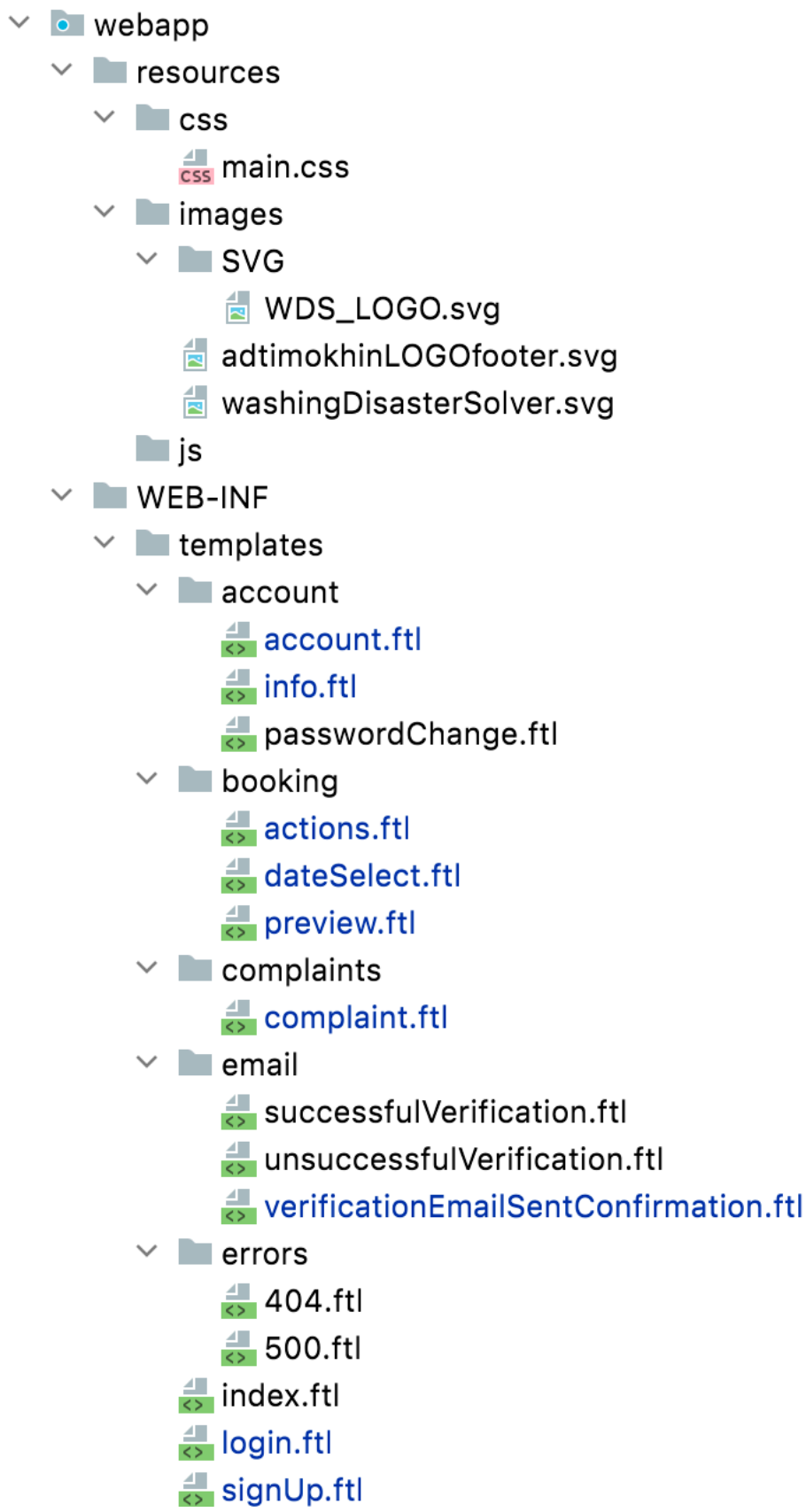
TECHNICAL SOLUTION SECTION

Files Hierarchy




Booking
Complaint
Machine
MachineType
Types
User
repository
BookingRepository
ComplaintRepository
MachineRepository
MachineTypeRepository
UserRepository
security
AuthProvider
ContextProvider
service


BookingService
BookingServiceImpl
ComplaintService
ComplaintServiceImpl
MachineService
MachineServiceImpl
MachineTypeService
MachineTypeServiceImpl
UserService
UserServiceImpl
util
backup
files
BookingFileWriter
FileWorker
BackupUtil
email
SimpleEmailSender
time
DateFormatResolver
TimeTable
TimeTableContainer
token
SimpleEmailTokenGenerator
TokenGenerator
Sorter
validation
errors
BookingError
UserError
BookingValidator
UserValidator
resources
email.properties
log4j.properties
persistence.properties


webapp
resources
css
main.css
images
SVG
WDS_LOGO.svg
adtimokhinLOGOfooter.svg
washingDisasterSolver.svg
js
WEB-INF
templates
account
account.ftl
info.ftl
passwordChange.ftl
booking
actions.ftl
dateSelect.ftl
preview.ftl
complaints
complaint.ftl
email
successfulVerification.ftl
unsuccessfulVerification.ftl
verificationEmailSentConfirmation.ftl
errors
404.ftl
500.ftl
index.ftl
login.ftl
signUp.ftl

Code:

DebugAspect:

```
package com.adtimokhin.aspect;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Arrays;

/**
 * @author adtimokhin
 * 18.10.2021
 **/

/**
 * A class that logs execution of methods using AOP
 */

@Aspect
public class DebugAspect {

    private final static Logger logger = LoggerFactory.getLogger(DebugAspect.class);


    @Pointcut("@annotation(DebugMethod)")
    public void debugExecution() {
    }


    /**
     * Logs execution of methods annotated with {@link DebugMethod} annotation.
     *
     * @param jp {@link ProceedingJoinPoint}
     * @throws Throwable if something goes wrong.
     */
    @Around("debugExecution()")
    public void testExecution(ProceedingJoinPoint jp) throws Throwable { // Objective 11.3

        String methodName = jp.getSignature().getName();
        String arguments = Arrays.toString(jp.getArgs());

        logger.debug("About to execute method {} with parameters {}", methodName, arguments);

        jp.proceed();

        logger.debug("Finished executing method {}", methodName);
    }

}
```

DebugMethod:

```
package com.adtimokhin.aspect;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * @author adtimokhin
 * 18.10.2021
 **/

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface DebugMethod {
}
```

NotEmptyArguments:

```
package com.adtimokhin.aspect;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * @author adtimokhin
 * 25.10.2021
 **/

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface NotEmptyArguments {
}
```

ParametersAspect:

```
package com.adtimokhin.aspect;

import com.adtimokhin.exceptions.EmptyFieldException;
import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.ArrayList;
import java.util.List;

/**
 * @author adtimokhin
 * 25.10.2021
 **/

@Aspect
public class ParametersAspect {

    private final static Logger logger = LoggerFactory.getLogger(ParametersAspect.class);


    @Pointcut("@annotation(NotEmptyArguments)")
    public void checkArguments() {
    }

    /**
     * Checks if parameters passed into a {@link JoinPoint} annotated with {@link
NotEmptyArguments} is null.
     *
     * @param joinPoint a method annotated with {@link NotEmptyArguments}.
     * @throws Throwable is something goes wrong.
     */
    @Before("checkArguments()")
    public void validateArgumentsNotEmpty(JoinPoint joinPoint) throws Throwable {
        Object[] args = joinPoint.getArgs();
        String className = joinPoint.getSignature().toString();
        List<Integer> nulls = new ArrayList<>();
        List<Integer> empties = new ArrayList<>();


        for (int i = 0; i < args.length; i++) {
            if (args[i] == null) {
                nulls.add(i);
                continue;
            }
            Object argument = args[i];
```

```
            if (argument.getClass().equals(String.class)) {
                String arg = (String) argument;
                if (arg.isEmpty()) {
                    empties.add(i);
                }
            }
        }

        if (nulls.size() != 0 || empties.size() != 0) {

            StringBuilder nullSB = new StringBuilder();
            StringBuilder emptySB = new StringBuilder();

            for (int i :
                    nulls) {
                nullSB.append(i);
                nullSB.append(", ");
            }

            for (int i :
                    empties) {
                emptySB.append(i);
                emptySB.append(", ");
            }

            logger.error("Class {} was invoked with insufficient arguments. " +
                    "Parameters at the following positions were null: {} " +
                    "and the following were Strings without any data in them: {} ", className,
nullSB, emptySB); // Objective 11.3

            throw new EmptyFieldException(className);
        }
    }


}
```

AspectConfig:

```
package com.adtimokhin.config;

import org.springframework.context.annotation.Configuration;

/**
 * @author adtimokhin
 * 18.10.2021
 **/

@Configuration
public class AspectConfig {
}
```

EmailConfig:

```
package com.adtimokhin.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;

import java.util.Properties;

/**
 * @author adtimokhin
 * 17.08.2021
 **/

@Configuration
@PropertySource("classpath:email.properties")
public class EmailConfig {
```

```
    @Value("${mail.host}")
    private String host;

    @Value("${mail.username}")
    private String username;

    @Value("${mail.password}")
    private String password;

    @Value("${mail.port}")
    private int port;

    @Value("${mail.protocol}")
    private String protocol;

    @Value("${mail.smtp.auth}")
    private String auth;

    @Value("${mail.smtp.ssl.enable}")
    private String sslEnable;


    /**
     * Creates a {@link Bean} of {@link JavaMailSender} that would be used to send API requests
to send emails.
     * @return a {@link Bean} of {@link JavaMailSender}
     */
    @Bean
    public JavaMailSender getJavaMailSender(){

        JavaMailSenderImpl javaMailSender = new JavaMailSenderImpl();

        javaMailSender.setHost(host); // setting smtp host
        javaMailSender.setUsername(username); // setting username
        javaMailSender.setPassword(password); // setting password for the email account
        javaMailSender.setPort(port);// setting port
        javaMailSender.setProtocol(protocol); // setting the protocol

        Properties properties = javaMailSender.getJavaMailProperties(); // some properties have
to be set by Properties object

        properties.setProperty("mail.smtp.auth" , auth);// this property tells that we have
configured
                                        // the authentication details so that connection can be
established

        properties.setProperty("mail.smtp.ssl.enable" , sslEnable); // telling JavaMail to use
SSL


        return javaMailSender;

    }


}
```

PersistanceConfig:

```
package com.adtimokhin.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.TransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

import javax.persistence.EntityManagerFactory;
import javax.sql.DataSource;
```

```java
/**
 * @author adtimokhin
 * 03.08.2021
 **/

@Configuration
@PropertySource("classpath:persistence.properties")
@EnableTransactionManagement
@EnableJpaRepositories("com.adtimokhin.repository")
public class PersistenceConfig {

    /**
     * Creates a {@link DataSource} object that is used to connect to a database.
     *
     * @param user     {@link String} username
     * @param password {@link String} password
     * @param url      {@link String} database URL
     * @param driver   {@link String} driver class name for a given database.
     * @return a {@link DataSource}
     */
    @Bean
    public DataSource getDataSource(@Value("${user}") String user,
                                    @Value("${password}") String password,
                                    @Value("${url}") String url,
                                    @Value("${driver}") String driver) {

        DriverManagerDataSource dataSource = new DriverManagerDataSource();

        dataSource.setUsername(user);
        dataSource.setPassword(password);
        dataSource.setUrl(url);
        dataSource.setDriverClassName(driver);

        return dataSource;
    }

    /**
     * Creates an object that is responsible for transactions in the database.
     *
     * @param entityManagerFactory object that creates entities for ORM interaction
     * @return {@link TransactionManager}
     */
    @Bean
    public JpaTransactionManager transactionManager(EntityManagerFactory entityManagerFactory) {
        JpaTransactionManager transactionManager = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(entityManagerFactory);
        return transactionManager;
    }

    /**
     * Creates an object that allows to use JPA with Hibernate's ORM.
     *
     * @return {@link HibernateJpaVendorAdapter}.
     */
    @Bean
    public JpaVendorAdapter getJpaVendorAdapter() {
        return new HibernateJpaVendorAdapter();
    }

    @Bean
    public EntityManagerFactory entityManagerFactory(DataSource dataSource, JpaVendorAdapter
jpaVendorAdapter) {

        LocalContainerEntityManagerFactoryBean emf = new
LocalContainerEntityManagerFactoryBean();
        emf.setDataSource(dataSource); // emf needs datasource to create Connection with the
database
        emf.setPackagesToScan("com.adtimokhin.model"); // tells Spring where to find our entities
        emf.setJpaVendorAdapter(jpaVendorAdapter); // emf needs jpaVendorAdapter to work with JPA
        emf.afterPropertiesSet();

        return emf.getObject();
    }

    @Bean
    public PersistenceExceptionTranslationPostProcessor persistenceExceptionTranslator() {
        return new PersistenceExceptionTranslationPostProcessor();
    }
```

```
}
```

SecurityConfig:

```
package com.adtimokhin.config;

import com.adtimokhin.model.User;
import com.adtimokhin.security.AuthProvider;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuild
er;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @author adtimokhin
 * 10.08.2021
 **/

@Configuration
@EnableWebSecurity
@ComponentScan("com.adtimokhin.security")
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private AuthProvider authProvider;

    /**
     * Method that configures security chain for our HTTP requests.
     *
     * @param http {@link HttpSecurity} is responsible for the security filtering chain.
     * @throws Exception if configuration is set incorrectly on the {@link HttpSecurity} object.
     */
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests() // Objective 10.3
                .antMatchers("/booking/**").authenticated() // Objective 10.1.1
                .antMatchers("/account/**").authenticated() // Objective 10.1.2
                .antMatchers("/complaint/**").authenticated() // Objective 10.1.3
                .antMatchers("/login").anonymous() // Objective 10.2.1
                .antMatchers("/sign_up").anonymous() // Objective 10.2.2
                .antMatchers("/*").permitAll() // Objective 10.1.4 and 10.2.3
                .and()
                .formLogin()
                .loginPage("/login")
                .loginProcessingUrl("/login/process")
                .usernameParameter("email")
                .passwordParameter("password")
                .failureUrl("/login?error=true") // Objective 3.4
                .defaultSuccessUrl("/booking/actions") // Objective 3.4
                .and()
                .logout()
                .logoutUrl("/logout") // Objective 9.2
                .logoutSuccessUrl("/") // Objective 9.3
                .and()
                .exceptionHandling()
                .accessDeniedPage("/booking/actions"); // Objective 10.4
    }

    /**
     * Creates a {@link Bean} of type {@link BCryptPasswordEncoder} that is used to encrypt and
decrypt {@link User#getPassword()}
     *
     * @return {@link BCryptPasswordEncoder}
     */
    @Bean
    public PasswordEncoder getPasswordEncoder() { // Objective 13.2
        return new BCryptPasswordEncoder(10);
    }
```

```
    /**
     * Sets an authentication provider that would be called to validate users that login onto the
platform. In this case - {@link AuthProvider}.
     *
     * @param auth {@link AuthenticationManagerBuilder}.
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) {
        auth.authenticationProvider(authProvider);
    }


}
```

SecurityInitializer:

```
package com.adtimokhin.config;

import org.springframework.core.annotation.Order;
import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

/**
 * @author adtimokhin
 * 10.08.2021
 **/

@Order(2)
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer {
}
```

SyncConfig:

```
package com.adtimokhin.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

/**
 * @author adtimokhin
 * 23.10.2021
 **/

@Configuration
@EnableScheduling
public class SyncConfig {
}
```

WebAppInitializer:

```
package com.adtimokhin.config;

import org.springframework.core.annotation.Order;
import org.springframework.web.context.WebApplicationContext;
import org.springframework.web.servlet.DispatcherServlet;
import org.springframework.web.servlet.FrameworkServlet;
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

/**
 * @author adtimokhin
 * 31.07.2021
 **/
@Order(1)
public class WebAppInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {
    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[0];
    }

    /**
     * Sets what configuration classes must be used when the program starts.
```

```
     *
     * @return {@link Class<?>[]} of configuration classes.
     */
    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class[]{WebConfig.class, PersistenceConfig.class,
                SecurityConfig.class, SyncConfig.class, AspectConfig.class, EmailConfig.class};
    }

    /**
     * Sets mappings to which dispatcher servlet must respond.
     *
     * @return {@link String[]} of URLs.
     */
    @Override
    protected String[] getServletMappings() {
        return new String[]{"/"};
    }

    /**
     * Generates {@link DispatcherServlet} class with custom parameters. The only custom
parameter is that the dispatcher must not handle 404 error by itself.
     *
     * @param servletAppContext {@link WebApplicationContext}
     * @return {@link FrameworkServlet}
     */
    @Override
    protected FrameworkServlet createDispatcherServlet(WebApplicationContext servletAppContext) {
        DispatcherServlet dispatcherServlet = (DispatcherServlet)
super.createDispatcherServlet(servletAppContext);
        dispatcherServlet.setThrowExceptionIfNoHandlerFound(true);
        return dispatcherServlet;
    }
}
```

AccountController:

```
package com.adtimokhin.controller;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Types;
import com.adtimokhin.model.User;
import com.adtimokhin.security.ContextProvider;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.util.Sorter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

/**
 * @author adtimokhin
 * 23.10.2021
 **/

/**
 * A class that is responsible for handling requests related to users' account information.
 **/
@Controller
@RequestMapping("account")
public class AccountController {

    @Autowired
    private BookingService bookingService;

    @Autowired
    private ContextProvider contextProvider;

    private final Sorter sorter = new Sorter();

    /**
     * A method that returns a page contain data about user's currently active bookings. On this
page user can as well cancel their bookings.
```

```
     *
     * @param msg   {@link String} message that is used to tell a user whether their booking
could have been canceled after the attempt to cancel the booking.
     * @param model {@link Model}.
     * @return a link to account.ftl page.
     */
    @GetMapping("/bookings")
    public String viewBookingInfo(
            @RequestParam(name = "msg", required = false) String msg,
            Model model) {
        // Objective 6
        User user = contextProvider.getUser();
        List<Booking> washingMachineBookingList = bookingService.findAllByUser(user,
Types.WASHING_MACHINE);
        List<Booking> dryingMachineBookingList = bookingService.findAllByUser(user,
Types.DRYING_MACHINE);

        model.addAttribute("name", user.getName());
        if (washingMachineBookingList != null) {
            model.addAttribute("washings",
sorter.sortBookings(sorter.getRelevantBookings(washingMachineBookingList)));
        }
        if (dryingMachineBookingList != null) {
            model.addAttribute("dryings",
sorter.sortBookings(sorter.getRelevantBookings(dryingMachineBookingList)));
        }

        if (msg != null) {
            model.addAttribute("msg", msg);
        }

        return "account/account"; // Objective 6.1
    }

    /**
     * A method used to return a user a page containing information about their account.
     *
     * @param model {@link Model} used to display relevant information about user.
     * @return info.ftl page.
     */
    @GetMapping("/")
    public String viewAccountInfo(Model model) {
        User user = contextProvider.getUser();
        model.addAttribute("name", user.getName());
        return "account/info";
    }

}
```

AuthController:

```
package com.adtimokhin.controller;

import com.adtimokhin.model.User;
import com.adtimokhin.security.AuthProvider;
import com.adtimokhin.service.UserService;
import com.adtimokhin.util.email.SimpleEmailSender;
import com.adtimokhin.util.token.SimpleEmailTokenGenerator;
import com.adtimokhin.validation.UserValidator;
import com.adtimokhin.validation.errors.UserError;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.LockedException;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.WebAttributes;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import javax.servlet.http.HttpServletRequest;
import java.util.List;

/**
 * @author adtimokhin
 * 16.10.2021
```

```
 **/


/**
 * A class that is responsible for handling URLs related to authentication matters.
 */
@Controller
public class AuthController {

    @Autowired
    private UserValidator userValidator;

    @Autowired
    private UserService userService;

    @Autowired
    private SimpleEmailSender emailSender;

    @Autowired
    private SimpleEmailTokenGenerator simpleEmailTokenGenerator;

    /**
     * Method that returns to user a login page.
     *
     * @param error   {@link Boolean} that indicates whether there was an error in validation of
the input. Extracted from the URL.
     * @param model   {@link Model} that is used to set variable content on the login page.
     * @param request {@link HttpServletRequest} that is used to analyse what validation error
had occured in {@link AuthProvider}
     * @return a view. In this case - a page with a login form.
     */
    @GetMapping("/login")
    public String loginGet(@RequestParam(value = "error", required = false) Boolean error,
                           Model model, HttpServletRequest request) {
        if (Boolean.TRUE.equals(error)) { // as error might be null we have to put this statement
in this order
            // Objective 3.4
            AuthenticationException authenticationException =
                    (AuthenticationException)
request.getSession().getAttribute(WebAttributes.AUTHENTICATION_EXCEPTION);
            if (authenticationException.getClass().equals(LockedException.class)) {
                model.addAttribute("error", "You have to verify your " +
                        "email before logging onto the page");
            } else {
                model.addAttribute("error", "No user with such credentials was found");
            }
        }
        return "login";
    }

    /**
     * Method that returns a user a sign up page
     *
     * @return a link to a sign_up.ftl page
     */
    @GetMapping("/sign_up")
    public String signUpGet() {
        return "signUp"; // Objective 1.1.
    }

    /**
     * Method that is used to validate input for creating a new user. Validations occur in {@link
UserValidator} class.
     *
     * @param email       {@link String} email
     * @param name        {@link String} name
     * @param password    {@link String} password
     * @param passwordTwo {@link String} confirm password
     * @param model       {@link Model} that is used to update content on the sing up page (to
pass any error messages that have occurred.)
     * @return link to sign_up.ftl if the sign_up was unsuccessful, or a link to a page that
confirms that a confirmation letter has been sent.
     */
    @PostMapping("/sign_up")
    public String signUpPost(@RequestParam(name = "email") String email,
                             @RequestParam(name = "name") String name,
                             @RequestParam(name = "password") String password,
                             @RequestParam(name = "passwordTwo") String passwordTwo,
                             Model model) {
```

```
        User user = new User(name, email, password);

        List<UserError> errorList = userValidator.validate(user, passwordTwo); // making our
validations

        // We decide whether there are any errors and then either return fail or success
validation message. Objective 1.4
        if (!errorList.isEmpty()) {
            model.addAttribute("errors", errorList);
            return "signUp";
        } else {
            user.setEmailToken(simpleEmailTokenGenerator.generate());// Objectives 1.6 and 1.6.1
            userService.save(user); // Objective 1.5
            emailSender.sendEmailVerificationEmail(user); // Objective 1.7
            return "email/verificationEmailSentConfirmation";
        }

    }


    /**
     * @return a link to index.ftl (home page)
     */
    @GetMapping("/")
    public String index() {
        return "index";
    }


    /**
     * Method that extracts from URL an email verification token created in {@link
SimpleEmailTokenGenerator} and verifies user's email if it is valid.
     *
     * @param token {@link String} email verification token.
     * @return a page that tells a user whether verification was successful or not.
     */
    @GetMapping("/verify/{token}")
    public String verifyEmailPost(@PathVariable(name = "token") String token) { // Objective 2.1
        User user = userService.findByEmailToken(token);
        if (user != null) { // if email was verified
            userService.removeEmailToken(user); // Objective 2.2
            return "email/successfulVerification";
        }
        return "email/unsuccessfulVerification";
    }

}
```

BookingController:

```
package com.adtimokhin.controller;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Types;
import com.adtimokhin.model.User;
import com.adtimokhin.security.ContextProvider;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.service.MachineService;
import com.adtimokhin.util.time.DateFormatResolver;
import com.adtimokhin.util.time.Timetable;
import com.adtimokhin.util.time.TimetableContainer;
import com.adtimokhin.validation.BookingValidator;
import com.adtimokhin.validation.errors.BookingError;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

import static com.adtimokhin.util.time.DateFormatResolver.DATE_FORMAT;
import static com.adtimokhin.util.time.DateFormatResolver.MINUTES_BEFORE_CANCEL;


/**
```

```
 * @author adtimokhin
 * 16.10.2021
 **/

/**
 * Class that handles all URL and requests regarding bookings.
 **/
@Controller
@RequestMapping("booking")
public class BookingController {

    @Autowired
    private BookingService bookingService;

    @Autowired
    private MachineService machineService;

    @Autowired
    private BookingValidator bookingValidator;

    @Autowired
    private TimetableContainer timeTableContainer;

    @Autowired
    private ContextProvider contextProvider;


    private final DateFormatResolver dateFormatResolver = new DateFormatResolver();

    private final static Logger logger = LoggerFactory.getLogger(BookingController.class);


    /**
     * @return a link to a page that displays with all possible actions on the website for
authenticated users.
     */
    @GetMapping("/actions")
    public String selectAction() {
        return "booking/actions";
    }


    /**
     * A method that returns a page with information about all machines of a certain type for a
given date. On this page user can also make a booking.
     *
     * @param machineType {@link String} that identifies a type of machinery. It should equal to
either {@link Types#WASHING_MACHINE} or {@link Types#DRYING_MACHINE}
     * @param date        {@link String} that represents date for which user wishes view booking
data.
     * @param error       {@link String} error message that might be present if user fails to
pass all validations conducted in {@link BookingValidator}.
     * @param msg         {@link String} message that appears if user passes validations
sucessfully.
     * @param model       {@link Model} that is used to pass information to preview.ftl. That
information includes: error, msg, data about bookings for a particular date, date.
     * @return preview.ftl page.
     */
    @GetMapping("/view/{type}")
    public String getAllWashingMachineData(@PathVariable(value = "type", required = false) String
machineType,
                                           @RequestParam(name = "date", required = false) String
date,
                                           @RequestParam(name = "error", required = false) String
error,
                                           @RequestParam(name = "msg", required = false) String
msg,
                                           Model model) {
        // getting information about the time when certain machines are available.
        String stringDate = timeTableContainer.getDay();
        String dateRepresent = stringDate.substring(0, "2021 18 18".length()).replace(" ", "-");


        if (date != null) { // choosing bookings for which date to display to user.
            if (dateFormatResolver.appropriateFormatWithDash(date)) { // Validating date set by
user. Objective 4.3
                String temp = date;
                date = date.replace("-", " ") + " 00:00";
                if (!dateFormatResolver.onTheSameDay(date, stringDate)) {
                    timeTableContainer.changeDay(date);
```

```
                    stringDate = date;
                    dateRepresent = temp;
                }
            }
        }

        model.addAttribute("date", stringDate);
        model.addAttribute("representativeDate", dateRepresent);

        List<Timetable> timeTables;
        if (Types.WASHING_MACHINE.toString().equals(machineType)) { // setting timetables of
bookings. Objective 4.1
            timeTables = timeTableContainer.getWashingMachineTimeTables();
        } else if (Types.DRYING_MACHINE.toString().equals(machineType)) {
            timeTables = timeTableContainer.getDryingMachineTimeTables();
        } else {
            model.addAttribute("error", "Please, select type of machinery to use!");
            return "booking/preview";
        }

        if (error != null) { // adding error and other messages.
            model.addAttribute("error", error);
        } else if (msg != null) {
            model.addAttribute("msg", msg);
        }
        model.addAttribute("machineType", machineType);
        model.addAttribute("timeTables", timeTables);

        return "booking/preview";
    }

    /**
     * A method that returns a page that displays a form to fill in to select a date for which
user wishes to view booking data.
     *
     * @param machineType {@link String} correctly inputted machine type.
     * @param date        {@link String} date that user has been using before to view the booking
data.
     * @param model       {@link Model}.
     * @return error page if there are some errors with the input data. Else - a page for
selecting a date.
     */
    @GetMapping("/choose/date/{type}")
    public String viewDates(@PathVariable(value = "type", required = false) String machineType,
                            @RequestParam(name = "date") String date,
                            Model model) { // Objective 4.2

        if (!(Types.DRYING_MACHINE.toString().equals(machineType) ||
Types.WASHING_MACHINE.toString().equals(machineType))) {
            return "redirect:/booking/view/error";
        }


        if (!dateFormatResolver.appropriateFormat(date)) {
            return "redirect:/booking/view/error";
        }

        String formattedDate = date.substring(0, "2021 18 18".length()).replace(" ", "-");

        int year = Integer.parseInt(formattedDate.substring(0, 4));

        String minDate = year + "-01-01";
        String maxDate = (year + 1) + "-01-01";

        model.addAttribute("currentDate", formattedDate);
        model.addAttribute("minDate", minDate);
        model.addAttribute("maxDate", maxDate);
        model.addAttribute("machineType", machineType);

        return "booking/dateSelect";
    }

    /**
     * A method that is used to cancel user's booking.
     *
     * @param machineType {@link String} that identifies a type of machinery. It should equal to
either {@link Types#WASHING_MACHINE} or {@link Types#DRYING_MACHINE}
     * @param id          {@link Integer} that is used to identify a booking's id in the system.
     * @param model       {@link Model}.
     * @return bookings.ftl
```

```
     */
    @PostMapping("/cancel/{type}")
    public String cancelBooking(@PathVariable(value = "type", required = false) String
machineType,
                                @RequestParam(name = "id") int id, Model model) {

        if (machineType == null) {
            model.addAttribute("msg", "You have to specify machine type!");
            logger.trace("Failed to cancel a booking: machineType specified is null."); //
Objective 11.2
            return "redirect:/account/bookings";
        } else if (Types.DRYING_MACHINE.toString().equals(machineType)) {
            Booking booking = bookingService.findById(id);
            if (booking != null) { // Objective 7.2.1
                if (booking.getUser().getId() == contextProvider.getUser().getId()) { //
Objective 7.2.2
                    if (dateFormatResolver.areFarEnoughInTime(booking.getStartDate(),
dateFormatResolver.today(), MINUTES_BEFORE_CANCEL)) { // Objective 7.2.3
                        logger.trace("About to delete booking for machine id {}, for time {} - {}
and by user {} ", booking.getMachine().getId(), booking.getStartDate(), booking.getEndDate(),
booking.getUser().getId()); // Objective 11.2
                        bookingService.delete(booking); // Objective 7.4
                        logger.trace("Cancellation was successful!"); // Objective 11.2
                        model.addAttribute("msg", "Cancellation was successful!"); // Objective
7.3
                        return "redirect:/account/bookings";
                    } else {
                        model.addAttribute("msg", "Could not cancel. You have to cancel " +
MINUTES_BEFORE_CANCEL + " minutes before your booking time"); // Objective 7.3
                        return "redirect:/account/bookings";
                    }
                }
                model.addAttribute("msg", "You do not own that booking");// Objective 7.3
                return "redirect:/account/bookings";
            }
        } else if (Types.WASHING_MACHINE.toString().equals(machineType)) {
            Booking booking = bookingService.findById(id);
            if (booking != null) { // Objective 7.2.1
                if (booking.getUser().getId() == contextProvider.getUser().getId()) { //
Objective 7.2.2
                    if (dateFormatResolver.areFarEnoughInTime(booking.getStartDate(),
dateFormatResolver.today(), MINUTES_BEFORE_CANCEL)) { // Objective 7.2.3
                        logger.trace("About to delete booking for machine id {}, for time {} - {}
and by user {} ", booking.getMachine().getId(), booking.getStartDate(), booking.getEndDate(),
booking.getUser().getId()); // Objective 11.2
                        bookingService.delete(booking); // Objective 7.4
                        logger.trace("Cancellation was successful!");// Objective 11.2
                        model.addAttribute("msg", "Cancellation was successful!");
                        return "redirect:/account/bookings";
                    } else {
                        model.addAttribute("msg", "Could not cancel. You have to cancel " +
MINUTES_BEFORE_CANCEL + " minutes before your booking time"); // Objective 7.3
                        return "redirect:/account/bookings";
                    }
                }
                model.addAttribute("msg", "You do not own that booking");// Objective 7.3
                return "redirect:/account/bookings";
            }
        }
        model.addAttribute("msg", "Could not identify machine type…");// Objective 7.3
        return "redirect:/account/bookings";
    }

    /**
     * A method that is used to add a booking after validating the inputted data.
     *
     * @param machineType {@link String} that identifies a type of machinery. It should equal to
either {@link Types#WASHING_MACHINE} or {@link Types#DRYING_MACHINE}
     * @param machineId   {@link Integer} that is used to identify a machine's id in the system.
     * @param date        {@link String} date that user has been using to view the booking data.
     * @param periods     {@link List<String>} that stores data about ids of periods that user
has selected to book.
     * @param model       {@link Model}.
     * @return error page if validation failed, or otherwise the preview.ftl page with the
succession message.
     */
    @PostMapping("/add/{type}")
    public String addBooking(@PathVariable(value = "type", required = false) String machineType,
                             @RequestParam(name = "machineId") String machineId,
                             @RequestParam(name = "date") String date,
```

```
                                @RequestParam(name = "periods") List<String> periods,
                                Model model) {

        if (date == null) {
            model.addAttribute("error", "Enter date in the first box. Enter it in format: " +
DATE_FORMAT);
            model.addAttribute("date", dateFormatResolver.today());

            if (!(Types.DRYING_MACHINE.toString().equals(machineType) ||
Types.WASHING_MACHINE.toString().equals(machineType))) {
                return "redirect:/booking/view/error";
            }
            return "redirect:/booking/view/" + machineType;
        }
        if (date.length() == "2021 11 06".length()) {
            date += " 00:00";
        }

        User user = contextProvider.getUser();

        logger.trace("User {} (name: {}) is about to attempt adding a booking", user.getId(),
user.getName());

        // Objective 5.3.1
        boolean isWashingMachine = true;
        if (machineType == null) {
            model.addAttribute("error", "You have to specify machine type!");
            logger.trace("Failed to add a new booking: machineType specified is null.");
            return "redirect:/booking/view/error";
        } else if (Types.DRYING_MACHINE.toString().equals(machineType)) {
            isWashingMachine = false;
        } else if (!Types.WASHING_MACHINE.toString().equals(machineType)) {
            model.addAttribute("error", "You have to specify machine type!");
            logger.trace("Failed to add a new booking: machineType specified is null.");
            return "redirect:/booking/view/error";
        }


        // validations
        List<BookingError> errors = bookingValidator.validate(machineType, machineId, date,
periods, user);
        if (errors == null) {
            // Objective 5.5
            int minId = Integer.parseInt(periods.get(0).substring(3));
            int maxId = Integer.parseInt(periods.get(periods.size() - 1).substring(3));

            // all time slots are 30 minutes in length. Bookings start at 6.00 am.
            String startHour = String.valueOf(6 + ((minId) / 2));
            String startMinute = String.valueOf((minId % 2) * 30);
            String endHour = String.valueOf(6 + ((maxId + 1) / 2));
            String endMinute = String.valueOf(((maxId + 1) % 2) * 30);

            Booking booking = new Booking(user,
                    machineService.findById(Integer.parseInt(machineId)),
                    dateFormatResolver.resolveTimeForDate(startHour, startMinute, date),
                    dateFormatResolver.resolveTimeForDate(endHour, endMinute, date));

            bookingService.save(booking);
            timeTableContainer.updateTimeTable(booking.getMachine().getId(), isWashingMachine);
            logger.trace("New booking was made.");

            model.addAttribute("msg", "Your booking is all set!");

        } else {
            // Objective 5.4
            logger.trace("An error had occurred while user {} tried to add a new booking. Error:
{}", user.getId(), errors.get(0).getMessage());
            model.addAttribute("error", errors.get(0).getMessage());
            model.addAttribute("date", date);
        }
        return "redirect:/booking/view/" + machineType;
    }

}
```

ComplaintController:

```
package com.adtimokhin.controller;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Complaint;
import com.adtimokhin.model.Types;
import com.adtimokhin.model.User;
import com.adtimokhin.security.ContextProvider;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.service.ComplaintService;
import com.adtimokhin.util.time.DateFormatResolver;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import java.util.List;

/**
 * @author adtimokhin
 * 23.10.2021
 **/

/**
 * A class that responds to all URLs and requests related to complaints.
 **/
@Controller
@RequestMapping("complaint")
public class ComplaintController {

    @Autowired
    private ComplaintService complaintService;

    @Autowired
    private BookingService bookingService;

    @Autowired
    private ContextProvider contextProvider;

    private final DateFormatResolver dateFormatResolver = new DateFormatResolver();

    /**
     * A method that is used to add a complaint into the system. The input is also validated.
     *
     * @param type     {@link String} that identifies a type of machinery. It should equal to
either {@link Types#WASHING_MACHINE} or {@link Types#DRYING_MACHINE}
     * @param idString {@link String} id of a booking.
     * @param model    {@link Model}
     * @return an account.ftl page that tells whether a complaint was added or not.
     */
    @PostMapping("/add")
    public String addComplaintProcess(
                                      @RequestParam(name = "id") String idString, Model model) {

        int id;
        try {
            id = Integer.parseInt(idString);
        } catch (Exception e) {
            model.addAttribute("msg", "id should be entered as an number!"); // Objective 8.4
            return "redirect:complaint/";
        }

        Booking booking = bookingService.getBookingForMachineWithId(id,
dateFormatResolver.today());
       if (booking == null) { // Objective 8.3.1
            model.addAttribute("msg", "Right now this machine is not booked or this is the first
booking on a day. This system is currently in no power to deal with this. Please, call someone to
help."); // Objective 8.4
            return "redirect:complaint/";
        }

        User user = contextProvider.getUser();
       String today = dateFormatResolver.today();
       boolean canMakeComplaint = false;
       for (Booking booking :
```

```
                bookings) {
            if (dateFormatResolver.datesMatch(today, booking.getStartDate())) {
                canMakeComplaint = true;
                break;
            }
        }

        if(!canMakeComplaint){
            model.addAttribute("msg", "You have no bookings for today. Hence, you cannot
        complain");
            return "redirect:complaint/";
        }

        Complaint complaint = new Complaint(user, booking);


        complaintService.save(complaint); // Objective 8.5
        model.addAttribute("msg", "Thank you for telling us about the problem you have! Staff is
on their way."); // Objective 8.4


        return "redirect:complaint/";
    }


    @GetMapping("/")
    public String viewAllComplaints(Model model, @RequestParam(name = "msg", required = false)
String msg) {
        List<Complaint> washingMachineComplaints =
complaintService.findAll(Types.WASHING_MACHINE);
        List<Complaint> dryingMachineComplaints = complaintService.findAll(Types.DRYING_MACHINE);

        if (washingMachineComplaints != null) {
            model.addAttribute("washingMachineComplaints", washingMachineComplaints);
        }
        if (dryingMachineComplaints != null) {
            model.addAttribute("dryingMachineComplaints", dryingMachineComplaints);
        }

        if (msg != null) {
            model.addAttribute("msg", msg);
        }

        return "complaints/complaint";
    }
}
```

DefaultExcpetionHandlerControlAdvice:

```
package com.adtimokhin.controller;

import com.adtimokhin.exceptions.NoSuchUserFoundException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.servlet.NoHandlerFoundException;

import java.util.Arrays;

/**
 * @author adtimokhin
 * 13.08.2021
 **/

/**
 * A class that handles all exceptions that threaten the execution of the program
 **/
@ControllerAdvice
public class DefaultExceptionHandlerControlAdvice { // Objective 12

    private final static Logger logger =
LoggerFactory.getLogger(DefaultExceptionHandlerControlAdvice.class);

    /**
     * Handles error 404
```

```
     * @return a 404.ftl page
     */
    @ResponseStatus(HttpStatus.NOT_FOUND) // 404
    @ExceptionHandler(value = {NoHandlerFoundException.class})
    public String noHandlerFoundExceptionHandler() { // Objective 12.2
        return "errors/404";
    }


    /**
     * A method that handles {@link NoSuchUserFoundException}.
     *
     * @param e {@link Exception} that was thrown that must be analysed and logged.
     * @return a 500.ftl page
     */
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR) // 500
    @ExceptionHandler(value = NoSuchUserFoundException.class)
    public String noSuchUserFoundExceptionHandler(Exception e) { // Objective 12.1
        logger.error("Internal server error it was caused by " + e.getMessage() + ".\nFull stack
trace:" + Arrays.toString(e.getStackTrace())); // Objective 11.1
        return "errors/500";
    }

    /**
     * A method that handles all other exceptions.
     *
     * @param e {@link Exception} that was thrown that must be analysed and logged.
     * @return a 500.ftl page
     */
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR) // 500
    @ExceptionHandler(value = Exception.class)
    public String sampleExceptionHandler(Exception e) { // Objective 12.1
        logger.error("Internal server error it was caused by " + e.getMessage() + ".\nFull stack
trace:" + Arrays.toString(e.getStackTrace())); // Objective 11.1
        return "errors/500";
    }


}
```

EmptyFiledException:

```
package com.adtimokhin.exceptions;

/**
 * @author adtimokhin
 * 08.10.2021
 **/

/**
 * A class that represents an error that a field is empty.
 */
public class EmptyFieldException extends Exception {

    public EmptyFieldException(String msg){
        super(msg);
    }
}
```

NoSuchUserFoundException:

```
package com.adtimokhin.exceptions;

/**
 * @author adtimokhin
 * 13.08.2021
 **/

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

/**
 * A class that tells that no user is associated with a given {@link
UsernamePasswordAuthenticationToken}
 */
```

```
public class NoSuchUserFoundException extends RuntimeException {
}
```

TokenOverflowException:

```
package com.adtimokhin.exceptions;

/**
 * @author adtimokhin
 * 11.11.2021
 **/

/**
 * A class that represents that there are too many tokens stored in the system.
 */
public class TokenOverflowException extends RuntimeException {

    public TokenOverflowException(String message) {
        super(message);
    }
}
```

Booking:

```
package com.adtimokhin.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.util.List;

import static javax.persistence.GenerationType.IDENTITY;

/**
 * @author adtimokhin
 * 01.01.2022
 **/


@Entity
@Table(name = "booking_table", schema = "main_schema")
@AllArgsConstructor
@NoArgsConstructor
@Getter
@Setter
public class Booking {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Integer id;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "machine_id")
    private Machine machine;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id")
    private User user;

    private String startDate;

    private String endDate;

    @OneToMany(mappedBy = "booking")
    private List<Complaint> complaints;


    public Booking(User user, Machine machine, String startDate, String endDate) {
        this.user = user;
        this.machine = machine;
        this.startDate = startDate;
```

```
        this.endDate = endDate;
    }

    @Override
    public String toString() {
        return "Booking{" +
                " userId=" + user.getId() +
                ", dryingMachineId=" + machine.getId() +
                ", startDate='" + startDate + '\'' +
                ", endDate='" + endDate + '\'' +
                '}';
    }

    public String presentNicely() {
        return "Booking for a drying machine with id " + machine.getId() + " made to start from "
+ startDate + " and finish at " + endDate;
    }

    public String getEndDate() {
        return this.endDate;
    }
}
```

Complaint:

```
package com.adtimokhin.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

import static javax.persistence.GenerationType.IDENTITY;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Entity
@Table(name = "complaint_table", schema = "main_schema")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Complaint {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Integer id;


    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "reporting_user_id")
    private User reportingUser;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "booking_id")
    private Booking booking;


    public Complaint(User reportingUser, Booking booking) {
        this.reportingUser = reportingUser;
        this.booking = booking;
    }
}
```

Machine:

```
package com.adtimokhin.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```java
import javax.persistence.*;
import java.util.List;

import static javax.persistence.GenerationType.IDENTITY;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Entity
@Table(name = "machine_table", schema = "main_schema")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Machine {

    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Integer id;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "type")
    private MachineType type;

    @OneToMany(mappedBy = "machine")
    private List<Booking> bookings;

}
```

MachineType:

```java
package com.adtimokhin.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.util.List;

import static javax.persistence.GenerationType.IDENTITY;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Entity
@Table(name = "machine_types_table", schema = "main_schema")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class MachineType {
    @Id
    @GeneratedValue(strategy = IDENTITY)
    private Integer id;

    @Column(name = "description", unique = true, nullable = false)
    private String typeDescription;

    @OneToMany(mappedBy = "type")
    private List<Machine> machineList;
}
```

Types:

```java
package com.adtimokhin.model;

/**
 * @author adtimokhin
 * 01.01.2022
 **/
```

```
public enum Types {
    WASHING_MACHINE, DRYING_MACHINE
}
```

User:

```
package com.adtimokhin.model;

import com.adtimokhin.model.bookings.DryingMachineBooking;
import com.adtimokhin.model.bookings.WashingMachineBooking;
import com.adtimokhin.model.complaints.DryingMachineComplaint;
import com.adtimokhin.model.complaints.WashingMachineComplaint;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

import javax.persistence.*;
import java.util.List;

/**
 * @author adtimokhin
 * 16.10.2021
 **/

@Entity
@Table(name = "user_table", schema = "main_schema")
@Getter
@Setter
@NoArgsConstructor
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String name;

    private String email;

    private String password;

    @OneToMany(mappedBy = "user")
    private List<DryingMachineBooking> dryingMachineBookingList;

    @OneToMany(mappedBy = "user")
    private List<WashingMachineBooking> washingMachineBookingsList;

    @OneToMany(mappedBy = "reportingUser", targetEntity = WashingMachineComplaint.class)
    private List<WashingMachineComplaint> reportingWashingMachineComplaints;

    @OneToMany(mappedBy = "reportingUser", targetEntity = DryingMachineComplaint.class)
    private List<DryingMachineComplaint> reportingDryingMachineComplaints;

    @Column(name = "email_token")
    private String emailToken;

    @OneToMany(mappedBy = "user")
    //@LazyCollection(LazyCollectionOption.FALSE)
    private List<Booking> bookings;

    @OneToMany(mappedBy = "reportingUser")
    private List<Complaint> complaints;


    public User(String name, String email, String password) {
        this.name = name;
        this.email = email;
        this.password = password;
    }

}
```

BookingRepository:

```java
package com.adtimokhin.repository;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.MachineType;
import com.adtimokhin.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/
@Repository
public interface BookingRepository extends JpaRepository<Booking, Integer> {


    /**
     * Method that finds {@link List<Booking>} by a {@link User#getId()} in ascending order.
     *
     * @param userId {@link User#getId()}
     * @return {@link List<Booking>}
     */
    @Query("SELECT b FROM Booking b WHERE b.user.id = :userId ORDER BY b.user.id ASC")
    List<Booking> findAllByUserIdOrderByIdAsc(@Param("userId") int userId);

    /**
     * Method that finds {@link List<Booking>} by a {@link Machine#getId()}.
     *
     * @param id {@link Machine#getId()}
     * @return {@link List<Booking>}
     */
    @Query("SELECT b FROM Booking b WHERE b.machine.id = :id")
    List<Booking> findAllByMachineId(@Param("id") int id);

    /**
     * Method that finds {@link List<Booking>} by a {@link User#getId()}.
     *
     * @param id {@link User#getId()}
     * @return {@link List<Booking>}
     */
    @Query("SELECT b FROM Booking b WHERE  b.user.id = :id")
    List<Booking> findAllByUserId(@Param("id") int id);


    /**
     * * Method that finds {@link List<Booking>} by a {@link User#getId()} and {@link
Booking#getStartDate()}
     *
     * @param userId {@link User#getId()}
     * @param date   {@link Booking#getStartDate()}
     * @return {@link List<Booking>}
     */
    @Query("SELECT b FROM Booking b WHERE  b.user.id = :userId AND b.startDate LIKE(:date)")
    List<Booking> findAllByUserIdAndStartDateStartingWith(@Param("userId") int userId,
@Param("date") String date);



    /**
     * * Method that finds {@link List<Booking>} by a {@link User#getId()} and {@link
MachineType}
     *
     * @param userId {@link User#getId()}
     * @param type   {@link MachineType}
     * @return {@link List<Booking>}
     */
    @Query("SELECT b FROM Booking b WHERE b.user.id = :userId AND b.machine.type= :type")
    List<Booking> findAllByUserIdAndMachine_Type(@Param("userId") int userId, @Param("type")
MachineType type);
}
```

ComplaintRepository:

```
package com.adtimokhin.repository;

import com.adtimokhin.model.Complaint;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.MachineType;
import com.adtimokhin.model.Types;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/
@Repository
public interface ComplaintRepository extends JpaRepository<Complaint, Integer> {

    /**
     * Finds all {@link Complaint} where description of {@link MachineType#getTypeDescription()}
of {@link MachineType} of {@link Machine} that is associated with the given {@linkplain
Complaint#getBooking()}
     *
     * @param description value from {@link Types}
     * @return {@link List<Complaint>}
     */
    @Query("SELECT c FROM Complaint c WHERE c.booking.machine.type.typeDescription =:
description")
    List<Complaint> findAllByBooking_Machine_Type_TypeDescription(@Param("description") String
description);

    /**
     * Deletes all {@link Complaint} where description of {@link MachineType} is equal to the
parameter passed.
     *
     * @param description value from {@link Types}
     */
    @Query("DELETE FROM Complaint c WHERE c.booking.machine.type.typeDescription =: description")
    void deleteAllByBooking_Machine_Type_TypeDescription(@Param("description") String
description);

}
```

MachieRepository:

```
package com.adtimokhin.repository;

import com.adtimokhin.model.Machine;
import com.adtimokhin.model.MachineType;
import com.adtimokhin.model.Types;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/
@Repository
public interface MachineRepository extends JpaRepository<Machine, Integer> {

    /**
     * Finds all {@link Machine} which have {@link MachineType#getTypeDescription()} set to the
parameter passed.
     * @param type value from {@link Types}
     * @return {@link List<Machine>}
     */
    @Query("SELECT m FROM Machine m WHERE m.type.typeDescription =: type")
```

```
    List<Machine> findAllByType_TypeDescription(@Param("type") String type);
}
```

MachineTypeRepository:

```
package com.adtimokhin.repository;

import com.adtimokhin.model.MachineType;
import com.adtimokhin.model.Types;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

public interface MachineTypeRepository extends JpaRepository<MachineType, Integer> {

    /**
     * Method finds the entity that represents a given type of the machinery
     *
     * @param description should match one of the values in the {@link Types}
     * @return {@link MachineType}
     */
    @Query("SELECT m FROM MachineType m WHERE m.typeDescription = :description")
    MachineType findByTypeDescription(@Param("description") String description);
}
```

UserRepository:

```
package com.adtimokhin.repository;

import com.adtimokhin.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

/**
 * @author adtimokhin
 * 16.10.2021
 **/
@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    /**
     * Returns a {@link User} with a given email.
     *
     * @param email {@link String} email.
     * @return {@link User}
     */
    @Query("SELECT u FROM User u where u.email = :email")
    User findByEmail(@Param("email") String email);

    /**
     * @return a {@link List<User>} where {@link User#getEmailToken()} is null.
     */
    @Query("SELECT u FROM User u where u.emailToken is not null ORDER BY u.emailToken ASC ")
    List<User> findAllByEmailTokenIsNotNullOrdOrderByEmailTokenAsc();

    /**
     * Returns a {@link User} with a given {@link User#getEmailToken()}.
     *
     * @param token {@link String} that represents {@link User#getEmailToken()}.
     * @return {@link User}
     */
    @Query("SELECT u FROM User u where u.emailToken = :token")
```

```
    User findByEmailToken(@Param("token") String token);

}

// Objective 13.3. Usage of JPA protects from first-order SQL-Injections because JPA use prepared
statements.
```

AuthProvider:

```
package com.adtimokhin.security;

import com.adtimokhin.model.User;
import com.adtimokhin.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.*;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

import java.util.ArrayList;

/**
 * @author adtimokhin
 * 10.08.2021
 **/

@Component
public class AuthProvider implements AuthenticationProvider {

    @Autowired
    private UserService userService;

    @Autowired
    private PasswordEncoder passwordEncoder;


    /**
     * Checks if the data provided by a user matches any of activated accounts. If so, an
authentication token would be returned to user.
     *
     * @param authentication {@link Authentication} object that provides all data that contains
authentication details like name and credentials.
     * @return {@link UsernamePasswordAuthenticationToken} that allows user to performed actions
available for authenticated users.
     * @throws AuthenticationException when an {@link UsernamePasswordAuthenticationToken} cannot
be created.
     */
    @Override
    public Authentication authenticate(Authentication authentication) throws
AuthenticationException { // Objective 3.3

        String email = authentication.getName(); // we get the username parameter
        User user = userService.findByEmail(email);

        if (user == null) { // Objective 3.3.1
            throw new AuthenticationCredentialsNotFoundException("User with such email was not
detected");
        }

        String password = authentication.getCredentials().toString(); // we get the credentials
that we will turn into string

        // Objective 13.2
        if (!passwordEncoder.matches(password, user.getPassword())) { // Objective 3.3.3
            throw new BadCredentialsException("Bad credentials");
        }

        if (user.getEmailToken() != null) { // Objective 3.3.2
            throw new LockedException("Email is not verified");
        }

        ArrayList<GrantedAuthority> authorities = new ArrayList<>();
        return new UsernamePasswordAuthenticationToken(user.getEmail(), user.getPassword(),
authorities); // Objective 3.5
    }
```

```
    /**
     * Internal Spring method used to check that the returned value from {@link
AuthProvider#authenticate(Authentication)} is of type {@link
UsernamePasswordAuthenticationToken}.
     *
     * @param aClass a class that must be checked if it is of type {@link
UsernamePasswordAuthenticationToken}.
     * @return true if it is of {@link UsernamePasswordAuthenticationToken} type.
     */
    @Override
    public boolean supports(Class<?> aClass) {
        return UsernamePasswordAuthenticationToken.class.equals(aClass);
    }
}
```

ContextProvider:

```
package com.adtimokhin.security;

import com.adtimokhin.exceptions.NoSuchUserFoundException;
import com.adtimokhin.model.User;
import com.adtimokhin.service.UserService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContext;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

/**
 * @author adtimokhin
 * 11.08.2021
 **/

@Component
public class ContextProvider {

    @Autowired
    private UserService userService;

    private final static Logger logger = LoggerFactory.getLogger(ContextProvider.class);

    /**
     * Method that extracts {@link User} from a {@link SecurityContext} that stores {@link
UsernamePasswordAuthenticationToken}
     * @return
     */
    public User getUser() {
                SecurityContext context = SecurityContextHolder.getContext(); // securityContext
contains security
        // information about a thread that a current User uses
        Authentication authentication = context.getAuthentication(); // getting authentication
token from the securityContext
        String principle = authentication.getPrincipal().toString(); // getting email from the
authentication token

        User user = userService.findByEmail(principle);

        if (user == null){ // checking if user is equal to null
            logger.error("No user is associated with currently running thread.");
            throw new NoSuchUserFoundException();
        }
        return user;
    }

}
```

BookingService:

```
package com.adtimokhin.service;
```

```java
import com.adtimokhin.aspect.DebugMethod;
import com.adtimokhin.aspect.NotEmptyArguments;
import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.Types;
import com.adtimokhin.model.User;
import com.adtimokhin.model.bookings.DryingMachineBooking;
import com.adtimokhin.model.machine.DryingMachine;
import jakarta.validation.constraints.NotNull;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Service
@Transactional
public interface BookingService {

    /**
     * @param machine {@link Machine}
     * @return a {@link List<Booking>} that have that {@link Machine} set.
     */
    @Transactional(readOnly = true)
    @NotEmptyArguments
    List<Booking> getBookingsForMachine(Machine machine);

    /**
     * Saves {@link Booking} in the database
     *
     * @param booking {@link Booking} to save in the database
     */

    @NotEmptyArguments
    void save(Booking booking);

    /**
     * Deletes {@link Booking} from the database
     *
     * @param booking {@link Booking} to delete.
     */
    @NotEmptyArguments
    void delete(Booking booking);

    /**
     * Finds all {@link Booking} with for a given {@link User}
     *
     * @param user {@link User}
     * @param type {@link Types} type of {@link Machine}
     * @return {@link List<Booking>}
     */
    @Transactional(readOnly = true)
    @NotEmptyArguments
    List<Booking> findAllByUser(User user, Types type);

    /**
     * Finds all {@link Booking} with for a given {@link Machine#getId()}
     *
     * @param id {@link Machine#getId()}
     * @return {@link List<Booking>}
     */
    @Transactional(readOnly = true)
    @DebugMethod
    List<Booking> findAllByMachineId(int id);

    /**
     * Finds a {@link Booking} by id.
     *
     * @param id {@link Booking#getId()}
     * @return {@link Booking}
     */
    @Transactional(readOnly = true)
    @DebugMethod
    Booking findById(int id);

    /**
```

```
     * Finds a {@link Booking} for a given {@link Machine} and date.
     *
     * @param id   {@link DryingMachine#getId()}
     * @param date {@link String} date.
     * @return {@link DryingMachineBooking}
     */
    @Transactional(readOnly = true)
    @DebugMethod
    Booking getBookingForMachineWithId(int id, String date);

    /**
     * Finds a {@link Booking} for a given {@link User} and date.
     *
     * @param userId {@link User#getId()}
     * @param date   {@link String} date.
     * @return {@link List<Booking>}
     */
    @Transactional(readOnly = true)
    @DebugMethod
    List<Booking> findAllMachineBookings(int userId,String date);


    /**
     * Finds all {@link Booking} for a week that contains the passed date, and a given {@link
User}.
     *
     * @param user {@link User}
     * @param type {@link Types} type of {@link Machine}
     * @param date {@link String} date
     * @return {@link List<Booking>}
     */
    @Transactional(readOnly = true)
    @DebugMethod
    @NotNull List<Booking> findAllForGivenWeek(User user, Types type, String date);
}
```

BookingServiceImpl:

```
package com.adtimokhin.service;

import com.adtimokhin.model.*;
import com.adtimokhin.repository.BookingRepository;
import com.adtimokhin.util.Sorter;
import com.adtimokhin.util.time.DateFormatResolver;
import com.adtimokhin.util.time.TimetableContainer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Component
public class BookingServiceImpl implements BookingService {

    @Autowired
    private TimetableContainer timeTableContainer;

    @Autowired
    private BookingRepository bookingRepository;

    @Autowired
    private MachineTypeService machineTypeService;

    private final DateFormatResolver dateFormatResolver = new DateFormatResolver();

    @Override
    public List<Booking> getBookingsForMachine(Machine machine) {
        return machine.getBookings();
    }

    @Override
    public void save(Booking booking) {
```

```
        bookingRepository.save(booking);
    }

    @Override
    public void delete(Booking booking) {
        Machine machine = booking.getMachine();
        bookingRepository.delete(booking);

        timeTableContainer.updateTimeTable(machine.getId(),
(machine.getType().getTypeDescription().equals(Types.WASHING_MACHINE.toString()))); // todo:
update the updateTimeTable method.

    }

    @Override
    public List<Booking> findAllByUser(User user, Types type) {

        MachineType machineType = machineTypeService.findByType(type);
        List<Booking> bookings = bookingRepository.findAllByUserId(user.getId());
        List<Booking> temp = new ArrayList<>();
        for (Booking b :
                bookings) {
            if (b.getMachine().getType().equals(machineType)) {
                temp.add(b);
            }
        }
        return temp;
    }

    @Override
    public List<Booking> findAllByMachineId(int id) {
        return bookingRepository.findAllByMachineId(id);
    }

    @Override
    public Booking findById(int id) {
        return bookingRepository.findById(id).orElse(null);
    }

    @Override
    public Booking getBookingForMachineWithId(int id, String date) {
        Sorter sorter = new Sorter();
        List<Booking> bookings = findAllByMachineId(id);
        if (bookings == null) {
            return null;
        }
        bookings = sorter.sortBookings(sorter.clearDataBoo(bookings, date));
        Booking previousBooking = null;
        Booking currentBooking = null;
        for (Booking booking :
                bookings) {
            if (dateFormatResolver.isTimeBigger(date, booking.getStartDate())) {
                if (dateFormatResolver.isTimeBigger(booking.getEndDate(), date)) {
                    currentBooking = booking;
                    break;
                }
            }

            previousBooking = booking;
        }

        if (currentBooking == null) {
            return null;
        }
        if (previousBooking == null) {
            // todo: this is the place for the improvement.
            return null;
        }
        return previousBooking;
    }

    @Override
    public List<Booking> findAllMachineBookings(int userId, String date) {
        Sorter sorter = new Sorter();
        List<Booking> bookings = bookingRepository.findAllByUserId(userId);
        if (bookings == null) {
            return null;
        }
        bookings = sorter.sortBookings(sorter.clearDataBoo(bookings, date));
        List<Booking> todaysBookings = new ArrayList<>();
```

```
        for (Booking booking :
                bookings) {
            if (dateFormatResolver.datesMatch(booking.getStartDate(), date)) {
                todaysBookings.add(booking);
            }
        }

        return todaysBookings;

    }

    @Override
    public List<Booking> findAllForGivenWeek(User user, Types type, String date) {
        String[] weekdays = dateFormatResolver.getWeekDays(date);
        List<Booking> bookings = new ArrayList<>();
        for (String weekday : weekdays) {
            List<Booking> bookingsTemp =
bookingRepository.findAllByUserIdAndStartDateStartingWith(user.getId(), weekday);
            List<Booking> b = new ArrayList<>();
            for (Booking booking :
                    bookingsTemp) {
                if (booking.getMachine().getType().getTypeDescription().equals(type.toString()))
{
                    b.add(booking);
                }
            }
            bookingsTemp = b;
            if (bookingsTemp != null) {
                bookings.addAll(bookingsTemp);
            }

        }

        if (bookings.size() == 0) {
            return null;
        }

        return bookings;
    }
}
```

ComplaintService:

```
package com.adtimokhin.service;

import com.adtimokhin.aspect.DebugMethod;
import com.adtimokhin.aspect.NotEmptyArguments;
import com.adtimokhin.model.Complaint;
import com.adtimokhin.model.Types;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Service
@Transactional
public interface ComplaintService {

    /**
     * Saves {@link Complaint} in a database.
     *
     * @param complaint {@link Complaint} that is saved in the database.
     */
    @NotEmptyArguments
    void save(Complaint complaint);


    /**
     * Returns a list of all {@link Complaint} of a given {@link Types}.
     *
     * @param type {@link Types}
     * @return {@link List <Complaint>}
```

```
     */
    @Transactional(readOnly = true)
    @DebugMethod
    List<Complaint> findAll(Types type);


    /**
     * Deletes all {@link Complaint} in the database of a given {@link Types}.
     *
     * @param type {@link Types}
     */
    @DebugMethod
    void deleteAll(Types type);
}
```

ComplaintServiceImpl:

```
package com.adtimokhin.service;

import com.adtimokhin.model.Complaint;
import com.adtimokhin.model.Types;
import com.adtimokhin.repository.ComplaintRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Component
public class ComplaintServiceImpl implements ComplaintService {


    @Autowired
    private ComplaintRepository complaintRepository;

    @Override
    public void save(Complaint complaint) {
        complaintRepository.save(complaint);
    }

    @Override
    public List<Complaint> findAll(Types type) {
        return
complaintRepository.findAllByBooking_Machine_Type_TypeDescription(type.toString());
    }

    @Override
    public void deleteAll(Types type) {
        complaintRepository.deleteAllByBooking_Machine_Type_TypeDescription(type.toString());
    }
}
```

MachieService:

```
package com.adtimokhin.service;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.Types;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Service
```

```
public interface MachineService {

    /**
     * Finds all {@link Machine} by {@link Types}.
     *
     * @return {@link List <Machine>}.
     */
    @Transactional
    List<Machine> findAll(Types type);


    /**
     * Finds a {@link Machine} by its id.
     *
     * @param id {@link Machine#getId()}.
     * @return {@link Machine}
     */
    @Transactional(readOnly = true)
    Machine findById(int id);

    List<Booking> findBookings(int machineId);
}
```

MachineServiceImpl:

```
package com.adtimokhin.service;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.Types;
import com.adtimokhin.repository.MachineRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import java.util.ArrayList;
import java.util.List;

/**
 * @author adtimokhin
 * 01.01.2022
 **/
@Component
public class MachineServiceImpl implements MachineService {

    @Autowired
    private MachineRepository machineRepository;

    @Override
    public List<Machine> findAll(Types type) {
        List<Machine> machines = machineRepository.findAll();
        List<Machine> temp = new ArrayList<>();
        for (Machine m :
                machines) {
            if (m.getType().getTypeDescription().equals(type.toString())) {
                temp.add(m);
            }
        }
        return temp;
    }

    @Override
    public Machine findById(int id) {
        return machineRepository.getById(id);
    }

    @Override
    @Transactional
    public List<Booking> findBookings(int machineId){
        Machine machine = machineRepository.findById(machineId).orElse(null);
        if(machine == null){
            return null;
        }
        return machine.getBookings();
```

```
    }
}
```

MachineTypeService:

```
package com.adtimokhin.service;

import com.adtimokhin.model.MachineType;
import com.adtimokhin.model.Types;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Service
public interface MachineTypeService {

    /**
     * Finds a {@link MachineType} with the given value of {@link
MachineType#getTypeDescription()}
     * @param type value from {@link Types}
     * @return {@link MachineType}
     */
    @Transactional(readOnly = true)
    MachineType findByType(Types type);
}
```

MachineTypeServiceImpl:

```
package com.adtimokhin.service;

import com.adtimokhin.model.MachineType;
import com.adtimokhin.model.Types;
import com.adtimokhin.repository.MachineTypeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

/**
 * @author adtimokhin
 * 01.01.2022
 **/

@Component
public class MachineTypeServiceImpl implements MachineTypeService {
    @Autowired
    private MachineTypeRepository machineTypeRepository;


    @Override
    public MachineType findByType(Types type) {
        return machineTypeRepository.findByTypeDescription(type.toString());
    }
}
```

UserService:

```
package com.adtimokhin.service;

import com.adtimokhin.aspect.DebugMethod;
import com.adtimokhin.aspect.NotEmptyArguments;
import com.adtimokhin.model.User;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
```

```java
/**
 * @author adtimokhin
 * 16.10.2021
 **/
@Service
@Transactional
public interface UserService {


    /**
     * Finds a {@link User} that corresponds to a specific email.
     *
     * @param email Represents uniquely identifying email of {@link User}.
     * @return {@link User} that corresponds to email parameter.
     */
    @Transactional(readOnly = true)
    @DebugMethod
    User findByEmail(String email);

    /**
     * Saves {@link User} object in the database.
     *
     * @param user {@link User} that needs to be saved in the database.
     */
    @NotEmptyArguments
    void save(User user); // Objective 1.5

    /**
     * Returns all emailTokens of all {@link User} entities that have one.
     *
     * @return {@link List<String>} of email tokens.
     */
    @Transactional(readOnly = true)
    @DebugMethod
    List<String> findAllEmailTokens();

    /**
     * Sets {@link User#getEmailToken()} to null, which indicates that email is verified.
     *
     * @param user {@link User} for which we need to remove the token.
     */
    @DebugMethod
    void removeEmailToken(User user);

    /**
     * Finds a {@link User} by a email verification token.
     *
     * @param token {@link User#getEmailToken()}
     * @return {@link User}
     */
    @Transactional(readOnly = true)
    @DebugMethod
    User findByEmailToken(String token);
}
```

UserServiceImpl:

```java
package com.adtimokhin.service;

import com.adtimokhin.model.User;
import com.adtimokhin.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

/**
 * @author adtimokhin
 * 16.10.2021
 **/
@Component
public class UserServiceImpl implements UserService {

    @Autowired
```

```
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;


    @Override
    public User findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    @Override
    public void save(User user) {
        user.setPassword(passwordEncoder.encode(user.getPassword()));// Objective 13.2
        userRepository.save(user);
    }

    @Override
    public List<String> findAllEmailTokens() {
        List<User> users = userRepository.findAllByEmailTokenIsNotNullOrdOrderByEmailTokenAsc();
        return users.stream().map(User::getEmailToken).collect(Collectors.toCollection(() -> new
ArrayList<>(users.size())));
    }

    @Override
    public void removeEmailToken(User user) { // Objective 2.2
        user.setEmailToken(null);
        userRepository.save(user);
    }

    @Override
    public User findByEmailToken(String token) {
        return userRepository.findByEmailToken(token);
    }
}
```

BookingFileWriter:

```
package com.adtimokhin.util.backup.files;


import com.adtimokhin.model.Booking;

import java.io.BufferedWriter;
import java.io.IOException;

/**
 * @author adtimokhin
 * 23.10.2021
 **/

public class BookingFileWriter implements FileWorker {
    private BufferedWriter writer;

    @Override
    public void openConnection(String filename) throws IOException {
        writer = new BufferedWriter(new java.io.FileWriter(filename));
    }

    /**
     * Saves data about a machine booking into a file.
     *
     * @param booking {@link Booking} that is saved in a file.
     * @throws IOException if could not save data into the file.
     */
    public void writeBookingData(Booking booking) throws IOException {
        writer.write(booking.toString());
        writer.write("\n");
        writer.flush();
    }

    @Override
    public void closeConnection() throws IOException {
        writer.close();
    }
}
```

FileWorker:

```
package com.adtimokhin.util.backup.files;

import java.io.File;
import java.io.IOException;

/**
 * @author adtimokhin
 * 23.10.2021
 **/

public interface FileWorker {
    /**
     * Opens connects with a {@link File} that has its name equal to the parameter passed.
     * @param filename name of the file to open
     * @throws IOException if the {@link File} does not exist
     */
    void openConnection(String filename) throws IOException;


    /**
     * Closes connection with the currently-open {@link File}
     * @throws IOException if failed to closed the connection.
     */
    void closeConnection() throws IOException;
}
```

BackupUtil:

```
package com.adtimokhin.util.backup;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.Types;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.service.ComplaintService;
import com.adtimokhin.service.MachineService;
import com.adtimokhin.util.Sorter;
import com.adtimokhin.util.backup.files.BookingFileWriter;
import com.adtimokhin.util.time.DateFormatResolver;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.io.File;
import java.io.IOException;
import java.util.List;

/**
 * @author adtimokhin
 * 23.10.2021
 **/
@Component
public class BackupUtil {

    @Autowired
    private BookingService bookingService;

    @Autowired
    private MachineService machineService;

    @Autowired
    private ComplaintService complaintService;
```

```
    private DateFormatResolver dateFormatResolver = new DateFormatResolver();


    private final static Logger logger = LoggerFactory.getLogger(BackupUtil.class);


    private static final String ABSOLUTE_PATH =
"/Users/atimokhina/Desktop/washingDisasterSolver/";



    /**
     * Saves all {@link Booking} that have been scheduled for this day into a separate file and
deletes them from the database. This routine is called every day at 22:30 - a time after the
laundry room closes.
     *
     * @throws IOException if failed to save data into a file (refer to {@link
BookingFileWriter#writeBookingData(Booking)})
     */
    @Scheduled(cron = "0 54 20 * * ?", zone = "Europe/Moscow") // Objective 14.2
    public void saveBookingInformation() throws IOException { // Objective 15.1
        String date = dateFormatResolver.today();
        logger.trace("Initiating deletion of all bookings for {}.", date);
        Sorter sorter = new Sorter();
        BookingFileWriter bookingFileWriter = new BookingFileWriter();
        String today = date.replace(" ", "_");
        String fileName = "bookings_" + today.substring(0, (today.length() - 5)) + ".txt";

        File saveFile = new File(ABSOLUTE_PATH + fileName);
        saveFile.createNewFile();

        bookingFileWriter.openConnection(saveFile.getAbsolutePath());
        // save all data
        List<Machine> washingMachines = machineService.findAll(Types.WASHING_MACHINE);
        for (Machine machine :
                washingMachines) {
            List<Booking> washingMachineBookings =
sorter.clearDataBoo(bookingService.getBookingsForMachine(machine), date);
            for (Booking booking :
                    washingMachineBookings) {
                bookingFileWriter.writeBookingData(booking);
                delete(booking);
            }
        }

        List<Machine> dryingMachines = machineService.findAll(Types.DRYING_MACHINE);
        for (Machine machine :
                dryingMachines) {
            List<Booking> dryingMachineBookings =
sorter.clearDataBoo(bookingService.getBookingsForMachine(machine), date);
            for (Booking booking :
                    dryingMachineBookings) {
                bookingFileWriter.writeBookingData(booking);
                delete(booking);
            }
        }

        bookingFileWriter.closeConnection();
        logger.trace("All bookings were deleted.");
    }

    /**
     * Deletes all complaints from a database at 23:30 every day.
     */
    @Scheduled(cron = "0 30 23 * * ?", zone = "Europe/London") // Objective 14.2
    public void deleteAllComplaints() {
        // Objective 15.1
        logger.trace("Initiating deletion of all complaints.");
        complaintService.deleteAll(Types.WASHING_MACHINE);
        complaintService.deleteAll(Types.DRYING_MACHINE);
        logger.trace("All complaints were deleted.");
    }

    private void delete(Booking booking) {
        // Objective 15.1
        bookingService.delete(booking);
    }
}
```

SimpleEmailSender:

```
package com.adtimokhin.util.email;

import com.adtimokhin.model.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMailMessage;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Component;

/**
 * @author adtimokhin
 * 11.11.2021
 **/
@Component
public class SimpleEmailSender {
    @Autowired
    private JavaMailSender javaMailSender;

    @Value("${mail.username}") // apparently, we can access the .properties file from other parts
of the application
    private String senderEmail;

    private static final String HOSTNAME = "localhost:8080";

    /**
     * Sends an email verification letter to user.
     *
     * @param recipient is {@link User} to which we will send our verification letter.
     **/
    @Async // Objective 14.1
    public void sendEmailVerificationEmail(User recipient) { // Objective 1.6
        MimeMailMessage message = new MimeMailMessage(javaMailSender.createMimeMessage());
        message.setFrom(senderEmail); // this is how we set the sender now
        message.setTo(recipient.getEmail()); // this is how you set recipients now
        message.setSubject("Email verification"); // this is going to be the subject of your
email
        message.setText("Please, follow this link to verify your email: \n"
                + HOSTNAME + "/verify/" + recipient.getEmailToken());// this text will be in your
email's body
        javaMailSender.send(message.getMimeMessage());

    }
}
```

DateFormatResolver:

```
package com.adtimokhin.util.time;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.util.Calendar;
import java.util.GregorianCalendar;


/**
 * @author adtimokhin
 * 16.10.2021
 **/
public class DateFormatResolver {

    private final static Logger logger = LoggerFactory.getLogger(DateFormatResolver.class);


    public static final String DATE_FORMAT = "YYYY MM dd HH:mm";

    public static final String DUMMY_DATE = "0000 00 00 00:00";
```

```
    public static final int YEAR = 0;
    public static final int MONTH = 1;
    public static final int DAY = 2;
    public static final int HOUR = 3;
    public static final int MINUTE = 4;

    public static final long MINUTES_BEFORE_CANCEL = 60 * 2; // 2 hours, but in minutes.

    public static final long MAX_LENGTH_FOR_BOOKING = 60 * 2 + 20; // 140 minutes.
    public static final long MAX_LENGTH_FOR_BOOKING_WASHING_MACHINE = 60 * 2 + 20; // 140
minutes.
    public static final long MAX_LENGTH_FOR_BOOKING_DRYING_MACHINE = 60 * 2 + 20; // 140 minutes.


    /**
     * This method extracts integer value from a String in format supported by the system
     * represented by {@value DATE_FORMAT}.
     * <p>
     * That date should be checked for format before passing into this method.
     *
     * @param date {@link String} that represents date from which to extract part.
     * @param part part of the date to extract. That includes {@value YEAR},{@value MONTH},
{@value DAY},
     *             {@value HOUR}, {@value MINUTE}.
     * @return {@link Integer} that represents that date part.
     */
    public int getDatePart(String date, int part) {
        String[] parts = date.split(" ");
        if (part < 3) {
            try {
                return Integer.parseInt(parts[part]);
            } catch (Exception e) {
                return -1;
            }
        } else {
            String[] hoursMinutes = parts[3].split(":");
            try {
                return Integer.parseInt(hoursMinutes[part - 3]);
            } catch (Exception e) {
                return -1;
            }
        }
    }

    /**
     * This method checks that two dates have equal values for {@link DateFormatResolver#HOUR}
and {@link DateFormatResolver#MINUTE}.
     *
     * @param start {@link String} that represents a start date
     * @param end   {@link String} that represents a finish date
     * @return true if two dates have equal values for {@link DateFormatResolver#HOUR} and {@link
DateFormatResolver#MINUTE}.
     */
    public boolean datesMatch(String start, String end)      {
        if (getDatePart(start, MINUTE) == getDatePart(end, MINUTE)) {
            return getDatePart(start, HOUR) == getDatePart(end, HOUR);
        }
        return false;
    }

    /**
     * Checks if the start time is larger than the end time (works if the {@link
DateFormatResolver#DAY} is the same).
     *
     * @param startHour   start time's hour value
     * @param startMinute start time's minute value
     * @param endHour     end time's hour value
     * @param endMinute   end time's minute value
     * @return true if start time is larger than the end time.
     */
    public boolean isTimeBigger(int startHour, int startMinute, int endHour, int endMinute) {
        if (startHour > endHour) {
            return true;
        }
        return startHour == endHour && startMinute >= endMinute;
    }

    /**
     * Checks if the start time is larger than the end time (works if the {@link
DateFormatResolver#DAY} is the same).
```

```
     *
     * @param timeOne {@link String} that represents start time in {@value DATE_FORMAT}
     * @param timeTwo {@link String} that represents end time in {@value DATE_FORMAT}
     * @return true if start time is larger than the end time.
     */
    public boolean isTimeBigger(String timeOne, String timeTwo) {
        return isTimeBigger(getDatePart(timeOne, HOUR),
                getDatePart(timeOne, MINUTE),
                getDatePart(timeTwo, HOUR),
                getDatePart(timeTwo, MINUTE));
    }

    /**
     * Converts input into a date in the format {@value DATE_FORMAT}
     *
     * @param hour   int value for {@link DateFormatResolver#HOUR}
     * @param minute int value for {@link DateFormatResolver#MINUTE}
     * @param date   {@link String} that represents a date to which the other inputs would be
attached. This input must be in the format: {@value DATE_FORMAT}.
     * @return new date in the format {@value DATE_FORMAT}.
     */
    public String resolveTimeForDate(String hour, String minute, String date) {
        if (hour.length() == 1) {
            hour = "0" + hour;
        }
        if (minute.length() == 1) {
            minute = "0" + minute;
        }

        return extractYearMonthDay(date) + hour + ":" + minute;
    }

    /**
     * Extracts {@link String} that contains {@link DateFormatResolver#YEAR}, {@link
DateFormatResolver#MONTH} and {@link DateFormatResolver#DAY} from the input.
     *
     * @param date {@link String} in the format {@value DATE_FORMAT}
     * @return {@link String} that contains {@link DateFormatResolver#YEAR}, {@link
DateFormatResolver#MONTH} and {@link DateFormatResolver#DAY}
     */
    public String extractYearMonthDay(String date) {
        return date.substring(0, 11);
    }

    /**
     * Checks if the first inputted date is smaller than the second one.
     *
     * @param dateOne {@link String} in format {@value DATE_FORMAT}
     * @param dateTwo {@link String} in format {@value DATE_FORMAT}
     * @return true if the first inputted date is smaller than the second one.
     */
    // Objective 5.3.3, 5.3.5, 5.3.6
    public boolean isDateBeforeAnother(String dateOne, String dateTwo) {
        if (getDatePart(dateOne, YEAR) == getDatePart(dateTwo, YEAR)) {
            if (getDatePart(dateOne, MONTH) == getDatePart(dateTwo, MONTH)) {
                if (getDatePart(dateOne, DAY) == getDatePart(dateTwo, DAY)) {
                    if (getDatePart(dateOne, HOUR) == getDatePart(dateTwo, HOUR)) {
                        return getDatePart(dateOne, MINUTE) < getDatePart(dateTwo, MINUTE);
                    } else {
                        return getDatePart(dateOne, HOUR) < getDatePart(dateTwo, HOUR);
                    }
                } else {
                    return getDatePart(dateOne, DAY) < getDatePart(dateTwo, DAY);
                }
            } else {
                return getDatePart(dateOne, MONTH) < getDatePart(dateTwo, MONTH);
            }
        } else {
            return getDatePart(dateOne, YEAR) < getDatePart(dateTwo, YEAR);
        }
    }

    /**
     * Does the same as the {@link DateFormatResolver#isDateBeforeAnother(String, String)},
except it does not check {@link DateFormatResolver#HOUR} and {@link DateFormatResolver#MINUTE}
values.
     *
     * @param dateOne {@link String} in format {@value DATE_FORMAT}
     * @param dateTwo {@link String} in format {@value DATE_FORMAT}
     * @return true if the first inputted date is smaller than the second one.
```

```
     */
    public boolean isDateBefore(String dateOne, String dateTwo) {
        if (getDatePart(dateOne, YEAR) == getDatePart(dateTwo, YEAR)) {
            if (getDatePart(dateOne, MONTH) == getDatePart(dateTwo, MONTH)) {
                return getDatePart(dateOne, DAY) < getDatePart(dateTwo, DAY);
            } else {
                return getDatePart(dateOne, MONTH) < getDatePart(dateTwo, MONTH);
            }
        } else {
            return getDatePart(dateOne, YEAR) < getDatePart(dateTwo, YEAR);
        }
    }

    /**
     * @return current date in the format {@value DATE_FORMAT}
     */
    public String today() {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy MM dd HH:mm");
        LocalDateTime now = LocalDateTime.now();
        return dtf.format(now);
    }

    /**
     * Checks if the two dates are the same except for {@link DateFormatResolver#HOUR} and {@link
DateFormatResolver#MINUTE} values.
     *
     * @param timeOne {@link String} in format {@value DATE_FORMAT}
     * @param timeTwo {@link String} in format {@value DATE_FORMAT}
     * @return true if the two dates are the same
     */
    public boolean onTheSameDay(String timeOne, String timeTwo) {
        // for this class we assume that we use DATE_FORMAT
        int dayOne = getDatePart(timeOne, DAY);
        int dayTwo = getDatePart(timeTwo, DAY);

        if (dayOne == dayTwo) {

            int monthOne = getDatePart(timeOne, MONTH);
            int monthTwo = getDatePart(timeTwo, MONTH);

            if (monthOne == monthTwo) {

                int yearOne = getDatePart(timeOne, YEAR);
                int yearTwo = getDatePart(timeTwo, YEAR);

                return yearOne == yearTwo;
            }
        }
        return false;
    }

    /**
     * Checks if the two dates have distance between them larger than the passed value
     *
     * @param dateOne  {@link String} in format {@value DATE_FORMAT}
     * @param dateTwo  {@link String} in format {@value DATE_FORMAT}
     * @param distance least allowed separation between two dates.
     * @return true if the two dates have distance between them larger than the passed value
     */
    public boolean areFarEnoughInTime(String dateOne, String dateTwo, long distance) { //
Objective 7.2.3
        return getMinutesBetweenDates(dateOne, dateTwo) > (distance);
    }

    /**
     * Checks if the date follows the format of {@value DATE_FORMAT}.
     *
     * @param date {@link String} that we check to be in the correct format or not.
     * @return true if the date follows the format of {@value DATE_FORMAT}.
     */
    public boolean appropriateFormat(String date) {
        String[] parts = date.split(" ");
        String[] origParts = DATE_FORMAT.split(" ");

        if (parts.length != origParts.length) {
            return false;
        }

        for (int i = 0; i < origParts.length - 1; i++) {
            try {
```

```
                Integer.parseInt(parts[i]);
                if (parts[i].length() != origParts[i].length()) {
                    return false;
                }
            } catch (Exception e) {
                return false;
            }
        }

        String[] hourMinParts = parts[parts.length - 1].split(":");
        String[] origHourMinParts = origParts[origParts.length - 1].split(":");

        if (hourMinParts.length != origHourMinParts.length) {
            return false;
        }

        for (int i = 0; i < origHourMinParts.length; i++) {
            try {
                Integer.parseInt(hourMinParts[i]);
                if (hourMinParts[i].length() != origHourMinParts[i].length()) {
                    return false;
                }
            } catch (Exception e) {
                return false;
            }
        }

        return true;
    }

    /**
     * Checks if date passed in the format defined by {@value DATE_FORMAT} and that each field is
in military time format.
     *
     * @param date {@link String} that needs to be validated.
     * @return true if the date is in the correct form.
     */
    public boolean appropriateFormatWithDash(String date) { // Objective 4.3.1
        if (date.length() != "2021-01-01".length()) {
            return false;
        }

        String[] parts = date.split("-");

        if (parts.length != 3) {
            return false;
        }

        for (int i = 0; i < parts.length; i++) {
            try {
                int integer = Integer.parseInt(parts[i]);
                if (i == 0) {
                    if (integer < 2020 || integer > 3000) {
                        return false;
                    }
                } else {
                    if (integer < 0 || integer > 32) {
                        return false;
                    }
                }
            } catch (Exception e) {
                return false;
            }
        }

        return true;
    }

    /**
     * Finds all dates in format {@value DATE_FORMAT} that occur on the same week as the passed
date.
     *
     * @param date {@link String} in format {@value DATE_FORMAT}
     * @return {@link String[]} of {@value DATE_FORMAT} values that occur on the same week as the
passed date.
     */
    public String[] getWeekDays(String date) {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy MM dd");
        int year = getDatePart(date, YEAR);
        int month = getDatePart(date, MONTH) - 1;
```

```
        int day = getDatePart(date, DAY);

        Calendar cal = Calendar.getInstance();
        cal.setTime(new GregorianCalendar(year, month, day).getTime());//Set specific Date if you
want to

        String[] week = new String[7];

        cal.set(Calendar.DAY_OF_WEEK, Calendar.SUNDAY);
        week[6] = dtf.format(cal.getTime().toInstant()
                .atZone(ZoneId.systemDefault())
                .toLocalDateTime());

        for (int i = Calendar.MONDAY; i <= Calendar.SATURDAY; i++) {
            cal.set(Calendar.DAY_OF_WEEK, i);
            week[i - 2] = dtf.format(cal.getTime().toInstant()
                    .atZone(ZoneId.systemDefault())
                    .toLocalDateTime());
        }
        return week;
    }

    /**
     * Finds the number of minutes between two passed dates.
     *
     * @param dateOne {@link String} in format {@value DATE_FORMAT}
     * @param dateTwo {@link String} in format {@value DATE_FORMAT}
     * @return the number of minutes between two passed dates.
     */
    public long getMinutesBetweenDates(String dateOne, String dateTwo) {
        int minuteDifference = 0;

        // comparing years
        int yearDate = getDatePart(dateOne, YEAR);
        int yearToday = getDatePart(dateTwo, YEAR);
        minuteDifference += (yearDate - yearToday) * 365 * 24 * 60;

        //comparing months
        int monthDate = getDatePart(dateOne, MONTH);
        int monthToday = getDatePart(dateTwo, MONTH);
        minuteDifference += (monthDate - monthToday) * 30 * 24 * 60;

        //comparing days
        int dayDate = getDatePart(dateOne, DAY);
        int dayToday = getDatePart(dateTwo, DAY);
        minuteDifference += (dayDate - dayToday) * 24 * 60;

        //comparing hours
        int hourDate = getDatePart(dateOne, HOUR);
        int hourToday = getDatePart(dateTwo, HOUR);
        minuteDifference += (hourDate - hourToday) * 60;

        //comparing minutes
        int minuteDate = getDatePart(dateOne, MINUTE);
        int minuteToday = getDatePart(dateTwo, MINUTE);
        minuteDifference += (minuteDate - minuteToday);


        return minuteDifference;
    }


}
```

Timetable:

```
package com.adtimokhin.util.time;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.bookings.WashingMachineBooking;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.util.Sorter;
import lombok.Getter;

import java.util.ArrayList;
import java.util.List;
```

```java
import static com.adtimokhin.util.time.DateFormatResolver.HOUR;
import static com.adtimokhin.util.time.DateFormatResolver.MINUTE;


/**
 * @author adtimokhin
 * 16.10.2021
 **/


public class Timetable {

    @Getter
    private Machine machine;

    private final static DateFormatResolver DATE_FORMAT_RESOLVER = new DateFormatResolver();

    @Getter
    private List<TimePeriod> timePeriods;

    /**
     * Represents a minimal length of a slot that a user can book. It must be in minutes, and it
must be a factor of 60.
     */
    public static final int SLOT_SIZE = 30; // in minutes

    /**
     * Initial index of a period
     */
    private int periodId = 0;


    /**
     * Method that is used to calculate free and occupied time slots based on data stored in a
database for a given {@link Machine}
     *
     * @param machine        {@link Machine} for which we must calculate the timetable
     * @param bookingService {@link BookingService} used to retrieve all data needed from the
database
     * @param date           {@link String} in format {@value DateFormatResolver#DATE_FORMAT}
that is used to find {@link WashingMachineBooking} for relevant date.
     * @param sorter         {@link Sorter} that is used to perform sorts.
     */
    public void generateTimeTable(Machine machine, BookingService bookingService, String date,
Sorter sorter) {

        this.machine = machine;

        List<Booking> bookings =
sorter.sortBookings(sorter.clearDataBoo(bookingService.getBookingsForMachine(machine), date));

        int size = bookings.size();

        timePeriods = new ArrayList<>(32);


        timePeriods.add(new TimePeriod(0, 0, 6, 0, false, -1));

        for (int i = 0; i < size; i++) {
            Booking current = bookings.get(i);
            // create TimePeriod for current Booking
            String startDate = current.getStartDate();
            String endDate = current.getEndDate();

            String oldEnd = timePeriods.get(timePeriods.size() - 1).getDate();

            if (!DATE_FORMAT_RESOLVER.datesMatch(startDate, oldEnd)) {
                createSlots(oldEnd, startDate, true);
            }
            createSlots(startDate, endDate, false);
        }

        String oldEnd = timePeriods.get(timePeriods.size() - 1).getDate();
        String startDate = "0000 00 00 22:00";

        if (!DATE_FORMAT_RESOLVER.datesMatch(startDate, oldEnd)) {
            createSlots(oldEnd, startDate, true);
        }
```

```
        timePeriods.add(new TimePeriod(22, 0, 23, 59, false, -1));
    }

    /**
     * Method used to create parameterised objects of class {@link TimePeriod}
     *
     * @param begin  {@link String} in format {@value DateFormatResolver#DATE_FORMAT}. Start time
     * @param finish {@link String} in format {@value DateFormatResolver#DATE_FORMAT}. Finish
time
     * @param isFree boolean that tells if the {@link TimePeriod} is vacant or not.
     */
    private void createSlots(String begin, String finish, boolean isFree) {

        int hourDiff = DATE_FORMAT_RESOLVER.getDatePart(finish, HOUR) -
DATE_FORMAT_RESOLVER.getDatePart(begin, HOUR);
        int minuteDiff = DATE_FORMAT_RESOLVER.getDatePart(finish, MINUTE) -
DATE_FORMAT_RESOLVER.getDatePart(begin, MINUTE);

        int timeslots = (hourDiff * 60 + minuteDiff) / SLOT_SIZE;

        int startHour = DATE_FORMAT_RESOLVER.getDatePart(begin, HOUR);
        int startMinute = DATE_FORMAT_RESOLVER.getDatePart(begin, MINUTE);

        int finishMinute;
        int finishHour = startHour;

        for (int i = 0; i < timeslots; i++) {
            finishMinute = startMinute + SLOT_SIZE;
            if (finishMinute == 60) {
                finishMinute = 0;
                finishHour++;
            }
            timePeriods.add(new TimePeriod(startHour, startMinute, finishHour, finishMinute,
isFree, periodId++));

            startHour = finishHour;
            startMinute = finishMinute;
        }

    }

    /**
     * Checks if the time in the given range has no bookings in it.
     *
     * @param startHour   {@link DateFormatResolver#HOUR}
     * @param startMinute {@link DateFormatResolver#MINUTE}
     * @param endHour     {@link DateFormatResolver#HOUR}
     * @param endMinute   {@link DateFormatResolver#MINUTE}
     * @return true if the time in the given range has no bookings in it.
     */
    public boolean isFreeBetween(int startHour, int startMinute, int endHour, int endMinute) {
        boolean start = false;

        for (TimePeriod timePeriod : timePeriods) {
            if (!start) {
                // then we compare starts of the periods
                if (startHour == timePeriod.START_HOUR) {
                    if (startMinute == timePeriod.START_MINUTE) {
                        if (timePeriod.IS_FREE) {
                            start = true;
                        } else {
                            return false;
                        }
                    }
                }

                if (start) { // we also need to check if this slot is also an end to the booking
                    if (endHour == timePeriod.END_HOUR) {
                        if (endMinute == timePeriod.END_MINUTE) {
                            return true;
                        }
                    }
                }
            } else {
                if (!timePeriod.IS_FREE) {
                    return false;
                }
                if (endHour == timePeriod.END_HOUR) {
                    if (endMinute == timePeriod.END_MINUTE) {
                        return true;
```

```
                    }
                }
            }
        }
        return false;
    }

    @Override
    public String toString() {
        return "TimeTable{" +
                "timePeriods=" + timePeriods +
                '}';
    }

    /**
     * Class that represents a single time slot that user can interact with.
     */
    public static class TimePeriod {
        private final int START_HOUR;
        private final int START_MINUTE;
        private final int END_HOUR;
        private final int END_MINUTE;
        @Getter
        private final boolean IS_FREE;
        @Getter
        private final int ID;


        public TimePeriod(int startHour, int startMinute, int endHour, int endMinute, boolean
isFree, int id) {
            this.START_HOUR = startHour;
            this.START_MINUTE = startMinute;
            this.END_HOUR = endHour;
            this.END_MINUTE = endMinute;
            this.IS_FREE = isFree;
            this.ID = id;
        }

        @Override
        public String toString() {
            return "TimePeriod{" +
                    "startHour=" + START_HOUR +
                    ", startMinute=" + START_MINUTE +
                    ", endHour=" + END_HOUR +
                    ", endMinute=" + END_MINUTE +
                    ", isFree=" + IS_FREE +
                    '}';
        }

        public String getTimeBounds() {
            StringBuilder resultString = new StringBuilder();
            String s = String.valueOf(START_HOUR);
            if (s.length() == 1) {
                resultString.append("0");
            }
            resultString.append(s).append(":");
            s = String.valueOf(START_MINUTE);
            if (s.length() == 1) {
                resultString.append("0");
            }
            resultString.append(s).append("--");
            s = String.valueOf(END_HOUR);
            if (s.length() == 1) {
                resultString.append("0");
            }
            resultString.append(s).append(":");

            s = String.valueOf(END_MINUTE);
            if (s.length() == 1) {
                resultString.append("0");
            }
            resultString.append(s);
            return resultString.toString();
        }

        public String getDate() {
            String hour = String.valueOf(END_HOUR);
            if (END_HOUR < 10) {
                hour = "0" + hour;
            }
```

```
            String minute = String.valueOf(END_MINUTE);
            if (END_MINUTE < 10) {
                minute = "0" + minute;
            }

            return "0000 00 00 " + hour + ":" + minute;
        }


    }
}
```

TimetableContainer:

```
package com.adtimokhin.util.time;

import com.adtimokhin.model.Machine;
import com.adtimokhin.model.Types;
import com.adtimokhin.model.machine.DryingMachine;
import com.adtimokhin.model.machine.WashingMachine;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.service.MachineService;
import com.adtimokhin.util.Sorter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

/**
 * @author adtimokhin
 * 16.10.2021
 **/

@Component
public class TimetableContainer {

    @Autowired
    private Sorter sorter;

    @Autowired
    private BookingService bookingService;

    @Autowired
    private MachineService machineService;

    public List<Timetable> washingMachineTimetables;
    public List<Timetable> dryingMachineTimetables;


    private String day;


    private final DateFormatResolver DATE_FORMAT_RESOLVER = new DateFormatResolver();

    /**
     * Gets a {@link Timetable} for a given {@link Machine}
     *
     * @param id                {@link WashingMachine#getId()}
     * @param isWashingMachine boolean that tells if the machine type is {@link
Types#WASHING_MACHINE}
     * @return a {@link Timetable} for a given {@link Machine}
     */
    public Timetable getTimeTable(int id, boolean isWashingMachine) {
        if (isWashingMachine) {
            if (washingMachineTimetables == null) {
                initializeLists();
            }
            try {
                return washingMachineTimetables.get(id - 1);
            } catch (IndexOutOfBoundsException e) {
                return null;
            }

        }
        if (dryingMachineTimetables == null) {
            initializeLists();
```

```
        }
        try {
            return dryingMachineTimetables.get(id - 1);
        } catch (IndexOutOfBoundsException e) {
            return null;
        }

    }

    /**
     * Regenerates a {@link Timetable} for a given {@link Machine}
     *
     * @param id               {@link WashingMachine#getId()}
     * @param isWashingMachine boolean that tells if the machine type is {@link
Types#WASHING_MACHINE}
     */
    public void updateTimeTable(int id, boolean isWashingMachine) {
            if (washingMachineTimetables == null) {
                initializeLists();
            }
            Timetable timeTable = new Timetable();
            Machine washingMachine = machineService.findById(washingMachineTimetables.get(id -
1).getMachine().getId());
            if (washingMachine == null) {
                return;
            }
            timeTable.generateTimeTable(washingMachine, bookingService, day, sorter);
            washingMachineTimetables.set((id - 1), timeTable);
    }

    /**
     * @return a list of {@link Timetable} for all {@link WashingMachine}.
     */
    public List<Timetable> getWashingMachineTimeTables() {
        if (washingMachineTimetables == null) {
            initializeLists();
        }
        return washingMachineTimetables;
    }

    /**
     * @return a list of {@link Timetable} for all {@link DryingMachine}.
     */
    public List<Timetable> getDryingMachineTimeTables() {
        if (dryingMachineTimetables == null) {
            initializeLists();
        }
        return dryingMachineTimetables;
    }

    /**
     * Initialises a {@link Timetable} for all {@link Machine}
     */
    private void initializeLists() {
        List<Machine> washingMachineList = machineService.findAll(Types.WASHING_MACHINE);
        List<Machine> dryingMachineList = machineService.findAll(Types.DRYING_MACHINE);

        if (day == null) {
            day = DATE_FORMAT_RESOLVER.today();
        }

        washingMachineTimetables = new ArrayList<>();
        dryingMachineTimetables = new ArrayList<>();

        for (Machine washingMachine :
                washingMachineList) {
            Timetable timeTable = new Timetable();
            timeTable.generateTimeTable(washingMachine, bookingService, day, sorter);
            washingMachineTimetables.add(timeTable);
        }

        for (Machine dryingMachine :
                dryingMachineList) {
            Timetable timeTable = new Timetable();
            timeTable.generateTimeTable(dryingMachine, bookingService, day, sorter);
            dryingMachineTimetables.add(timeTable);
        }
    }

    /**
```

```
     * Updates all {@link Timetable} objects to represent timetables for another date.
     *
     * @param newDay {@link String} in format {@value DateFormatResolver#DATE_FORMAT}
     */
    public void changeDay(String newDay) {
        day = newDay;
        initializeLists();
    }

    /**
     * @return a {@link String} in format {@value DateFormatResolver#DATE_FORMAT} that represents
a date for which all {@link Timetable} objects display data.
     */
    public String getDay() {
        if (day == null) {
            day = DATE_FORMAT_RESOLVER.today();
        }
        return day;
    }
}
```

SimpleEmailTokenGenerator:

```
package com.adtimokhin.util.token;

import com.adtimokhin.exceptions.TokenOverflowException;
import com.adtimokhin.model.User;
import com.adtimokhin.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.*;

/**
 * @author adtimokhin
 * 11.11.2021
 **/

@Component
public class SimpleEmailTokenGenerator implements TokenGenerator {

    /**
     * Represents a length of the token that verifies emails
     */
    private final static int TOKEN_LENGTH = 5;

    /**
     * List of symbols that can be used to create an email verification token
     */
    private final int[] symbols;

    @Autowired
    private UserService userService;

    /**
     * Method generates a list of symbols that would be used to create email tokens.
     */
    public SimpleEmailTokenGenerator() {
        symbols = new int[10];
        int k = 48;
        for (int i = 0; i < 10; i++) {
            symbols[i] = k + i;
        }
    }

    /**
     * Generates a unique email verification token for {@link User}.
     *
     * @return {@link String} that represents a {@link User#getEmailToken()}
     */
    @Override
    public String generate() { // Objective 1.6.1

        List<String> tokens = userService.findAllEmailTokens();

        if (tokens.size() == Math.pow((TOKEN_LENGTH), symbols.length)) {
            throw new TokenOverflowException("Overflow of email tokens.");
```

```
        }

        List<Integer> chars = new ArrayList<>();
        for (int symbol : symbols) {
            chars.add(symbol);
        }

        return generateSymbol(chars, 0, tokens, "");
    }

    /**
     * Method that methodically generates a symbol for the token until the token of length
{@value TOKEN_LENGTH} is generated.
     *
     * @param chars        {@link List} of all possible symbols from which the token can be made.
     * @param pos          number of a digit that is currently generated.
     * @param viableTokens tokens that could potentially be equal to the one that is being
generated.
     * @param currentToken {@link String} that represents the symbols
     * @return {@link String} that represents a unique {@value TOKEN_LENGTH}-long email token.
     */
    private String generateSymbol(List<Integer> chars, int pos, List<String> viableTokens, String
currentToken) {

        if (pos == TOKEN_LENGTH) {
            return currentToken;
        }

        // generate a list of chars in some random order
        Collections.shuffle(chars);

        for (int i = 0; i < chars.size(); i++) {
            int charToTry = Integer.parseInt(Character.toString(chars.get(i)));
            List<String> matchDigit = new ArrayList<>();
            for (String usedToken :
                    viableTokens) {
                if (Integer.parseInt(Character.toString(usedToken.charAt(pos))) == (charToTry)) {
                    matchDigit.add(usedToken);
                }
            }
            if (matchDigit.size() == 0) {
                currentToken += charToTry;
                return currentToken + generateRest(currentToken.length(), chars);
            } else if (matchDigit.size() != Math.pow(chars.size(), TOKEN_LENGTH - pos - 1)) {
                String potentialToken = generateSymbol(chars, pos + 1, matchDigit, currentToken +
String.valueOf(charToTry)); // Recursion
                if (potentialToken != null) {
                    return potentialToken;
                }
            }
        }
        if (pos == 0) {
            throw new TokenOverflowException("Overflow of email tokens.");
        }
        return null;


    }


    /**
     * Method finishes generating a token at the point where it is irrelevant what symbols to
include into the token.
     *
     * @param startPosition represents a position from which we carry on building.
     * @param symbols       represents a list of symbols from which the remainder of the token
can be build.
     * @return a {@link String} of length {@link SimpleEmailTokenGenerator#TOKEN_LENGTH} -
{@param startPosition}, consisting of random characters from {@link
SimpleEmailTokenGenerator#symbols}.
     */
    private String generateRest(int startPosition, List<Integer> symbols) {
        StringBuilder stringBuilder = new StringBuilder();

        for (int i = 0; i < TOKEN_LENGTH - startPosition; i++) {
            stringBuilder.append(Integer.parseInt(Character.toString(symbols.get(i))));
        }

        return stringBuilder.toString();
```

```
    }
}
```

TokenGenerator:

```
package com.adtimokhin.util.token;

import org.springframework.stereotype.Component;

/**
 * @author adtimokhin
 * 11.11.2021
 **/

@Component
public interface TokenGenerator {



    /**
     * @return a {@link String} that represents a unique token.
     */
    String generate();

}
```

Sorter:

```
package com.adtimokhin.util;

import com.adtimokhin.model.bookings.Booking;
import com.adtimokhin.model.bookings.DryingMachineBooking;
import com.adtimokhin.model.bookings.WashingMachineBooking;
import com.adtimokhin.util.time.DateFormatResolver;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

/**
 * @author adtimokhin
 * 16.10.2021
 **/

@Component
public class Sorter {
    private static final DateFormatResolver dateTimeResolver = new DateFormatResolver();

    /**
     * Method that sorts a passed {@link List<com.adtimokhin.model.Booking>} in ascending order
     * of {@link com.adtimokhin.model.Booking#getEndDate()}.
     * <p>
     * Method uses merge sort algorithm.
     *
     * @param unsortedList {@link List<com.adtimokhin.model.Booking>} that will be sorted.
     * @return a sorted {@link List<com.adtimokhin.model.Booking>}.
     */
    public List<Booking> sortBookings(List<Booking> unsortedList) {
        Booking[] list = unsortedList.toArray(new Booking[0]);
        mergeSort(list, list.length);
        return Arrays.asList(list);
    }

    /**
     * Method removes from the list passed all {@link com.adtimokhin.model.Booking} that do not
match the date parameter.
     *
     * @param unclearList {@link List<com.adtimokhin.model.Booking>} that will be cleaned from
{@link com.adtimokhin.model.Booking}
     *                    that don't have the same date set for {@link
com.adtimokhin.model.Booking#getStartDate()} as in the parameters.
     * @param date        {@link String} that represents a date defined by a format specified by
{@link DateFormatResolver#DATE_FORMAT}.
     * @return a cleared {@link List<Booking>}.
     */
    public List<Booking> clearDataBoo(List<Booking> unclearList, String date) {
```

```
        List<Booking> bookings = new ArrayList<>();
        for (Booking booking : unclearList) {
            if (dateTimeResolver.onTheSameDay(date, booking.getStartDate())) {
                bookings.add(booking);
            }
        }
        return bookings;
    }

    /**
     * Method that finds {@link com.adtimokhin.model.Booking} that are set on a date that is not
yet has been passed.
     *
     * @param unclearList {@link List<Booking>} from which we will find a list of {@link Booking}
     *                    that have {@link com.adtimokhin.model.Booking#getEndDate()} set after
the {@link DateFormatResolver#today()}.
     * @return {@link List<Booking>} that contains only still relevant {@link Booking}.
     */
    public List<Booking> getRelevantBookings(List<Booking> unclearList) { // Objective 6.2
        List<Booking> bookings = new ArrayList<>();
        String today = dateTimeResolver.today();
        for (com.adtimokhin.model.Booking booking : unclearList) {
            if (!dateTimeResolver.isDateBeforeAnother(booking.getEndDate(),
                    today)) {
                bookings.add(booking);
            }
        }
        return bookings;
    }

    private void mergeSort(Booking[] bookings, int listSize) {
        if (listSize < 2) {
            return;
        }
        int mid = listSize / 2;
        Booking[] leftArray = new Booking[mid];
        Booking[] rightArray = new Booking[listSize - mid];
        for (int i = 0; i < mid; i++) {
            leftArray[i] = bookings[i];
        }
        for (int i = mid; i < listSize; i++) {
            rightArray[i - mid] = bookings[i];
        }
        mergeSort(leftArray, mid);
        mergeSort(rightArray, listSize - mid);
        merge(bookings, leftArray, rightArray, mid, listSize - mid);
    }

    public static void merge(
            Booking[] bookings, Booking[] leftArray, Booking[] rightArray, int
            left, int right) {
        int i = 0, j = 0, k = 0;
        while (i < left && j < right) {
            if (dateTimeResolver.isTimeBigger(leftArray[i].getEndDate(),
rightArray[j].getEndDate())) {
                bookings[k++] = leftArray[i++];
            } else {
                bookings[k++] = rightArray[j++];
            }
        }
        while (i < left) {
            bookings[k++] = leftArray[i++];
        }
        while (j < right) {
            bookings[k++] = rightArray[j++];
        }
    }
}
```

BookingError:

```
package com.adtimokhin.validation.errors;

import lombok.AllArgsConstructor;
import lombok.Getter;

/**
```

```
 * @author adtimokhin
 * 16.10.2021
 **/
@AllArgsConstructor
public class BookingError {

    @Getter
    public String message;

}
```

UserError:

```
package com.adtimokhin.validation.errors;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;

/**
 * @author adtimokhin
 * 16.10.2021
 **/

@AllArgsConstructor
public class UserError {

    @Getter
    public String message;

}
```

BookingValidator:

```
package com.adtimokhin.validation;

import com.adtimokhin.model.Booking;
import com.adtimokhin.model.Machine;
import com.adtimokhin.model.Types;
import com.adtimokhin.model.User;
import com.adtimokhin.service.BookingService;
import com.adtimokhin.service.MachineService;
import com.adtimokhin.util.time.DateFormatResolver;
import com.adtimokhin.util.time.Timetable;
import com.adtimokhin.util.time.TimetableContainer;
import com.adtimokhin.validation.errors.BookingError;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;

import static com.adtimokhin.util.time.DateFormatResolver.DATE_FORMAT;
import static com.adtimokhin.util.time.Timetable.SLOT_SIZE;

/**
 * @author adtimokhin
 * 16.10.2021
 **/

@Component
public class BookingValidator {

    @Autowired
    private BookingService bookingService;

    @Autowired
    private MachineService machineService;

    @Autowired
    private TimetableContainer timeTableContainer;

    /**
     * Maximum number of minutes that one person can book washing machines for.
```

```
     */
    public final static int MAX_WASHING_MACHINE_WEEKLY_BOOKING_TIME = 5 * 60;

    /**
     * Maximum number of minutes that one person can book drying machines for.
     */
    public final static int MAX_WASHING_DRYING_WEEKLY_BOOKING_TIME = 6 * 60;

    private static final DateFormatResolver dateFormatResolver = new DateFormatResolver();

    /**
     * This method performs all appropriate checks in order to consider a {@link Booking} to be
valid.
     *
     * @param machineType      {@link String} that represents a type of {@link Machine}. This
should either be {@link Types#WASHING_MACHINE} or {@link Types#DRYING_MACHINE}
     * @param machineId        {@link Machine} id.
     * @param date             {@link String} in format {@value DateFormatResolver#DATE_FORMAT}
that would be used to set the {@link Booking}. It is used to set year, month and day for the
start and end of the booking.
     * @param periods          {@link List<String>} of ids for selected periods.
     * @param user             {@link User} that made a booking.
     * @return {@link List<BookingError>}. It is empty is no errors occurred.
     */
    public List<BookingError> validate(String machineType, String machineId, String date,
List<String> periods, User user) {
        List<BookingError> errors = new ArrayList<>();

        // Check that input can be parsed.
        int machineIdInt;
        try {
            machineIdInt = Integer.parseInt(machineId);
            Machine machine = machineService.findById(machineIdInt);
            if (machine == null){
                errors.add(new BookingError("This machine does not exist."));
                return errors;
            }
        } catch (Exception e) {
            errors.add(new BookingError("Could not resolve machine that you have selected."));
            return errors;
        }


        // Check that date is in correct format.
        if (date == null) {
            errors.add(new BookingError("Date is not provided."));
            return errors;
        } else if (date.equals("")) {
            errors.add(new BookingError("Date is not provided."));
            return errors;
        }

        if (!dateFormatResolver.appropriateFormat(date)) {
            errors.add(new BookingError("Please check that you leave time in format like this: "
+ DATE_FORMAT));
            return errors;
        }

        if (dateFormatResolver.isDateBefore(date, dateFormatResolver.today())) {
            errors.add(new BookingError("Booking is made for day that already has passed."));
            return errors;
        }

        // Check that appropriate machine type was chosen.
        boolean isWashingMachine = false;
        // Objective 5.3.1
        if (machineType.equals(Types.WASHING_MACHINE.toString())) {
            if (machineService.findById(machineIdInt) == null) { // Objective 5.3.2
                errors.add(new BookingError("No such machine is found."));
                return errors;
            }

            isWashingMachine = true;
        } else if (machineType.equals(Types.DRYING_MACHINE.toString())) {
            if (machineService.findById(machineIdInt) == null) { // Objective 5.3.2
                errors.add(new BookingError("No such machine is found."));
                return errors;
            }
        } else {
            errors.add(new BookingError("You have to choose a valid machine type."));
```

```
            return errors;
        }

        // check that there are periods selected
        if (periods == null) {
            errors.add(new BookingError("You have to select some time to make a booking."));
            return errors;
        }

        if (periods.size() == 0) {
            errors.add(new BookingError("You have to select some time to make a booking."));
            return errors;
        }

        // Objective 5.3.3 - 5.3.7 . As mentioned before, there is no need to validate the time
format because user must only tick boxes. The boxes correspond to certain time slots. Thus,
special validation is not required.
        // check if periods are in sequential order.
        try {
            int periodId = Integer.parseInt(periods.get(0).substring(3));
            for (int i = 1; i < periods.size(); i++) {
                int nextId = Integer.parseInt(periods.get(i).substring(3));
                if (nextId != (periodId + 1)) {
                    errors.add(new BookingError("You should select your bookings in sequential
order."));
                    return errors;
                }

                periodId = nextId;
            }
        } catch (Exception e) {
            errors.add(new BookingError("Could not resolve time slots selected."));
            return errors;

        }

        // Objective 5.3.8
        // check if spot is free
        int minId = Integer.parseInt(periods.get(0).substring(3));
        int maxId = Integer.parseInt(periods.get(periods.size() - 1).substring(3));

        int startHourInt = 6 + ((minId) / (60 / SLOT_SIZE));
        int startMinuteInt = (minId % (60 / SLOT_SIZE)) * SLOT_SIZE;
        int endHourInt = 6 + ((maxId + 1) / (60 / SLOT_SIZE));
        int endMinuteInt = ((maxId + 1) % (60 / SLOT_SIZE)) * SLOT_SIZE;


        Timetable timeTable = timeTableContainer.getTimeTable(machineIdInt, isWashingMachine);
        if (!timeTable.isFreeBetween(startHourInt, startMinuteInt, endHourInt, endMinuteInt)) {
            errors.add(new BookingError("This time slot is not available."));
            return errors;
        }

        // Objective 5.3.3, 5.3.5, 5.3.6
        // Check that booking is possible.
        if
(dateFormatResolver.isDateBeforeAnother(dateFormatResolver.resolveTimeForDate(String.valueOf(star
tHourInt), String.valueOf(startMinuteInt), date), dateFormatResolver.today())) {
            errors.add(new BookingError("You try to book for time that has already passed."));
            return errors;
        }

        // Objective 5.3.9
        // check what is the total booking time for this week and whether user can or can't book
for that long.
        long timeOfBooking = (long) periods.size() * SLOT_SIZE; // in minutes

        if (isWashingMachine) {
            List<Booking> bookings = bookingService.findAllForGivenWeek(user,
Types.WASHING_MACHINE, date);
            int weekBookTime = 0;
            if (bookings != null) {
                for (Booking booking :
                        bookings) {
                    weekBookTime +=
dateFormatResolver.getMinutesBetweenDates(booking.getEndDate(), booking.getStartDate());
                }
            }

            if (weekBookTime + timeOfBooking > MAX_WASHING_MACHINE_WEEKLY_BOOKING_TIME) {
```

```
                errors.add(new BookingError(String.format("You can book up to %d minutes of
washings. " +
                                "So, you can book only %d more minutes.",
MAX_WASHING_MACHINE_WEEKLY_BOOKING_TIME,
                        (MAX_WASHING_MACHINE_WEEKLY_BOOKING_TIME - weekBookTime))));
                return errors;
            }


        } else {
            List<Booking> bookings = bookingService.findAllForGivenWeek(user,
Types.DRYING_MACHINE, date);
            int weekBookTime = 0;
            if (bookings != null) {
                for (Booking booking :
                        bookings) {
                    weekBookTime +=
dateFormatResolver.getMinutesBetweenDates(booking.getEndDate(), booking.getStartDate());
                }
            }

            if (weekBookTime + timeOfBooking > MAX_WASHING_DRYING_WEEKLY_BOOKING_TIME) {
                errors.add(new BookingError(String.format("You can book up to %d minutes of
drying. " +
                                "So, you can book only %d more minutes.",
MAX_WASHING_DRYING_WEEKLY_BOOKING_TIME,
                        (MAX_WASHING_DRYING_WEEKLY_BOOKING_TIME - weekBookTime))));
                return errors;
            }
        }

        return null;
    }

}
```

UserValidator:

```
package com.adtimokhin.validation;

import com.adtimokhin.model.User;
import com.adtimokhin.service.UserService;
import com.adtimokhin.validation.errors.UserError;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * @author adtimokhin
 * 16.10.2021
 **/

@Component
public class UserValidator { // Objective 1.3

    private static final int MIN_PASSWORD_LENGTH = 7;

    @Autowired
    private UserService userService;

    /**
     * This method validates input for user to be registered on the platform.
     *
     * @param user        {@link User} that contains all the fields that require validation.
     * @param passwordTwo {@link String} that represents a check-password
     * @return {@link List<UserError>} or null. If null is returned, then no errors have been
detected.
     */
    public List<UserError> validate(User user, String passwordTwo) {
        ArrayList<UserError> errors = new ArrayList<>(); // we will add UserErrors to this List.

        //check one - password is at least 7 characters long
        if (user.getPassword().equals("")) {
```

```
            errors.add(new UserError("Your password field is empty."));
        } else {
            if (user.getPassword().length() < MIN_PASSWORD_LENGTH) { // Objective 1.3.4 -
password is at least 7 characters long
                errors.add(new UserError("Password should be at least " + MIN_PASSWORD_LENGTH + "
characters long."));
            } else if (!user.getPassword().equals(passwordTwo)) { // Objective 1.3.3 - passwords
match
                errors.add(new UserError("Passwords do not match."));
            }
        }

        //check two - email is in correct format. Objective 1.3.1
        if (!isEmailInCorrectFormat(user.getEmail())) {
            errors.add(new UserError("Email is in invalid form."));
        } else {
            // check three - email is not yet in use. Objective 1.3.2
            if (userService.findByEmail(user.getEmail()) != null) {
                errors.add(new UserError("This email is already in use."));
            }
        }

        // Objective 1.3.6
        if (user.getName().equals("")) {
            errors.add(new UserError("You should enter your name and surname."));
        } else if (user.getName().split(" ").length != 2) {
            errors.add(new UserError("You should enter your name and surname."));
        }

        return errors;
    }

    private static final String emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
    private static final Pattern emailPattern = Pattern.compile(emailRegex);

    /**
     * Method uses regular expressions to check if the input data is in a format of an email
(defined by {@link #emailRegex})
     *
     * @param email {@link String} to be validated.
     * @return true if the input data follows the {@link #emailRegex}
     */
    private boolean isEmailInCorrectFormat(String email) { // Objective 1.3.1
        Matcher matcher = emailPattern.matcher(email);
        return matcher.matches();

    }
}
```

email.properties:

```
mail.host=smtp.yandex.ru
mail.username=kolestnitsa-maria@yandex.ru
mail.password=
mail.port=465
mail.protocol=smtps
mail.smtp.auth=true
mail.smtp.ssl.enable=true
```

log4j.properties:

```
log4j.appender.trace=org.apache.log4j.RollingFileAppender
log4j.appender.trace.File=/Users/atimokhina/Desktop/washingDisasterSolver/log/trace.log
log4j.appender.trace.MaxFileSize=10MB
log4j.appender.trace.MaxBackupIndex=10
log4j.appender.trace.layout=org.apache.log4j.PatternLayout
log4j.appender.trace.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p [%l]: %m%n
log4j.category.com.adtimokhin= TRACE, trace
```

persistsence.properties:

```
user=adtimokhin
password=
url=jdbc:postgresql://localhost:5432/washing_queues_database
driver=org.postgresql.Driver
```

main.css:

```
/* Footer */
.footer {
    position: relative;
    bottom: 0;
    width: 100%;
    height: auto;
    line-height: 60px;
    background-color: #A4A69F;
}

.logo-container {
    margin: auto;
    max-width: 240px;
}

.logo-container img {
    bottom: 0;
}

.footer-link {
    display: block;
    text-align: center;
}

a.footer-link-title {
    display: block;
    text-align: center;
    text-decoration-line: none;
    color: #f2f2f4;
}

a:hover.footer-link-title {
    color: #F2F2F2;
}

a.footer-link {
    height: 40px;
    text-decoration-line: none;
    color: #585954;

}

a:hover.footer-link {
    color: #F2F2F2;
}


/*Text*/


h3 {
    color: #0D0D0D;
    text-decoration-line: underline;
    text-decoration-color: #0D0D0D;
    font-weight: 400;
    font-family: Roboto, Rockwell, 'Arial Black', sans-serif;
}

.section {
    margin-top: 1rem;
    margin-bottom: 9rem;
}

.paragraph {

    color: #585954;

    margin-top: 1rem;
```

```
    font-family: Rockwell, Georgia, 'Times New Roman', Times, serif;
    font-weight: bold;
    font-style: italic;
    font-size: 18px;
    line-height: 26pt;

}

.main-context {
    margin-top: 8rem;
    margin-bottom: 7rem;
}

.invisible-container {
    height: 14rem;
}

/*Button*/
.button {
    display: inline-block;
    padding: 0.5em 3em;
    border: 0.16em solid #A4A69F;
    margin: 0 0.3em 0.3em 0;
    box-sizing: border-box;
    text-decoration: none;
    text-transform: uppercase;
    font-family: 'Roboto', sans-serif;
    font-weight: 400;
    color: #A4A69F;
    text-align: center;
    transition: all 0.15s;
    align-content: center;
}

.button:hover {
      color: #585954;
    border-color: #585954;
}

.button:active {
    color: #F2F2F2;
    border-color: #F2F2F2;

}

@media all and (max-width: 30em) {
      a.button {
        display: block;
        margin: 0.4em auto;
    }
}

/*form-container*/
.form-container {
    min-height: 5rem;
    align-items: center;
    justify-content: center;
    margin: 5rem 0 5rem 0;
    padding: 5rem 0 5rem 0;

    background: #585954;
    box-shadow: 60px -16px #0D0D0D;
}

div.input-holder {
    align-items: center;
    justify-content: center;
    margin: 0 0.3em 0.3em 0;
    padding-top: 2rem;
    padding-bottom: 4rem;
}

div.input-holder input {
    padding: 0.5em 3em;
    border: 0.16em solid #A4A69F;
    box-sizing: border-box;
    margin: auto;
    font-family: 'Roboto', sans-serif;
    color: #585954;
```

```
    background-color: #A4A69F;
    -webkit-transition: 0.5s;
    transition: 0.5s;
    outline: none;

}

div.input-holder input:hover {
    border: 0.16em solid #585954;

}

div.input-holder input:focus {
    border: 0.16em solid #F2F2F2;

}


div.input-holder p {
    margin: auto;
    text-align: center;
    color: #F2F2F2;
    font-family: 'Roboto', sans-serif;
    font-weight: 400;
}

.form-title {
    margin: auto;
    text-align: center;
    color: #F2F2F2;
    font-family: 'Roboto', sans-serif;
    font-weight: bolder;
}

::-moz-selection { /* Code for Firefox */
    color: #585954;
    background: #F2F2F2;
}

::selection {
    color: #585954;
    background: #F2F2F2;
}

.error-msg {
    text-align: center;
    color: #585954;
    font-family: 'Roboto', sans-serif;
    font-weight: bolder;
    margin: auto auto 4rem;
}


div.input-holder input[type=radio] {
    padding: 0.5em 3em;
    border: 0.16em solid #A4A69F;
    box-sizing: border-box;
    margin: auto;
    font-family: 'Roboto', sans-serif;
    color: #585954;
    background-color: #A4A69F;
    -webkit-transition: 0.5s;
    transition: 0.5s;
    outline: none;
    cursor: pointer;
    height: 2rem;
    width: 2rem;

}


.table-container {
    margin: 2rem 4rem 5rem 4rem;
}


.table-container table {
    color: #0D0D0D;
    border-collapse: collapse;
    border: 1px solid #585954;
```

```
    background-color: #A4A69F;
    width: 100%;
}

.table-container tr, th {
    padding: 1rem 2rem 1rem 2rem;
    border-width: 3px;
    text-align: center;
    font-family: Rockwell, Georgia, 'Times New Roman', Times, serif;
    font-weight: bold;
    font-style: italic;
    font-size: 18px;

}
```

account.ftl:

```
<#--Objecive 6-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Account</title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body  style="background: rgb(214, 217, 208);">


<div class="container main-context">

    <div class="row">
        <div class="container">
            <div class="row">
                <div class="col-lg-4 col-0"></div>
                <div class="col-lg-4 col-12">
                    <a href="/account/" class="button mx-auto d-block" style="">Return</a>
                </div>
                <div class="col-lg-4 col-0"></div>
            </div>
        </div>
    </div>

    <#if msg??>
        <div class="row">
            <div class="col-12">
                <h4 class="error-msg">${msg}</h4>
            </div>
        </div>
    </#if>

    <div class="row">
        <div class="col-1"></div>
        <div class="col-10">
            <div class="form-container">
                <div class="input-holder">
                    <h4 class="form-title">My Current Bookings:</h4>
                </div>
                <div class="input-holder">
                    <h3 style="color: #F2F2F2; text-decoration-color: #F2F2F2;">Washing Machine
Bookings:</h3>
                </div>
                <#if washings??>
                    <#list washings as w>
                        <div class="input-holder">
                            <p>${w.presentNicely()}</p>
                            <form method="post" action="/booking/cancel/washing_machines">
                                <input type="hidden" name="id" value="${w.getId()}">
                                <#--                    Objective 13.1-->
```

```
                                <input type="hidden" name="${_csrf.parameterName}"
value="${_csrf.token}">
<#--                                Objective 7.1 -->
                                <input type="submit" value="Cancel" class="mx-auto d-block
button">
                            </form>
                        </div>
                    </#list>
                <#else>
                    <div class="input-holder">
                        <p>You have to bookings for washing machines</p>
                    </div>
                </#if>

                <div class="input-holder">
                    <h3 style="color: #F2F2F2; text-decoration-color: #F2F2F2;">Drying Machine
Bookings:</h3>
                </div>
                <#if dryings??>
                    <#list dryings as d>
                        <div class="input-holder">
                            <p>${d.presentNicely()}</p>
                            <form method="post" action="/booking/cancel/drying_machines">
                                <input type="hidden" name="id" value="${d.getId()}">
                                <#--                        Objective 13.1-->
                                <input type="hidden" name="${_csrf.parameterName}"
value="${_csrf.token}">
                                <#--                                    Objective 7.1 -->
                                <input type="submit" value="Cancel" class="mx-auto d-block
button">
                            </form>
                        </div>
                    </#list>
                <#else>
                    <div class="input-holder">
                        <p>You have to bookings for drying machines</p>
                    </div>
                </#if>
            </div>
        </div>
    </div>
    <div class="col-1"></div>
</div>


<#--Objective 9.1-->
<form method="post" action="/logout">
    <#--                        Objective 13.1-->
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
    <input type="submit" value="Logout" class="mx-auto d-block button">
</form>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
```

```
    </div>
</footer>
</body>
</html>
```

info.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Actions Page</title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg"/>
</head>
<body style="background: rgb(214, 217, 208);">

<#--Main container-->
<div class="container main-context">
    <div class="row">
        <div class="container">
            <div class="row">
                <div class="col-lg-4 col-0"></div>
                <div class="col-lg-4 col-12">
                    <a href="/booking/actions" class="button mx-auto d-block" style="">Return</a>
                </div>
                <div class="col-lg-4 col-0"></div>
            </div>
        </div>
    </div>
    <div class="row">
        <div class="col-12">
            <h4 class="error-msg">Logged As: ${name}</h4>
        </div>

        <div class="row">
            <div class="container">
                <div class="row">
                    <div class="col-md-4 col-xs-3"></div>
                    <div class="col-md-4 col-xs-6">
                        <a href="/account/bookings" class="button mx-auto d-block" style="">View
My Bookings</a>
                    </div>
                    <div class="col-md-4 col-xs-3"></div>
                </div>
            </div>
            </div>
            <form method="post" action="/logout">
                <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
                <input type="submit" value="Logout" class="mx-auto d-block button">
            </form>
        </div>
        <div class="row">
            <div class="container invisible-container">
                <div class="col-12"></div>
            </div>
        </div>
    </div>

    <!-- Footer -->
    <footer class="footer">
        <div class="container">
            <div class="row">
                <!-- photo -->
                <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                    <div class="logo-container">
                        <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-
auto d-block">
                    </div>
```

```
                </div>
                <!-- Other action buttons -->
                <div class="col-md-4 col-sm-2 col-xs-6">
                    <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                    <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                    <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                       class="footer-link">LinkedIn</a>
                    <a href="https://www.instagram.com/adtimokhin/" target="_blank"
class="footer-link">Instagram</a>
                </div>

                <div class="col-md-4 col-sm-3 col-xs-6">
                    <a href="#" class="footer-link-title">Terms & policy</a>
                </div>
            </div>
        </div>
    </footer>

</body>
</html>
```

actions.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Actions Page</title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">

<#--Main container-->
<div class="container main-context">
    <div class="row">
        <div class="container">
            <div class="row">
                <div class="col-md-4 col-xs-3"></div>
                <div class="col-md-4 col-xs-6">
                    <a href="/booking/view/WASHING_MACHINE" class="button mx-auto d-block"
style="">Book A Washing Machine</a>
                </div>
                <div class="col-md-4 col-xs-3"></div>
            </div>
        </div>
        <div class="container">
            <div class="row">
                <div class="col-md-4 col-xs-3"></div>
                <div class="col-md-4 col-xs-6">
                    <a href="/booking/view/DRYING_MACHINE" class="button mx-auto d-block"
style="">Book A Drying Machine</a>
                </div>
                <div class="col-md-4 col-xs-3"></div>
            </div>
        </div>
        <div class="container">
            <div class="row">
                <div class="col-md-4 col-xs-3"></div>
                <div class="col-md-4 col-xs-6">
                    <a href="/account/" class="button mx-auto d-block" style="">My Account</a>
                </div>
                <div class="col-md-4 col-xs-3"></div>
            </div>
        </div>
        <div class="container">
            <div class="row">
```

```
                <div class="col-md-4 col-xs-3"></div>
                <div class="col-md-4 col-xs-6">
<#--                    Objective 8.1-->
                    <a href="/complaint/" class="button mx-auto d-block" style="">My Machine Is
Occupied</a>
                </div>
                <div class="col-md-4 col-xs-3"></div>
            </div>
        </div>
    </div>
    <div class="row">
        <div class="container invisible-container">
            <div class="col-12"></div>
        </div>
    </div>
</div>

<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>

</body>
</html>
```

dateSelect.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Select Date</title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg"/>
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">
    <#if error??>
        <div class="row">
            <div class="col-12">
                <h4 class="error-msg">${error}</h4>
            </div>
```

```
        </div>
    </#if>
    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <form method="get" action="/booking/view/${machineType}">
                    <div class="input-holder">
                        <h4 class="form-title">Check machine availability</h4>
                    </div>
                    <div class="input-holder">
                        <#--                        Objective 4.2-->
                        <#--                        Objective 4.3 This objectives is achieved by
a use of the "date" input type. It automatically converts date into the format YYYY MM dd HH:mm--
>
                        <p class="mx-auto d-block">Select date to view<br/></p>
                        <input type="date" name="date" class="mx-auto d-block"
value="${currentDate}"
                               min="${minDate}" max="${maxDate}">
                    </div>
                    <div>
                        <div class="popup">
                            <input type="submit" value="View machines" class="mx-auto d-block
button">
                        </div>
                    </div>
                </form>
            </div>
        </div>
    </div>
    <div class="col-2"></div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto d-
block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

preview.ftl:

```
<#--Objective 4.1-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver
        | <#if machineType == "washing_machines">Washing Machines<#else >Drying
Machines</#if></title>
    <link href="../../resources/css/main.css" rel="stylesheet">
```

```
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>
    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg"/>

</head>
<body style="background: rgb(214, 217, 208);">

<div class="container main-context">

    <div class="row">
        <div class="container">
            <div class="row">
                <div class="col-lg-4 col-0"></div>
                <div class="col-lg-4 col-12">
                    <a href="/booking/actions" class="button mx-auto d-block" style="">Return</a>
                </div>
                <div class="col-lg-4 col-0"></div>
            </div>
        </div>
    </div>


    <#if error??>
        <div class="row">
            <div class="col-12">
                <h4 class="error-msg"> Error message: ${error}</h4>
            </div>
        </div>
    </#if>

    <#if msg??>
        <div class="row">
            <div class="col-12">
                <h4 class="error-msg">Message: ${msg}</h4>
            </div>
        </div>
    </#if>

    <div class="row">
        <div class="col-12">
            <h4 class="error-msg">This are the bookings for ${representativeDate}:</h4>
        </div>
    </div>

    <#--    Objectives 4.1, 5.1, 5.2.(1-4) -->
    <#--    As explained in the design section, most of the objectives from the 5.2.(1-4) list a
achieved internally, without a need from the user to enter any information mannually.-->
    <#if timeTables??>
        <div class="row">
            <div class="col-2"></div>
            <div class="col-8">
                <#list timeTables as timeTable>
                    <div class="table-container">
                        <h3><#if machineType == "WASHING_MACHINE">Washing Machine<#else >Drying
Machine</#if>
                            (id: ${timeTable.getMachine().getId()})</h3>
                        <form method="post" action="/booking/add/${machineType}">
                            <table>
                                <tr>
                                    <#list timeTable.timePeriods as periods>
                                        <th>${periods.getTimeBounds()}</th>
                                    </#list>
                                </tr>
                                <tr>
                                    <#list timeTable.timePeriods as periods>
                                        <#if periods.isIS_FREE()>
                                            <th style="background-color: #F2F2F2">
                                                Vacant
                                                <input type="checkbox" name="periods"
value="id_${periods.getID()}">
                                            </th> <#else>
                                            <th>Occupied
                                                <input type="checkbox" name="periods"
value="id_${periods.getID()}"
                                                       disabled
```

```
                                                    hidden></th></#if>
                                    </#list>
                                </tr>
                            </table>

                            <input type="hidden" name="machineId"
value="${timeTable.getMachine().getId()}">
                            <input type="hidden" name="date" value="${date}">
                            <#--                        Objective 13.1-->
                            <input type="hidden" name="${_csrf.parameterName}"
value="${_csrf.token}">
                            <div class="input-holder">
                                <input type="submit" value="Make a reservation" class="mx-auto d-
block button">
                            </div>

                        </form>
                    </div>
                </#list>
            </div>
            <div class="col-2"></div>
        </div>
    </#if>

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <form method="get" action="/booking/choose/date/${machineType}">
                    <div class="input-holder">
                        <h4 class="form-title">SELECT DATE TO VIEW </h4>
                    </div>
                    <input type="hidden" name="date" value="${date}" class="mx-auto d-block">
                    <input type="hidden" name="machineType" value="${machineType}" class="mx-auto
d-block">
                    <div class="input-holder">
                        <input type="submit" class="mx-auto d-block button" value="Look up
machines for this date">
                    </div>
                    <#--                    <input type="hidden" name="${_csrf.parameterName}"
value="${_csrf.token}">-->
                </form>
            </div>
        </div>
        <div class="col-2"></div>
    </div>
</div>

<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto d-
block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

complaint.ftl:

```
<#--Objective 8.1-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Reports Page</title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">
<div class="container main-context">

    <div class="row">
        <div class="container">
            <div class="row">
                <div class="col-lg-4 col-0"></div>
                <div class="col-lg-4 col-12">
                    <a href="/booking/actions" class="button mx-auto d-block" style="">Return</a>
                </div>
                <div class="col-lg-4 col-0"></div>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <form method="post" action="/complaint/add">
                    <div class="input-holder">
                        <h4 class="form-title">What machine is occupied?</h4>
                    </div>
                    <div class="input-holder">
                        <p class="mx-auto d-block">Machine Id<br/></p>
<#--                        Objective 8.2.1-->
                        <input type="text" name="id" class="mx-auto d-block">
                    </div>
                    <div class="input-holder">
                        <div class="popup">
                            <input type="submit" value="Submit" class="mx-auto d-block button">
                        </div>
                    </div>
                    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
                </form>
            </div>
        </div>
    </div>
    <div class="col-2"></div>
</div>
<div class="container main-context">
    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">

            <div class="container">
                <div class="row">
                    <div class="col-12">
                        <h4 class="error-msg">Current Complaints:</h4>
<#--                        Objective 8.5-->
                    </div>
                </div>

                <div class="row">
                    <div class="col-12 paragraph section">
                        <#if washingMachineComplaints??>
                            <p>Complaints about washing machine users: </p>
                            <ul class="list-unstyled">
```

```
                                    <ul>
                                        <#list washingMachineComplaints as compalint>
                                            <li>
                                                Booking for washing
                                                machine
${compalint.getBooking().getWashingMachine().getId()} that finishes
                                                at ${compalint.getBooking().getEndDate()} by
                                                user ${compalint.getBooking().getUser().getName()}
                                            </li>
                                        </#list>
                                    </ul>
                                </ul>
                            <#else>
                                <p>No complaints about usage of washing machines</p>
                            </#if>
                        </div>
                    </div>

                    <div class="row">
                        <div class="col-12 paragraph section">
                            <#if dryingMachineComplaints??>
                                <p>Complaints about washing machine users: </p>
                                <ul class="list-unstyled">
                                    <ul>
                                        <#list dryingMachineComplaints as compalint>
                                            <li>
                                                Booking for washing machine
${compalint.getBooking().getDryingMachine().getId()}
                                                that finishes
                                                at ${compalint.getBooking().getEndDate()} by
                                                user ${compalint.getBooking().getUser().getName()}
                                            </li>
                                        </#list>
                                    </ul>
                                </ul>
                            <#else>
                                <p>No complaints about usage of drying machines</p>
                            </#if>
                        </div>
                    </div>

                </div>

            </div>
        </div>
        <div class="col-2"></div>
</div>
<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

sucessfulVerification.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | 404! </title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <div class="input-holder">
                    <h4 class="form-title">You have successfully verified your email!</h4>
                </div>
            </div>
        </div>
        <div class="col-2"></div>
    </div>
    <div class="container">
        <div class="row">
            <div class="col-lg-4 col-0"></div>
            <div class="col-lg-4 col-12">
                <a href="/login" class="button mx-auto d-block" style="">Login</a>
            </div>
            <div class="col-lg-4 col-0"></div>
        </div>
    </div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
```

```
</html>
```

unsucessfulVerification.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | 404! </title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <div class="input-holder">
                    <h4 class="form-title">Your attempt to validate your email was unsuccessful
:(</h4>
                </div>
            </div>
        </div>
        <div class="col-2"></div>
    </div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

verificationEmailSentConfirmation:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | 404! </title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <div class="input-holder">
                    <h4 class="form-title">We have sent you an email with a verification code.
Check it out ;)</h4>
                </div>
            </div>
        </div>
        <div class="col-2"></div>
    </div>
    <div class="container">
        <div class="row">
            <div class="col-lg-4 col-0"></div>
            <div class="col-lg-4 col-12">
                <a href="/login" class="button mx-auto d-block" style="">Login</a>
            </div>
            <div class="col-lg-4 col-0"></div>
        </div>
    </div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

400.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | 404! </title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <div class="input-holder">
                    <h4 class="form-title">It is 404! We didn't find the page you've tried to
access :(</h4>
                </div>
            </div>
        </div>
    </div>
    <div class="col-2"></div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

500.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>Washing Disaster Solver | 404! </title>
    <link href="../../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">

    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <div class="input-holder">
                    <h4 class="form-title">It is Server Error! Argh... Can't do much about it
now. You can try reporting
                        it, but we have recorded the cause of this error. We will try to fix it
asap.</h4>
                </div>
            </div>
        </div>
    </div>
    <div class="col-2"></div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto
d-block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

index.ftl:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Home Page</title>
    <link href="../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
```

```
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

        <link rel="icon" type="image/x-icon" href="../../resources/images/SVG/WDS_LOGO.svg" />

</head>
<body style="background: rgb(214, 217, 208);">

<!-- Navigation bar -->
<div class="container">
    <div class="row">
        <nav class="navbar navbar-light justify-content-center"
             style="background-color: #A4A69F;">
            <img class="navbar-brand" src="../../resources/images/washingDisasterSolver.svg"/>
        </nav>
    </div>
</div>

<#--Main container-->
<div class="container main-context">
    <div class="row section">
        <h3>What is it?</h3>
        <div class="container">
            <div class="row">
                <div class="col-sm-1 col-xs-0">
                </div>
                <div class="col-sm-10 col-xs-12">
                    <p class="paragraph">
                        Lorem ipsum dolor sit amet consectetur, adipisicing elit. Minima
repudiandae quos velit animi.
                        Labore ipsum facere iusto commodi doloribus, temporibus nihil quo est vel
sint voluptates quas
                        perspiciatis odio, veniam iure magnam voluptas provident rem atque eaque
cum possimus suscipit?
                        Iure et fugit quo, quasi voluptatum hic maxime totam eius dicta quae
nostrum natus id dolores
                        mollitia doloribus quaerat, consectetur praesentium ratione adipisci
voluptas a illum
                        cupiditate. Nihil, nulla ullam delectus id inventore ratione doloribus
sint deleniti enim, ex
                        molestiae blanditiis tempora placeat similique explicabo, quo repellat
earum? Incidunt harum
                        tempore, minus, est eius nostrum sapiente voluptatum a et, perspiciatis
officia vel aut! Numquam
                        eum, enim molestias officia esse quod nobis facilis accusamus
exercitationem laborum omnis
                        voluptate dicta neque voluptates veniam velit? Voluptatem voluptatum id,
tempora atque ex, hic
                        totam animi, in dolores libero recusandae? Ab minus quia eos eveniet?
Dolores explicabo tenetur
                        fugit aspernatur, eos, voluptate sit expedita mollitia alias, ipsa quos.
Aperiam saepe
                        dignissimos vel id libero deserunt officiis fugit. Distinctio culpa
corrupti, quisquam tempora
                        illo magni, optio mollitia exercitationem in nihil possimus neque fugiat
impedit saepe. Ut est
                        ipsa rem blanditiis eligendi perspiciatis, enim nesciunt, quae debitis
perferendis ratione
                        veniam dolorem non quas iusto sint distinctio excepturi?

                    </p>
                </div>
                <div class="col-sm-1 col-xs-0"></div>
            </div>
        </div>
    </div>

    <div class="row section">
        <h3>How to participate?</h3>
        <div class="container">
            <div class="row">
                <div class="col-sm-1 col-xs-0"></div>
                <div class="col-sm-10 col-xs-12">
                    <p class="paragraph">
                        Lorem ipsum dolor sit amet consectetur, adipisicing elit. Minima
repudiandae quos velit animi.
                        Labore ipsum facere iusto commodi doloribus, temporibus nihil quo est vel
sint voluptates quas
```

```
                        perspiciatis odio, veniam iure magnam voluptas provident rem atque eaque
cum possimus suscipit?
                        Iure et fugit quo, quasi voluptatum hic maxime totam eius dicta quae
nostrum natus id dolores
                        mollitia doloribus quaerat, consectetur praesentium ratione adipisci
voluptas a illum
                        cupiditate. Nihil, nulla ullam delectus id inventore ratione doloribus
sint deleniti enim, ex
                        molestiae blanditiis tempora placeat similique explicabo, quo repellat
earum? Incidunt harum
                        tempore, minus, est eius nostrum sapiente voluptatum a et, perspiciatis
officia vel aut! Numquam
                        eum, enim molestias officia esse quod nobis facilis accusamus
exercitationem laborum omnis
                        voluptate dicta neque voluptates veniam velit? Voluptatem voluptatum id,
tempora atque ex, hic
                        totam animi, in dolores libero recusandae? Ab minus quia eos eveniet?
Dolores explicabo tenetur
                        fugit aspernatur, eos, voluptate sit expedita mollitia alias, ipsa quos.
Aperiam saepe
                        dignissimos vel id libero deserunt officiis fugit. Distinctio culpa
corrupti, quisquam tempora
                        illo magni, optio mollitia exercitationem in nihil possimus neque fugiat
impedit saepe. Ut est
                        ipsa rem blanditiis eligendi perspiciatis, enim nesciunt, quae debitis
perferendis ratione
                        veniam dolorem non quas iusto sint distinctio excepturi?

                    </p>
                </div>
                <div class="col-sm-1 col-xs-0"></div>
            </div>
        </div>
    </div>

    <div class="row">
        <div class="container">
            <div class="row">
                <div class="col-lg-4 col-0"></div>
                <div class="col-lg-4 col-12">
                    <a href="/login" class="button mx-auto d-block" style="">Login</a>
                </div>
                <div class="col-lg-4 col-0"></div>
            </div>
        </div>
        <div class="container">
            <div class="row">
                <div class="col-lg-4 col-0"></div>
                <div class="col-lg-4 col-12">
                    <a href="/sign_up" class="button mx-auto d-block" style="">Sign up</a>
                </div>
                <div class="col-lg-4 col-0"></div>
            </div>
        </div>
    </div>
</div>

<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto d-
block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                    class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>
```

```
            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>

</body>
</html>
```

login.ftl:

```
<#--Objective 3.1-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Login Page</title>
    <link href="../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>
    <link rel="icon" type="image/x-icon" href="../../resources/images/SVG/WDS_LOGO.svg" />
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">
    <#if error??>
        <div class="row">
            <div class="col-12">
                <h4 class="error-msg">${error}</h4>
            </div>
        </div>
    </#if>
    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <#--Objective 3.2-->
                <form method="post" action="/login/process">
                    <div class="input-holder">
                        <h4 class="form-title">LOGIN</h4>
                    </div>
                    <div class="input-holder">
                        <#--Objective 3.2.1-->
                        <p class="mx-auto d-block">Email<br/></p>
                        <input type="text" name="email" class="mx-auto d-block">
                    </div>
                    <div class="input-holder">
                        <#--Objective 3.2.2-->
                        <p class="mx-auto d-block">Password</p>
                        <input type="password" name="password" class="mx-auto d-block">
                    </div>
                    <div class="input-holder">
                        <div class="popup">
                            <input type="submit" value="Login" class="mx-auto d-block button">
                        </div>
                    </div>
<#--                    Objective 13.1-->
                    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
                </form>
            </div>
        </div>
    </div>
    <div class="col-2"></div>
</div>


<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
```

```
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto d-
block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>
</body>
</html>
```

signUp.ftl:

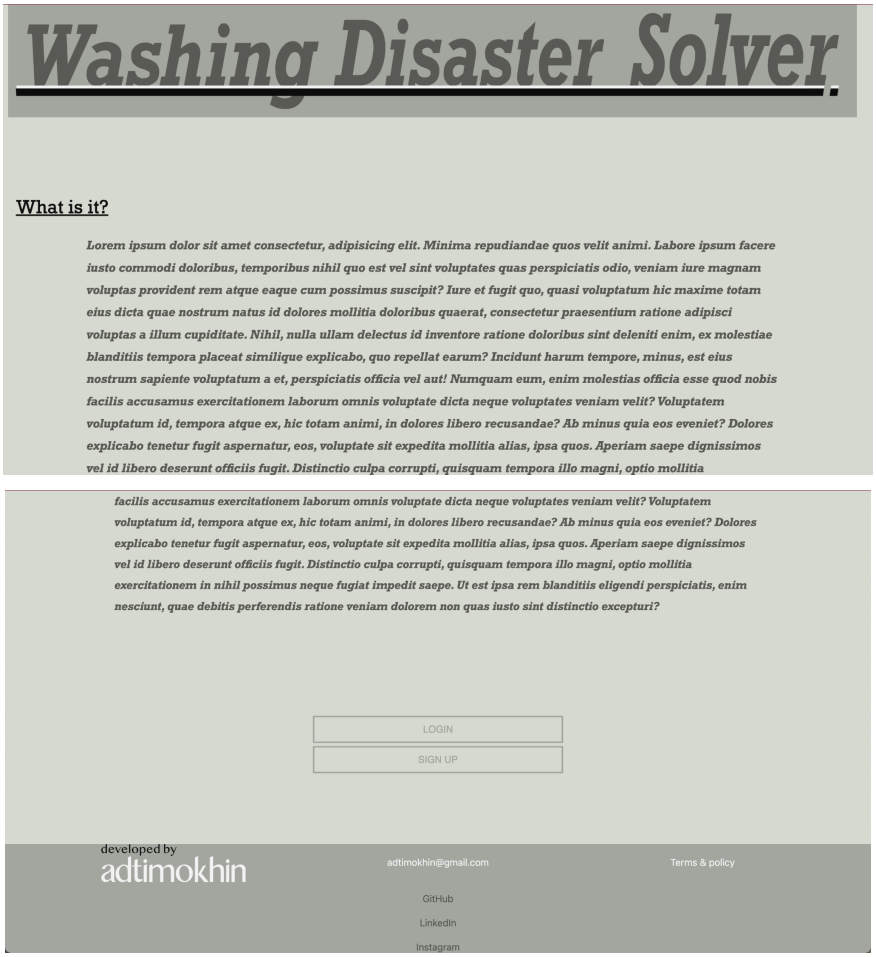
```
<#--Objective 1.1-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Washing Disaster Solver | Sign Up Page</title>
    <link href="../../resources/css/main.css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
          integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
            crossorigin="anonymous"></script>

    <link rel="icon" type="image/x-icon" href="../../resources/images/SVG/WDS_LOGO.svg"/>
</head>
<body style="background: rgb(214, 217, 208);">


<div class="container main-context">
    <#if errors??>
        <div class="row">
            <div class="col-12">
                <#list errors as error>
                    <h4 class="error-msg">${error.getMessage()}</h4>
                </#list>
            </div>
        </div>
    </#if>
    <div class="row">
        <div class="col-2"></div>
        <div class="col-8">
            <div class="form-container">
                <#--                Objective 1.2.-->
                <form method="post" action="/sign_up">
                    <div class="input-holder">
                        <h4 class="form-title">SIGN UP</h4>
                    </div>
                    <div class="input-holder">
                        <#--                            Objective 1.2.2-->
                        <p class="mx-auto d-block">Enter your email address<br/></p>
                        <input type="text" name="email" class="mx-auto d-block">
                    </div>
                    <div class="input-holder">
                        <#--                            Objective 1.2.1-->
                        <p class="mx-auto d-block">Enter your name</p>
```

```
                    <input type="text" name="name" class="mx-auto d-block">
                </div>
                <div class="input-holder">
                    <#--                            Objective 1.2.3-->
                    <p class="mx-auto d-block">Enter your password</p>
                    <input type="password" name="password" class="mx-auto d-block">
                </div>
                <div class="input-holder">
                    <#--                            Objective 1.2.4-->
                    <p class="mx-auto d-block">Confirm your password</p>
                    <input type="password" name="passwordTwo" class="mx-auto d-block">
                </div>
                <div class="input-holder">
                    <div class="popup">
                        <input type="submit" value="Register" class="mx-auto d-block button">
                    </div>
                </div>
                <#--                    Objective 13.1-->
                <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
            </form>
        </div>
    </div>
</div>
<div class="col-2"></div>
</div>

<!-- Footer -->
<footer class="footer">
    <div class="container">
        <div class="row">
            <!-- photo -->
            <div class="col-md-4 col-sm-5 col-xs-12" id="logo-container">
                <div class="logo-container">
                    <img src="../../resources/images/adtimokhinLOGOfooter.svg" class="mx-auto d-
block">
                </div>
            </div>
            <!-- Other action buttons -->
            <div class="col-md-4 col-sm-2 col-xs-6">
                <a href="#" class="footer-link-title">adtimokhin@gmail.com</a>
                <a href="https://github.com/adtimokhin/" target="_blank" class="footer-
link">GitHub</a>
                <a href="https://www.linkedin.com/in/aleksandr-timokhin-5300361b6"
target="_blank"
                   class="footer-link">LinkedIn</a>
                <a href="https://www.instagram.com/adtimokhin/" target="_blank" class="footer-
link">Instagram</a>
            </div>

            <div class="col-md-4 col-sm-3 col-xs-6">
                <a href="#" class="footer-link-title">Terms & policy</a>
            </div>
        </div>
    </div>
</footer>

</body>
</html>
```

FINAL PRODUCT

In the design and testing sections of my coursework I am referring to different pages of the website. Here is a summary of how my final product looks.

Index page

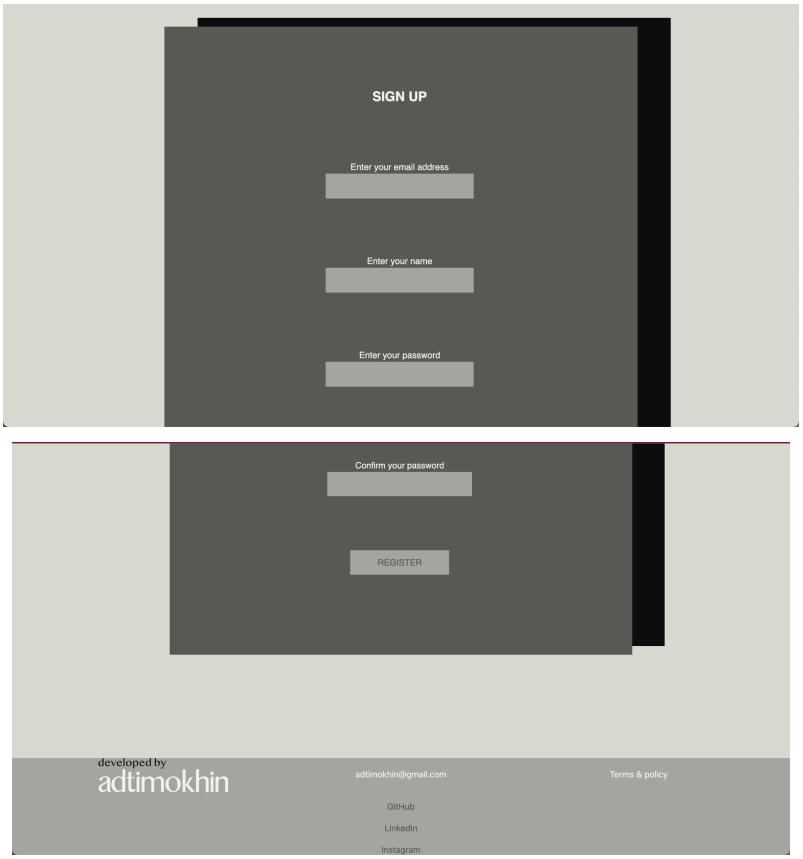


URL: http://localhost:8080/

This is my landing page. It should contain a quick description of the project's purpose. Currently this text is replaced with the "lorem ipsum".

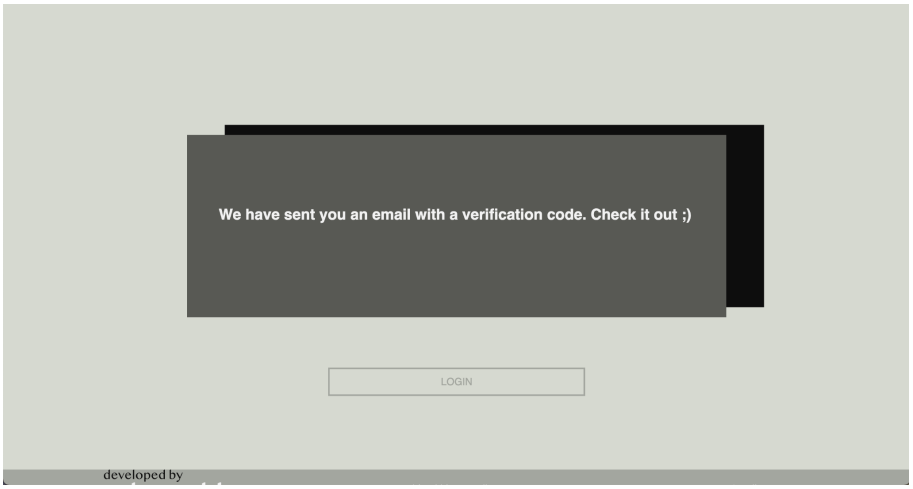
The page also contains two navigation buttons. One ("login") would take a user to the login page and the other ("sign up") would take them to a registration page.

Registration page



URL: http://localhost:8080/sign_up

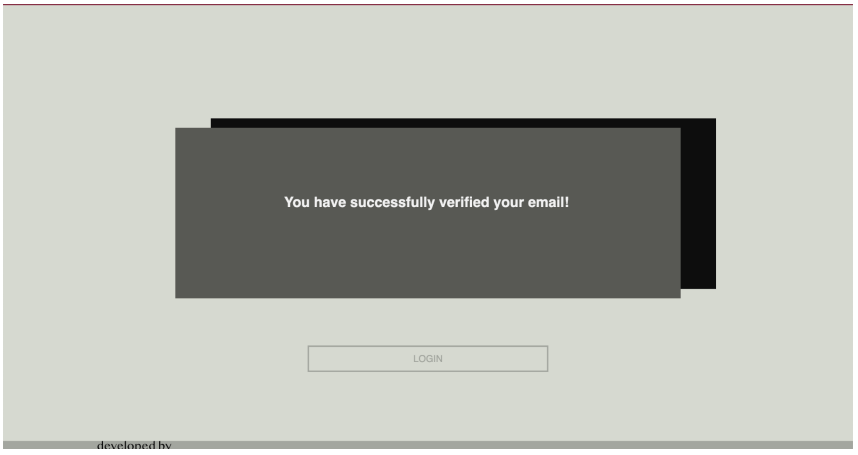
This page allows users to create an account. After successful validation of their input, users would receive an email with a link to verify their email. The users would be redirected to a page that would tell that they have successfully created an account and now they must validate their email. It looks like this:



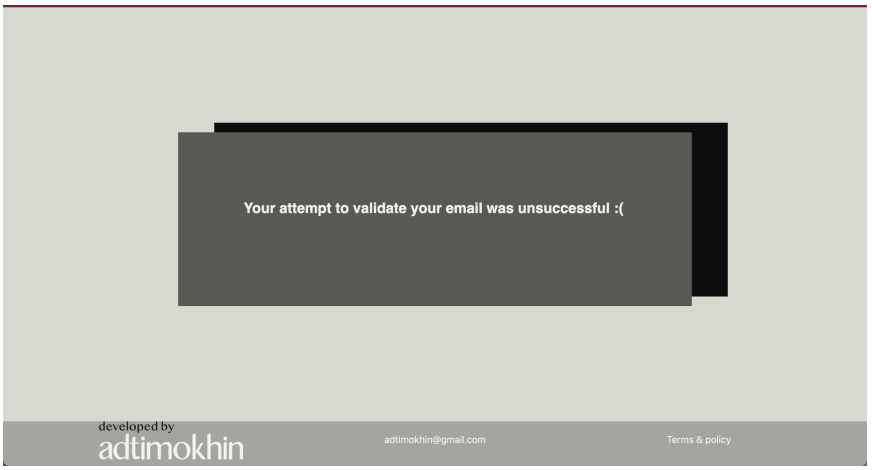
If the input is not entered correctly, the user would be redirected to this page which would contain error messages.

Email Verification page

Upon following the link in the email sent (URL: localhost:8080/verify/{token}) The user would see one of the two pages:

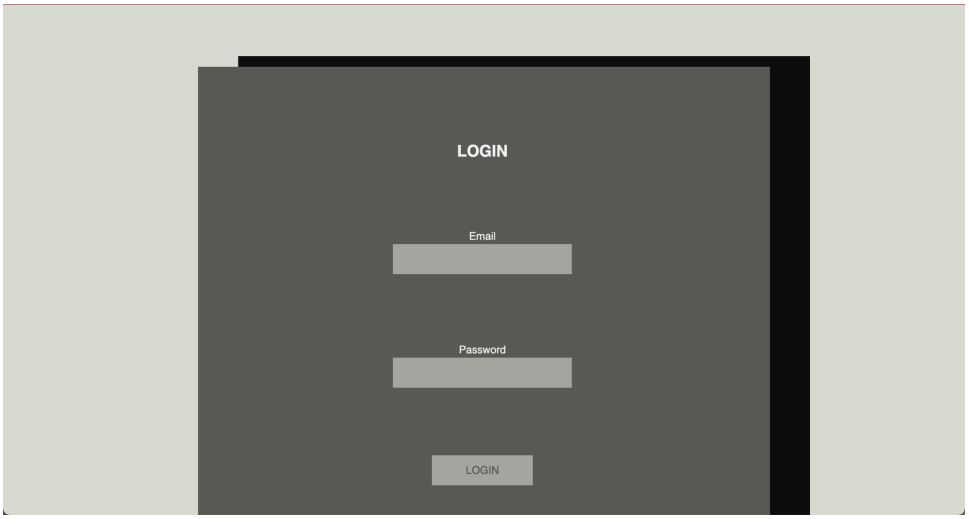


If email was verified



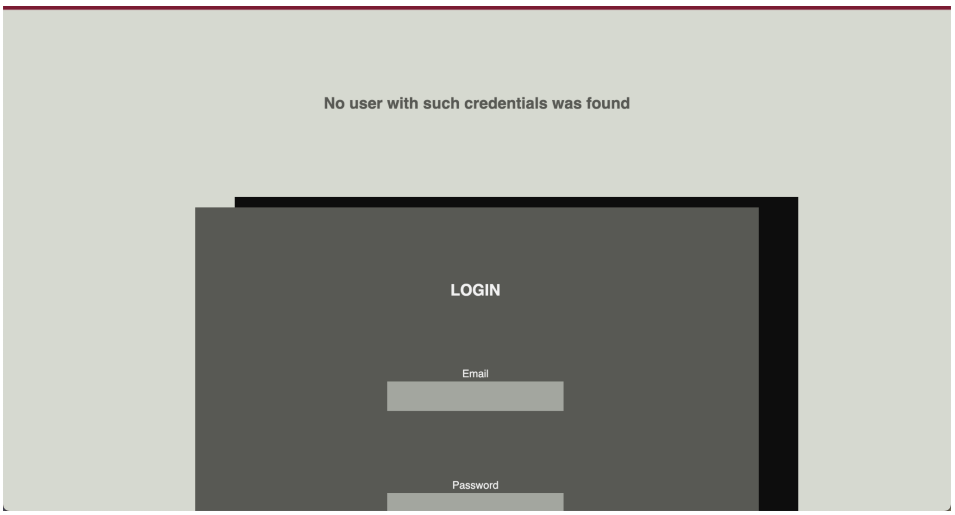
If email was not verified

Login page



URL: http://localhost:8080/login

This is a page where registered users can login into the platform. They need to provide their email and password. If the input is validated (that also refers to the fact that email was verified), then the user is assigned an authentication token and then redirected to the home page (**URL: http://localhost:8080/booking/actions**). However, if some of the validations fail, the user would be redirected to **http://localhost:8080/login?error=true** where a user would see on the top of the page a message that would tell them what validations have failed. It might look like this:



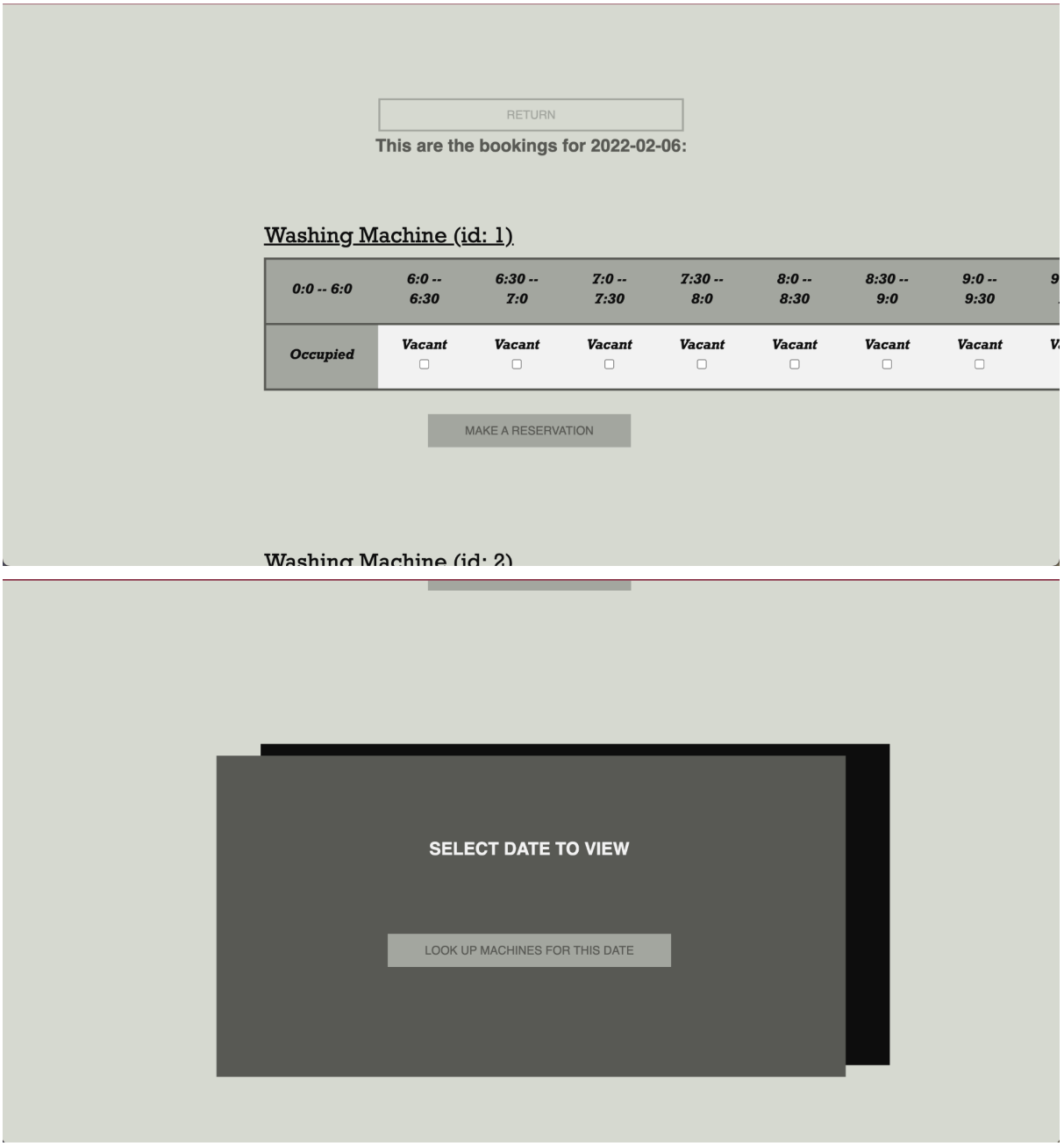
Home page

URL: http://localhost:8080/booking/actions

This is a navigation page that gives a list of all pages that an authenticated user can access. That includes: booking washing machine; booking drying machine; view information about the account; complain. Corresponding buttons are present on the home page.

Book Washing/Drying Machine page

Depending on the type specified in the URL, either timetables for washing or drying machines would be displayed.



An example page for washing machine timetables

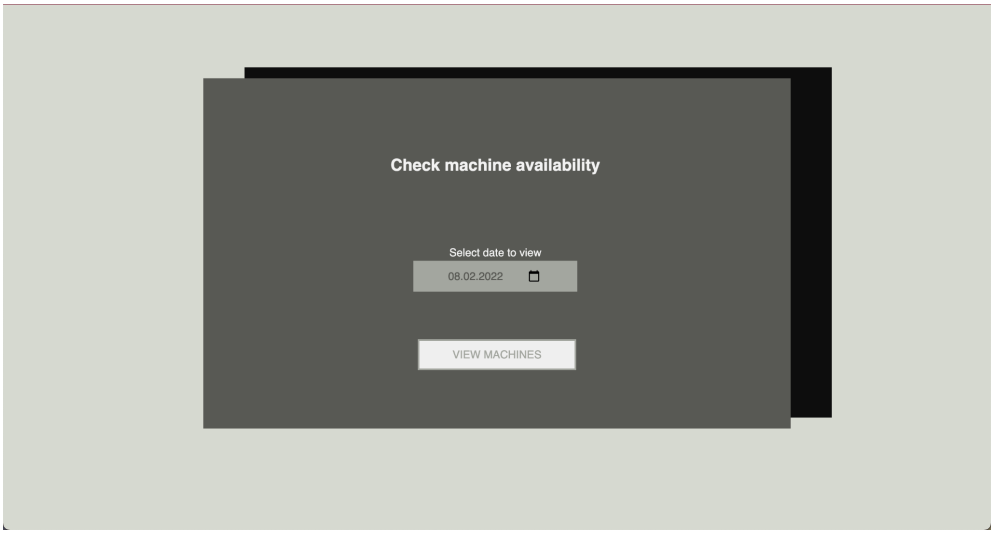
URL: http://localhost:8080/booking/view/washing_machines

This page also contains a button to a page where users can select the date for which they want to make their booking. After selecting the date, the user would be redirected back to this page, but the URL would now also contain the date for which the timetables should be generated.

Example URL: http://localhost:8080/booking/view/washing_machines?date=2022-02-08

After the user selects time periods and clicks “make a reservation” button, the input is validated and the user is redirected to the same page which contains either an error or a succession message.

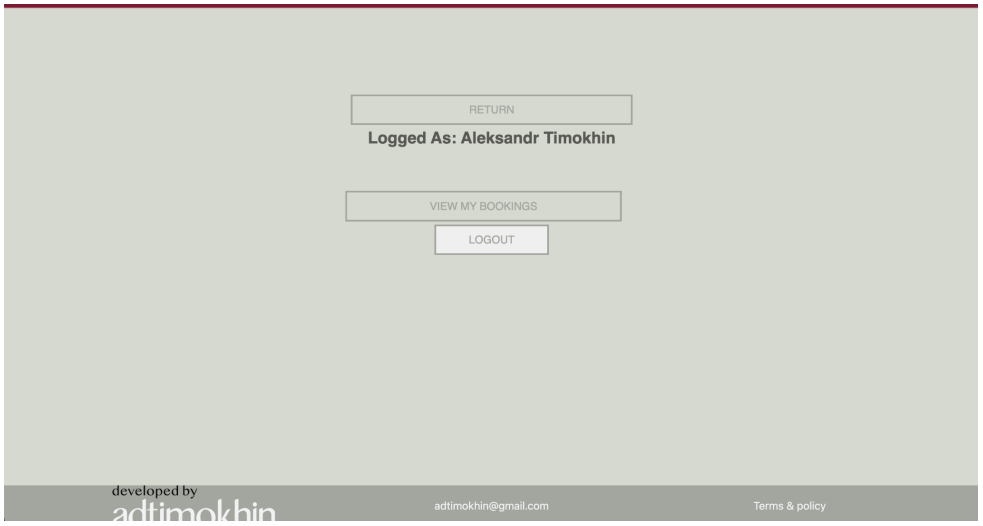
Select Date page



URL: http://localhost:8080/booking/choose/date/{machine type}?date={date}

On this page users can select a date for which timetables would be generated. Machine type in the URL helps to navigate to the correct page afterwards, and the date indicates the date for which the timetables are configured for. It is used to set the default value in the calendar input.

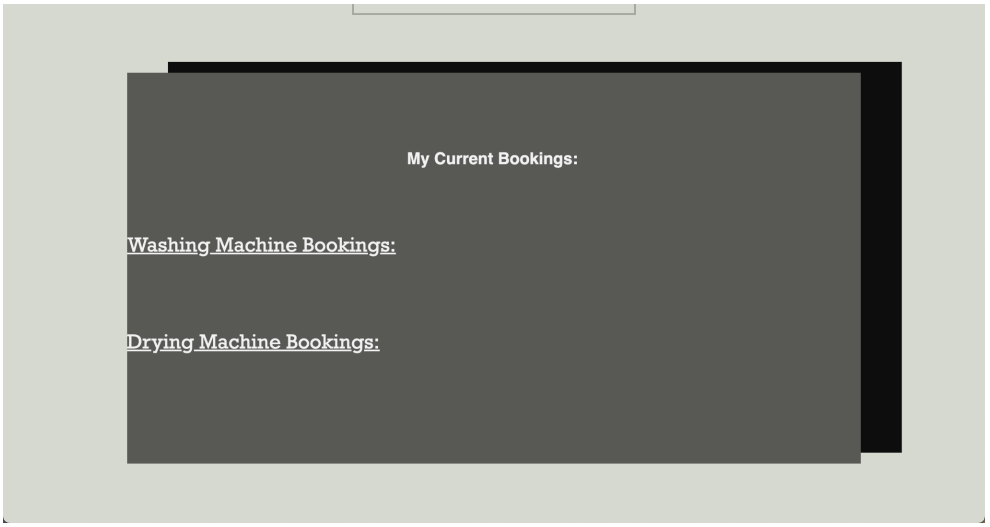
Account page



URL: http://localhost:8080/account/

This is another navigation page that currently allows users access a page about their most recent bookings. This page also has the “logout”button. It will make user to logout of the website.

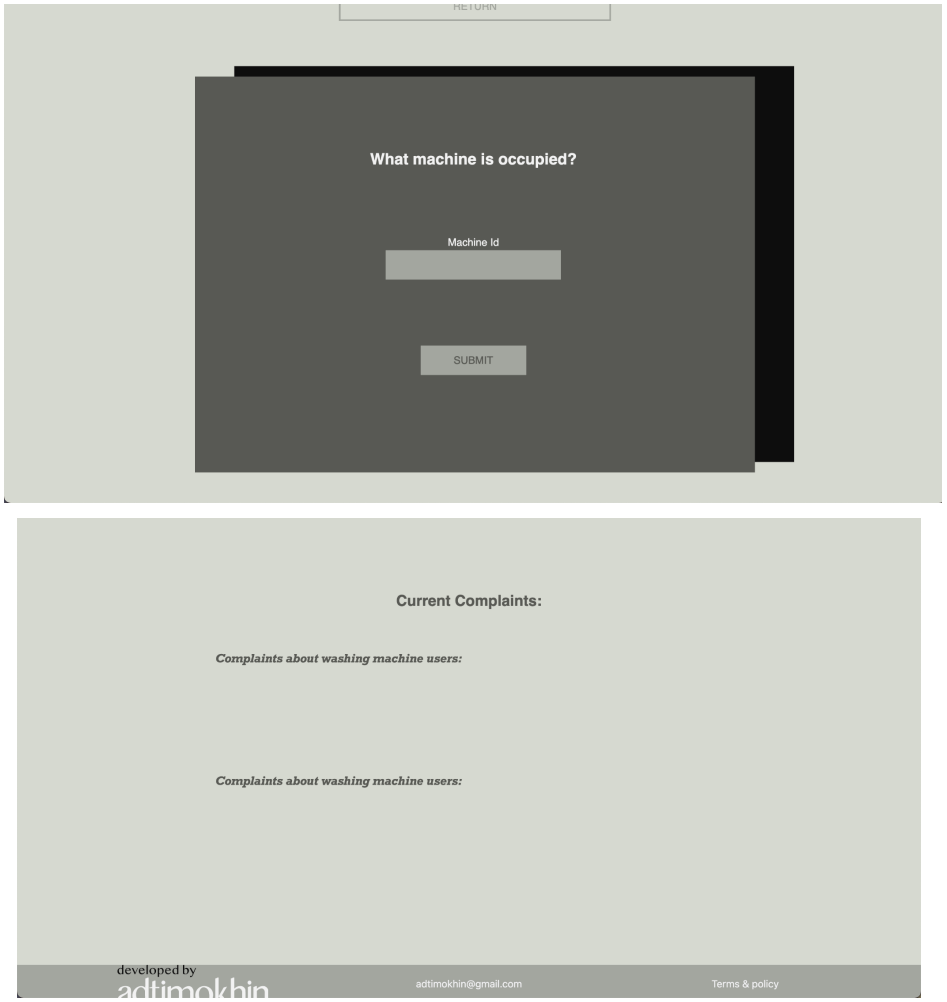
Current Bookings page



ULR: http://localhost:8080/account/bookings

This is a page that displays active bookings of the logged in user. Under each booking there is a button that allows the user to cancel that booking. If validations fail, the user would be redirected to this page that would have an error message on the top. If the validations were passed, instead of the error message, the user would see a confirmation that the booking was cancelled.

Complaints page



URL: http://localhost:8080/complaint/

This page allows users to enter a machine id that they should currently be using, but which is occupied.

If the complaint's validations are passed, the complaint would be generated and would be displayed under the "Current Complaints" section of that webpage. Otherwise, an error message would appear on top of the page.

TESTING SECTION

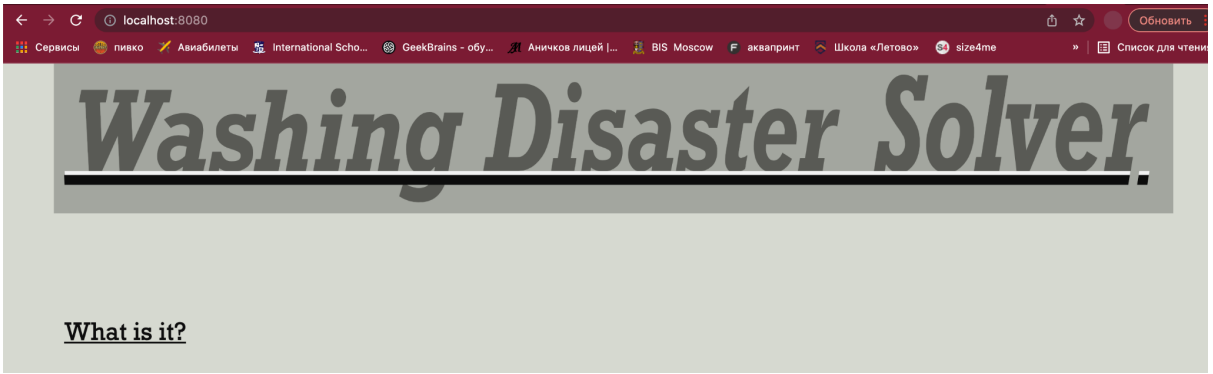
Test №1:

Purpose: To check that unauthenticated users cannot access content for authenticated users.

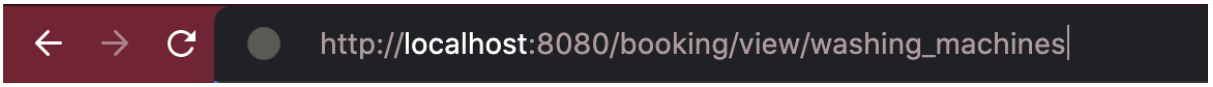
Test Data: Enter to the URL bar the following: "localhost:8080/booking/view/washing_machines". This is a valid URL that can only be accessed by an authenticated user. The user in our test would start from the index page, unauthenticated.

Expected Outcome: The user should be redirected to the login page.

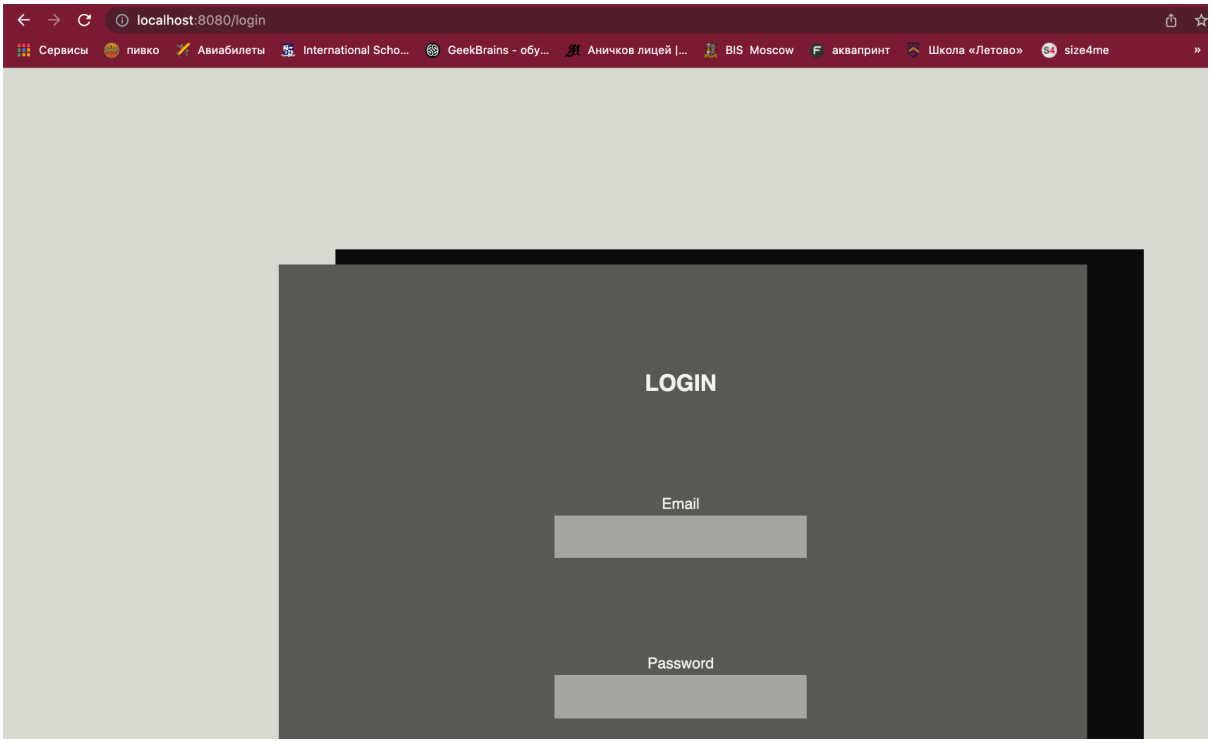
Actual Outcome:
Index page before entering another URL:



URL entered:



The page that the user is shown:



Result:

The test was passed successfully.

Test № 2:

Purpose: Test that when invalid data is entered into the sign-up form there are appropriate error messages.

Test Data:

Field	Value
email	sample1
name	Joe Green
password	qwerty
confirm password	qwerty1

Expected Outcome:

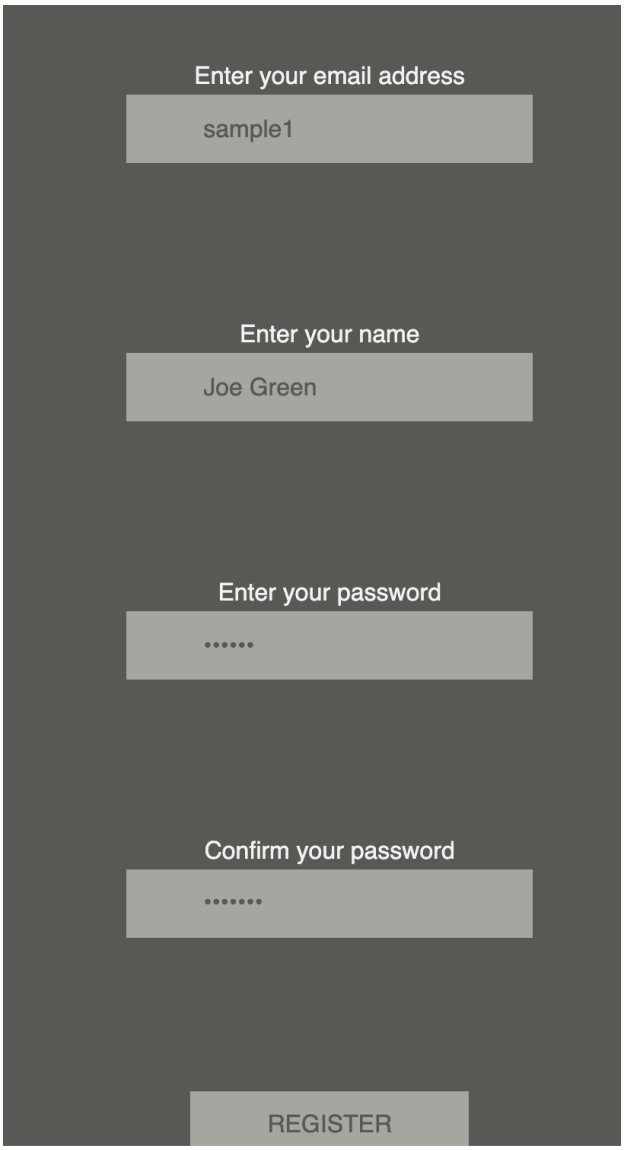
Two error messages should appear:

- email is in invalid form
- password should be at least 7 characters long

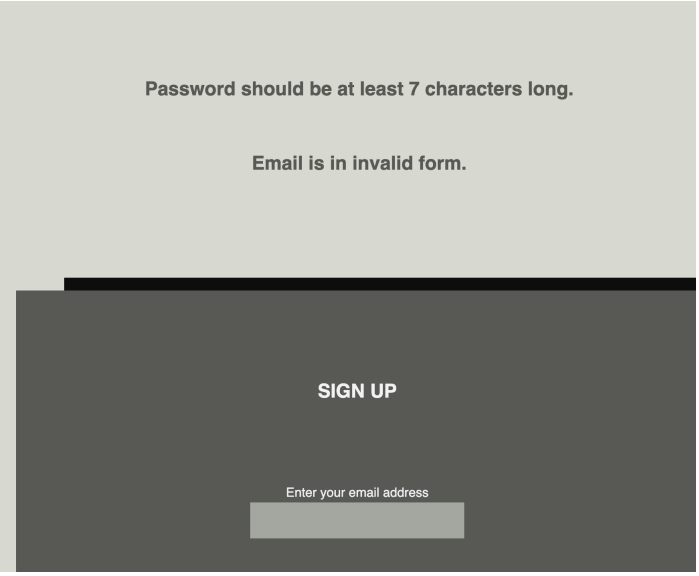
Other issues with the input's validity must not be addressed because they are not worth mentioning unless the errors above are fixed.

Actual Outcome:

Filled in form with the data from the table:



Error messages after the submission:



Result: The test was passed successfully.

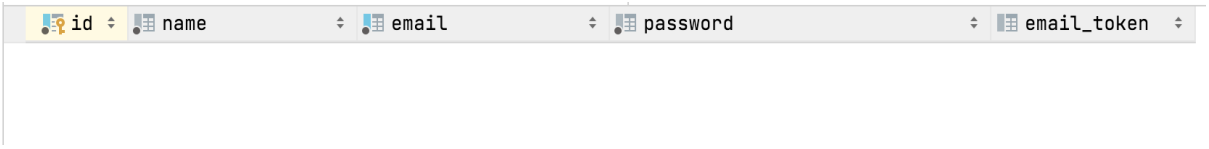
Test № 3:

Purpose: To check that email is sent with a unique email verification token to one's email if the sign-up form was filled correctly.

Test Data:
Input into the form:

Field	Value
email	adtimokhin@gmail.com
name	Aleksandr Timokhin
password	qwerty12345
confirm password	qwerty12345

Data in the user's table:

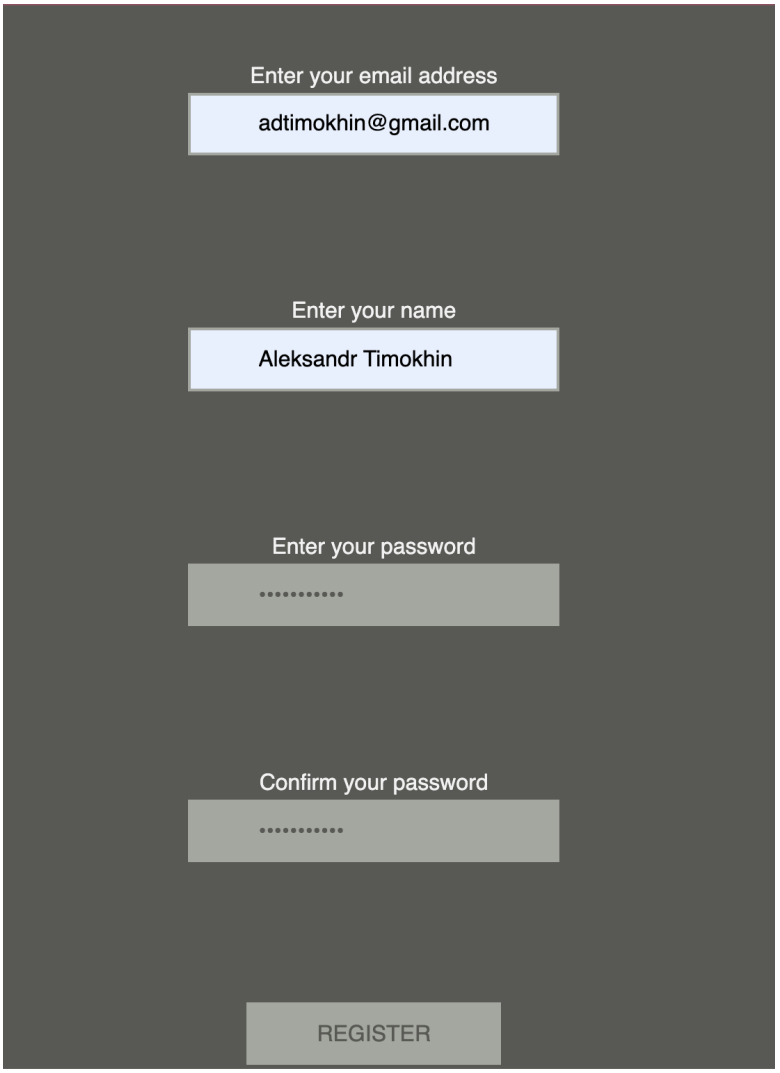


Expected Outcome:

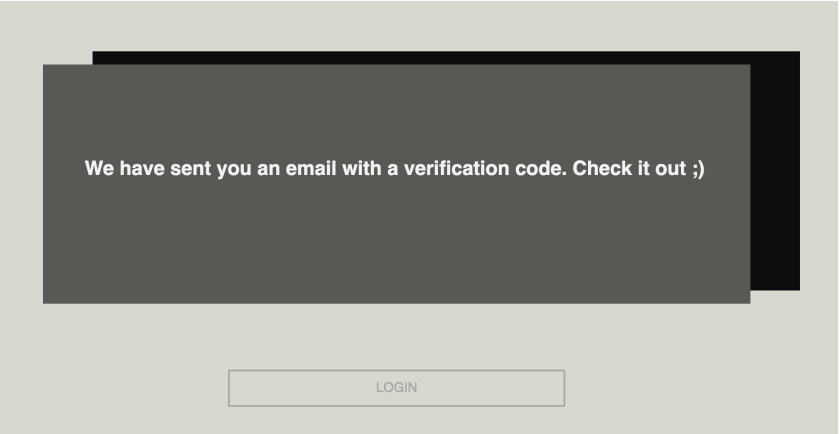
- There should be a message on the user's screen telling that an email verification token was sent to them.
- User is added to database with email of adtimokhin@gmail.com
- User's password must be encrypted
- A unique email verification token must be generated for that user
- An email with the link to verify the email must be sent to the email provided

Actual Outcome:

Filled in form with the data from the table:



Message on the user's screen:



Records in database after “register” button was pressed:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	88163

(The token is unique)

Mail dropbox after the user was added to the database:

Email verification Inbox ×

kolestnitsa-maria@yandex.ru
to me

Please, follow this link to verify your email:
localhost:8080/verify/88163

Result:

The test was passed successfully. Though the task of sending the email was not asynchronous - it took about 10 seconds before the user was shown a page with the “email is sent” message.

Test № 4:

Purpose: To validate that the emailToken routine truly generates unique tokens.

The code that generates the tokens has been repurposed to generate 100,000 unique tokens of 5 digits. Later the items in the list of tokens is fed into a set. If the sizes of the two data structures then the token-generating routine truly generates unique tokens.

Code:

```
import java.util.*;

/**
* @author adtimokhin
* 30.01.2022
**/

public class Main {

    private static int[] symbols;
    /**
     * Represents a length of the token that verifies emails
     */
    private final static int TOKEN_LENGTH = 5;


    private static final Random random = new Random();


    private static final List<String> tokens = new ArrayList<>();

    public static void main(String[] args) throws Exception {
        symbols = new int[10];
        int k = 48;
        for (int i = 0; i < 10; i++) {
            symbols[i] = k + i;
        }

        for (int i = 0; i < 100000; i++) {
            tokens.add(generate());
        }

        Set<String> set = new HashSet<>(tokens);
        System.out.println("set.size() = " + set.size());
        System.out.println("tokens = " + tokens.size());

    }


    public static String generate() throws Exception { // Objective 1.6.1

        if (tokens.size() == Math.pow((TOKEN_LENGTH), symbols.length)) {
            throw new Exception("Overflow of email tokens.");
        }


        List<Integer> chars = new ArrayList<>();
        for (int symbol : symbols) {
            chars.add(symbol);
        }

        // sort existing tokens
```

```
        return generateSymbol(chars, 0, tokens, "");
    }

    private static String generateSymbol(List<Integer> chars, int pos, List<String>
viableTokens, String currentToken) throws Exception {

        if (pos == TOKEN_LENGTH) {
            return currentToken;
        }

        // generate a list of chars in some random order
        Collections.shuffle(chars);

        for (int i = 0; i < chars.size(); i++) {
            int charToTry = Integer.parseInt(Character.toString(chars.get(i)));
            List<String> matchDigit = new ArrayList<>();
            for (String usedToken :
                    viableTokens) {
                if (Integer.parseInt(Character.toString(usedToken.charAt(pos))) ==
(charToTry)) {
                    matchDigit.add(usedToken);
                }
            }
            if (matchDigit.size() == 0) {
                currentToken += charToTry;
                return currentToken + generateRest(currentToken.length(), chars);
            } else if (matchDigit.size() != Math.pow(chars.size(), TOKEN_LENGTH - pos - 1))
{
                String potentialToken = generateSymbol(chars, pos + 1, matchDigit,
currentToken + String.valueOf(charToTry));
                if (potentialToken != null) {
                    return potentialToken;
                }
            }
        }
        if (pos == 0) {
            throw new Exception("Overflow of email tokens.");
        }
        return null;


    }

    private static String generateRest(int startPosition, List<Integer> symbols) {
        StringBuilder stringBuilder = new StringBuilder();

        for (int i = 0; i < TOKEN_LENGTH - startPosition; i++) {
            stringBuilder.append(Integer.parseInt(Character.toString(symbols.get(i))));
        }

        return stringBuilder.toString();
    }
}
```

Expected Outcome:

The lengths of the set and list would both equal 100,000.

Actual Outcome:

```
set.size() = 100000
tokens = 100000

Process finished with exit code 0
```

Test № 5:

Purpose: To check without verified email address users cannot log into the platform, even though the user credentials are entered correctly.

Test Data:
Input into the login form:

Field	Value
email	adtimokhin@gmail.com
password	qwerty12345

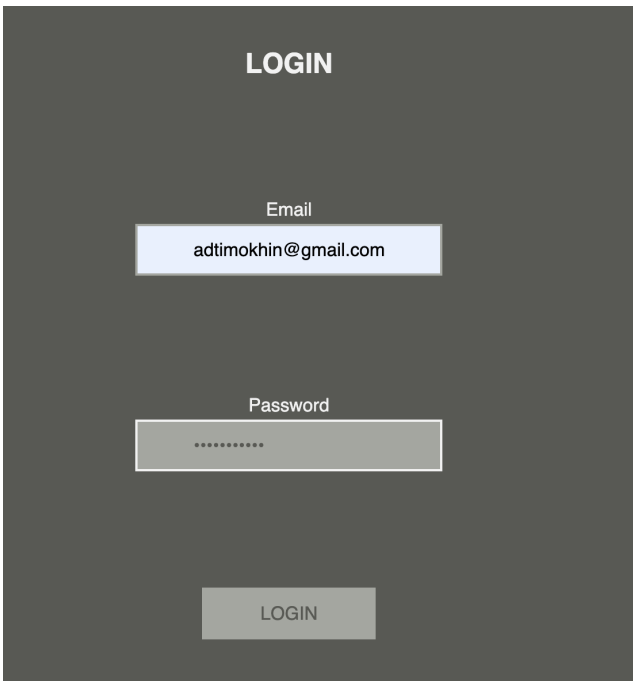
User's record in the database:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	88163

Expected Outcome:

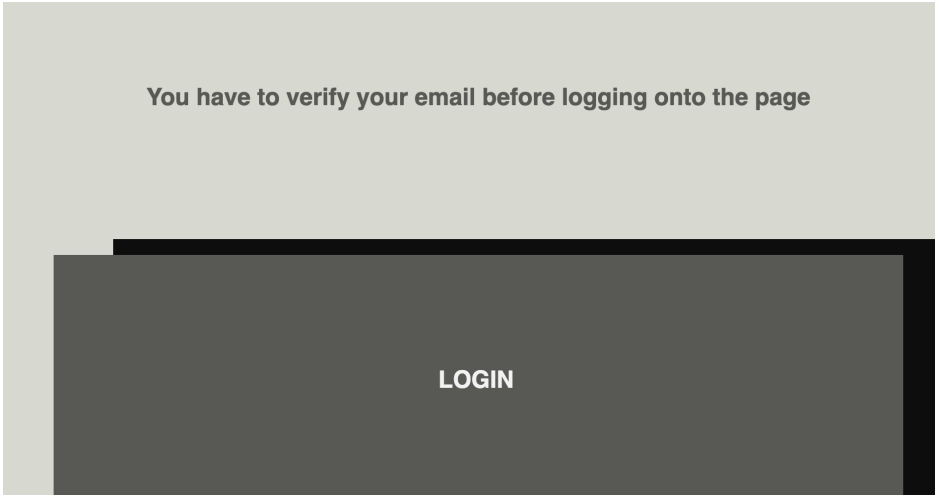
- After entering data into the form and sending it, the user should be redirected to the same page.
- URL now must end on "?error=true"
- There should be a message on the top of the page telling: "You have to verify your email before logging onto the page".

Actual Outcome:
Filled in form with the data from the table:



The URL after the “login” button was pressed:

The login page after the page had reloaded:



Result: The test was passed successfully.

Test № 6:

Purpose: To test that following the link provided in the email actually marks the email as valid.

Procedure:

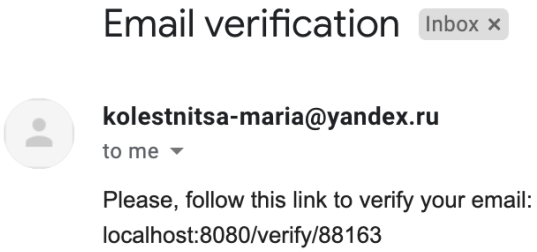
- Follow the link sent to the email.

Test Data:

- User data:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	88163

- Email received:

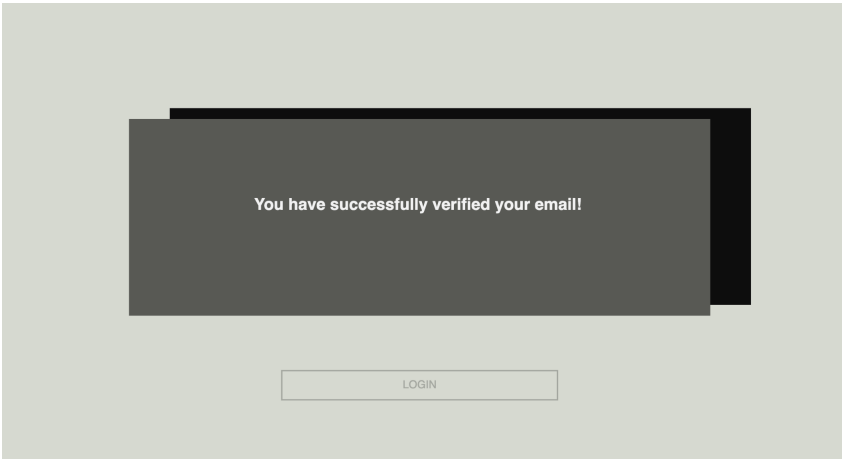


Expected Outcome: After following the link the email token from the database (related to user with id 60) should be set to null.

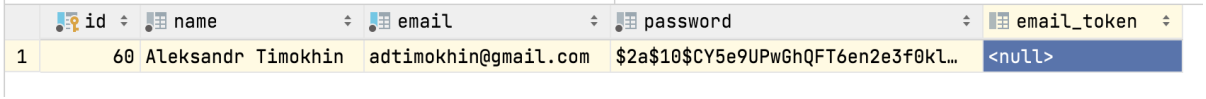
Actual Outcome:

- Link followed:

- Message page received from the server:



- Data in the database after following the link:



	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	<null>

Result: The actual outcome met the predictions, hence the test was passed successfully and the system can verify emails.

Test № 7:

Purpose: To validate that the user will not be let onto the website if the login information provided is incorrect.

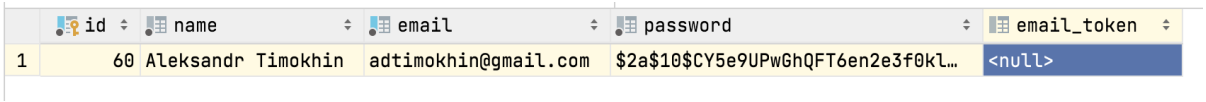
Procedure:

- Enter the test data into the login form.
- Observe the error message

Test Data:
Data to enter into the form:

Field	Value
email	adtimokhin@gmail.com
password	qwerty12345qwerty12345

Data in the database:

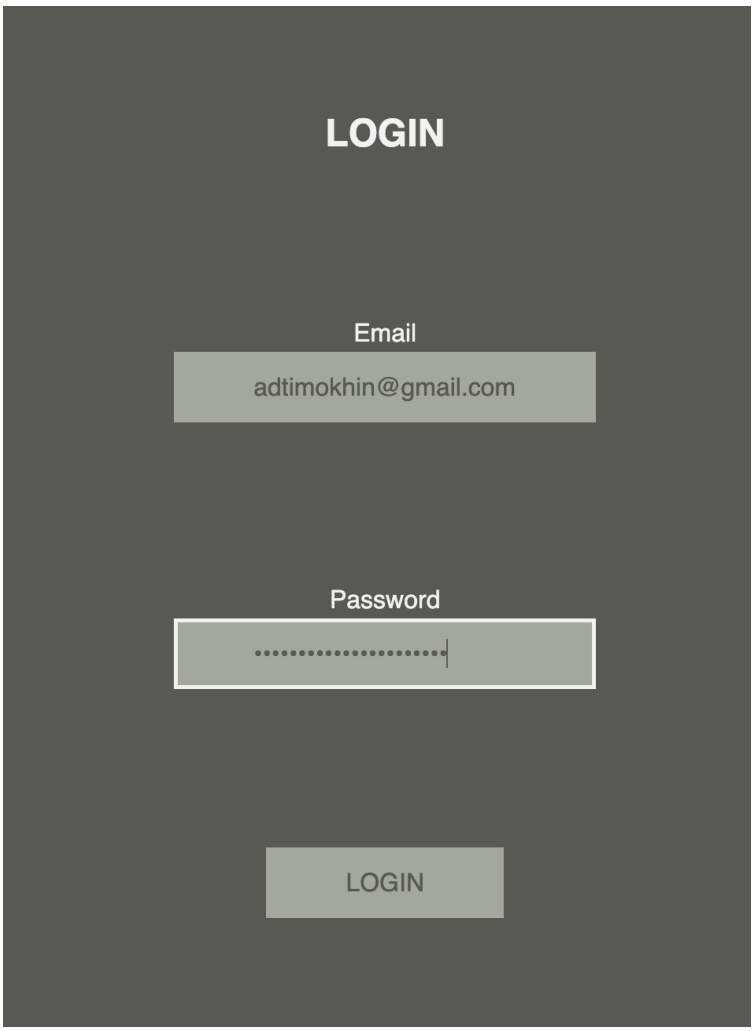


	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	<null>

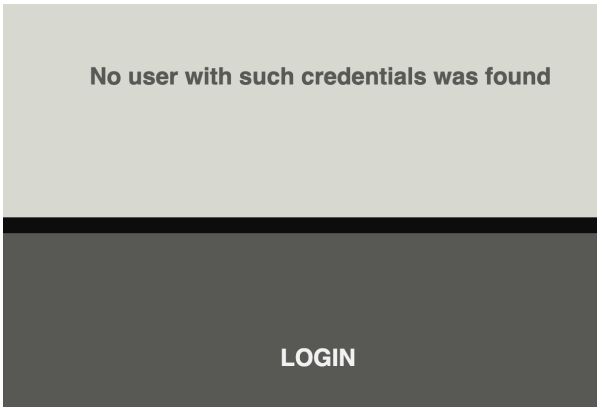
Expected Outcome:
As the password entered does not match the original (qwerty12345), the user must be redirected to the login page. On the page the user must see a message that tells that user with the credentials provided was not found in the database.

Actual Outcome:

- Login form filled in:



- User's screen after the user pressed the "login" button:



Result: The test was passed successfully.

Test № 8:

Purpose: To verify that logout functionality functions properly.

Procedure:

- Authenticate the user with the credentials stored in the users' database.
- Navigate to the http://localhost:8080/account/ page
- Press the "logout" button.
- Observer where the user is redirected.
- Attempt to enter a URL that should only be accessed by an authenticated user.
- Observe where the user would be redirected.

Test Data:

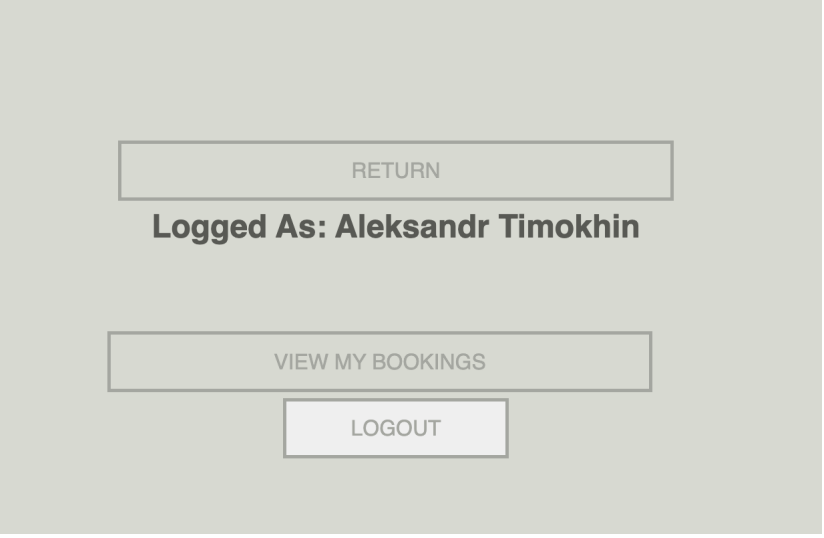
- Users table:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	<null>

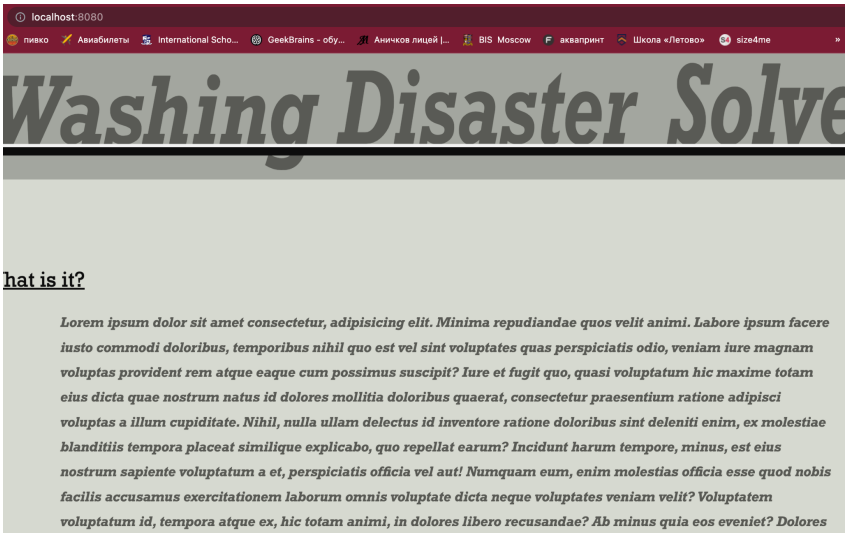
Expected Outcome:
After pressing the "logout" button, the user should be redirected to the index page. On an attempt to visit a page with URL http://localhost:8080/account/bookings the user should remain on the login page.

Actual Outcome:

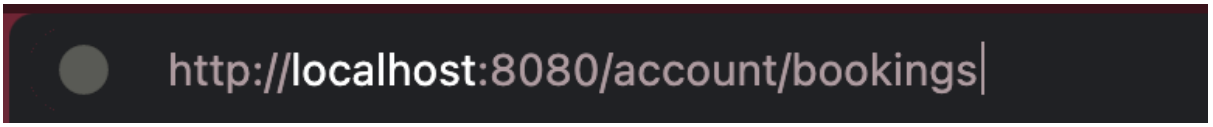
- Account page after authenticating:



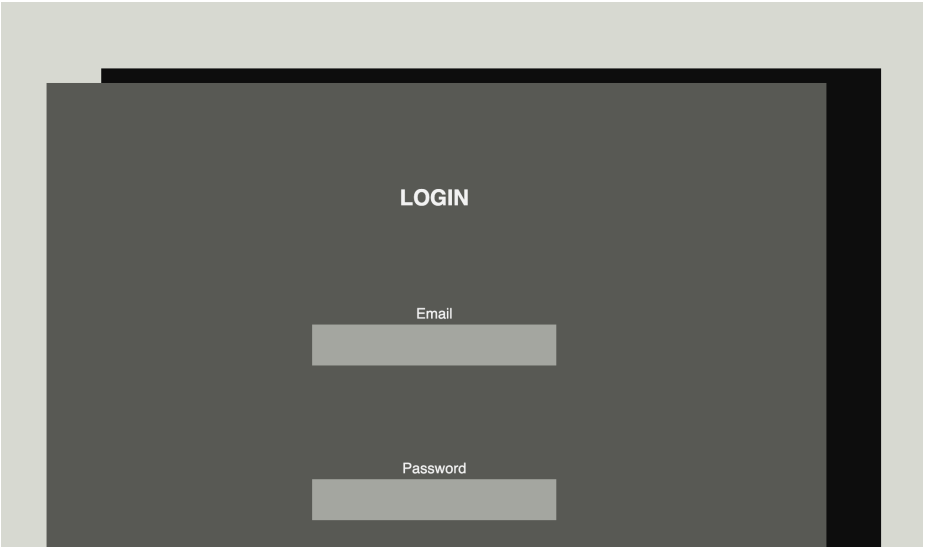
- User's screen after pressing the "logout" button:



- URL entered after logging out:



- User's screen after pressing after attempting to view the http://localhost:8080/account/bookings page:



Result:
The actual outcome had followed the predicted behaviour. Thus, the test is successfully passed.

Test № 9:

Purpose: To check that after a user has selected another date on a special page, the content of the drying machine time tables updates. The test also checks that the bookings are presented correctly on the page (that means that the booked slots a coloured differently and have no tick box next to them).

How This Would Be Tested:

1) Make up a total of 5 bookings for drying machines for 2 separate days.
2) 5 of these bookings should be setted for today (22 December 2021)
3) 5 of the bookings must be set for 25 December 2021

Test Method:

Users must enter a page with the drying machine bookings. Then they should go to the page where they should select 25 December. Then they should be redirected to the drying machine bookings page with the updated content.

Test Data:

Machine bookings table:

	id ▲ 1	machine_id	user_id	start_time	end_time
1	39	7	60	2021 12 25 14:30	2021 12 25 15:00
2	40	8	60	2021 12 25 10:00	2021 12 25 12:00
3	41	6	60	2021 12 25 09:00	2021 12 25 11:00
4	42	5	60	2021 12 25 18:00	2021 12 25 20:00
5	43	5	60	2021 12 22 18:00	2021 12 22 19:00
6	44	8	60	2021 12 22 09:00	2021 12 22 09:30
7	45	8	60	2021 12 22 10:00	2021 12 22 12:30
8	46	7	60	2021 12 22 13:00	2021 12 22 14:00
9	47	6	60	2021 12 22 09:00	2021 12 22 10:00
10	50	6	60	2021 12 25 11:00	2021 12 25 12:00

Machine table:

	id	type
1	1	1
2	2	1
3	3	1
4	4	1
5	5	2
6	6	2
7	7	2
8	8	2

Machine Types table:

	id	description
1	1	WASHING_MACHINE
2	2	DRYING_MACHINE

Expected Outcome:
When enter the "view bookings for the drying machines" page for the first time we should see:

- Date 2021-12-22 stated at the top of the page
- 1 booking for the drying machine 5 (from 18:00 until 19:00)
- 1 booking for the drying machine 6 (from 9:00 until 10:00)
- 1 booking for the drying machine 7 (from 13:00 until 14:00)
- 2 bookings for the drying machine 8 (from 11:00 until 12:00 and from 9:00 until 11:00)

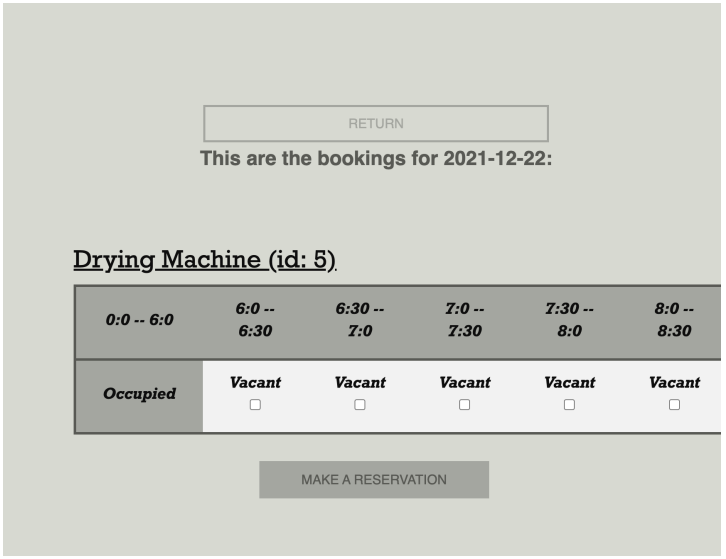
When we press the "change date" button on the same page we must be redirected to a page where we can select a date.

After selecting the date to be the 25th of December 2021, we must be redirected back to the "view bookings for the drying machines" page where we should see the following:

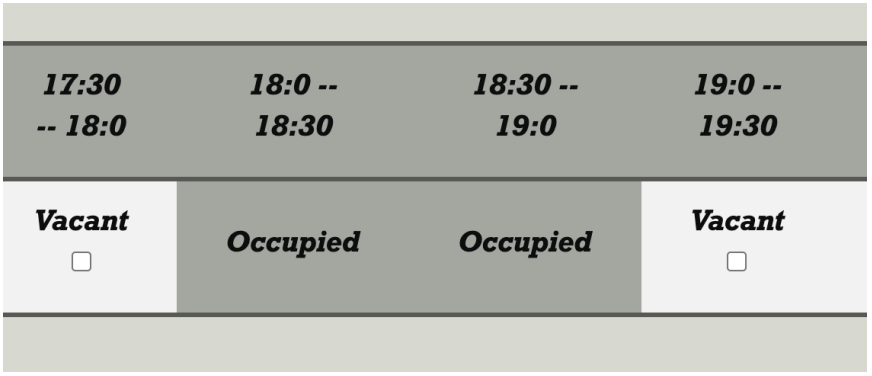
- Date 2021-12-25 stated at the top of the page.
- 1 booking for the drying machine 5 (from 18:00 until 20:00)
- 2 bookings for the drying machine 6 (from 10:00 until 12:00 and from 18:00 until 20:00)
- 1 booking for the drying machine 7 (from 14:30 until 15:00)
- 1 booking for the drying machine 8 (from 10:00 until 12:00)

For the times that are marked as booked, there should be no available tick boxes to press, and the colour of the matching cells must be grey-ish.

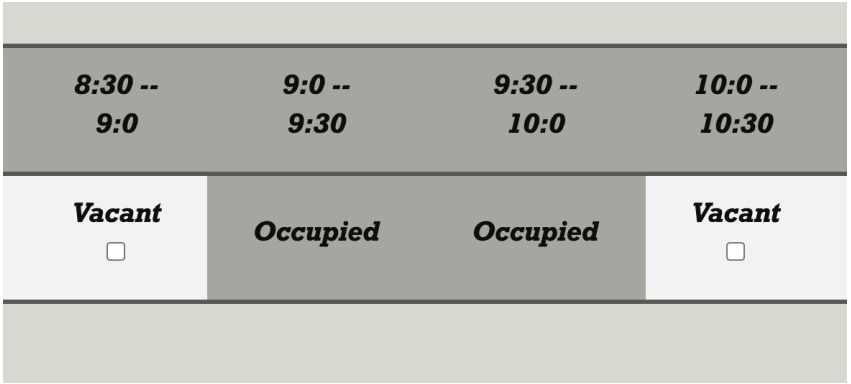
Actual Outcome:
The page, when the user enters it for the first time:



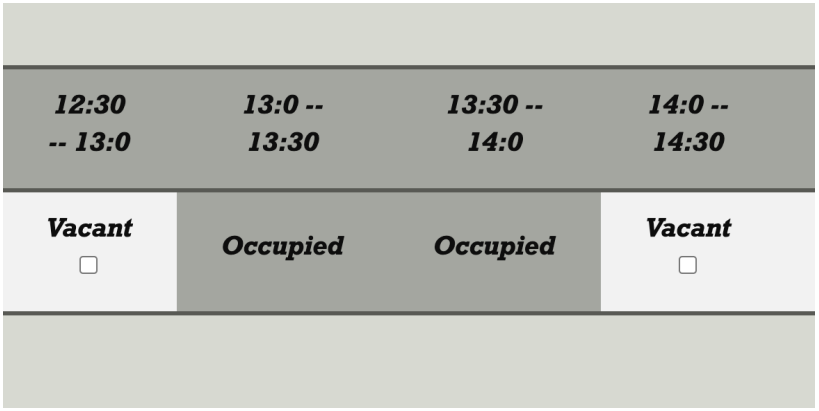
Machine with id 5:



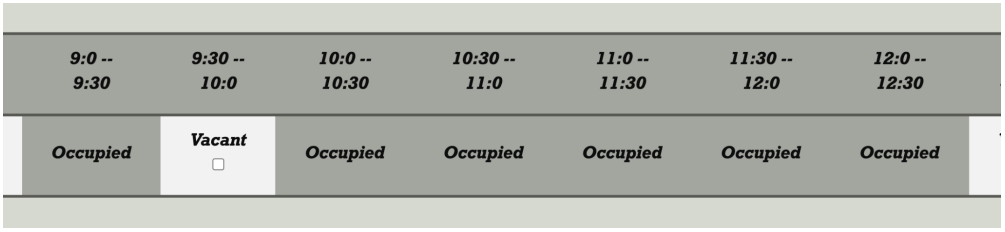
Machine with id 6:



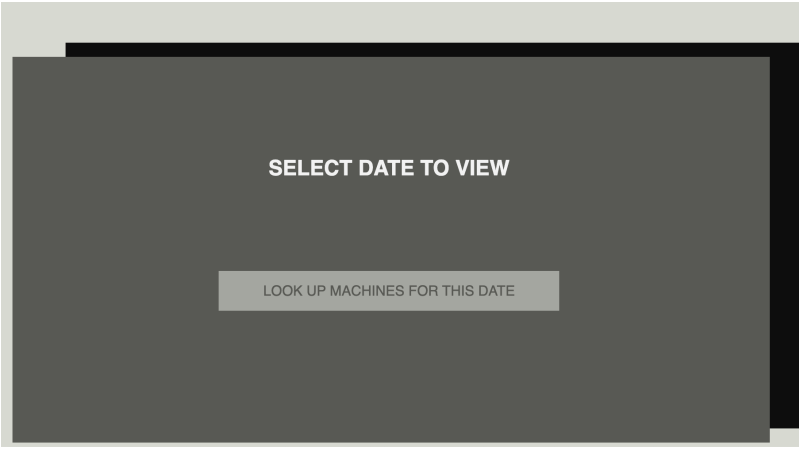
Machine with id 6:



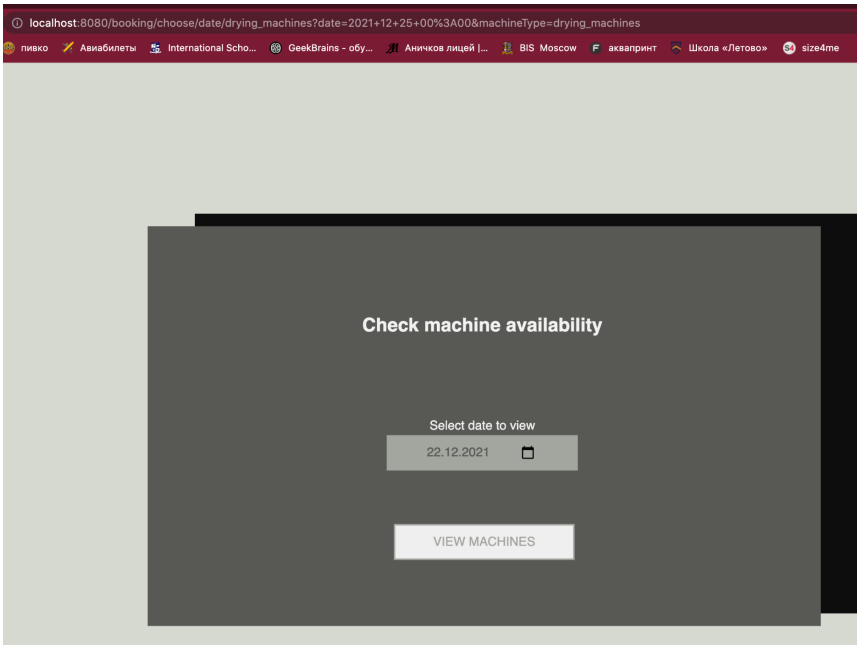
Machine with id 8:



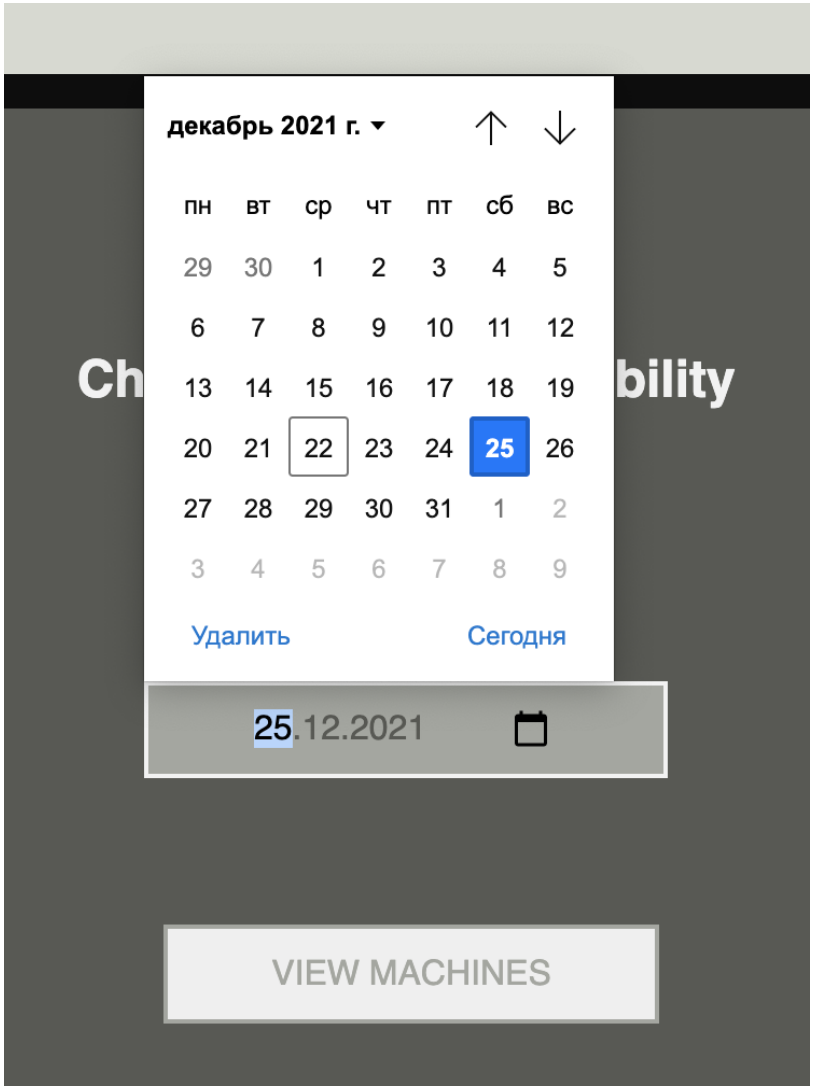
The “select date to view” button on the “view bookings for the drying machines” page:



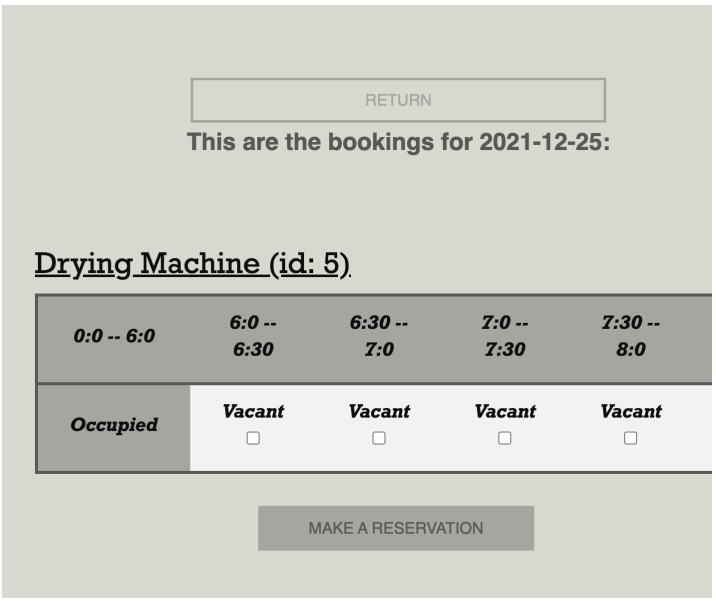
Page that appears after the button was pressed:



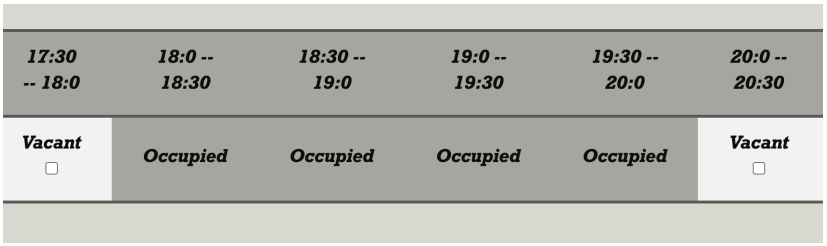
Selecting 25th of December in the box provided:



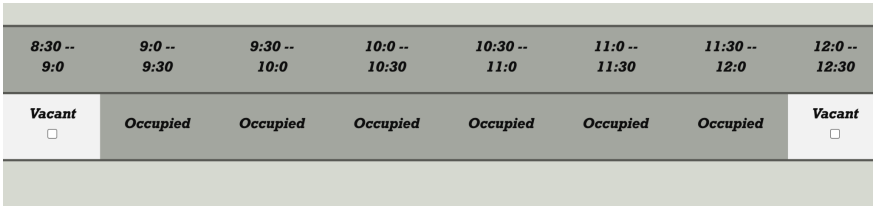
After pressing the “view machines” button:



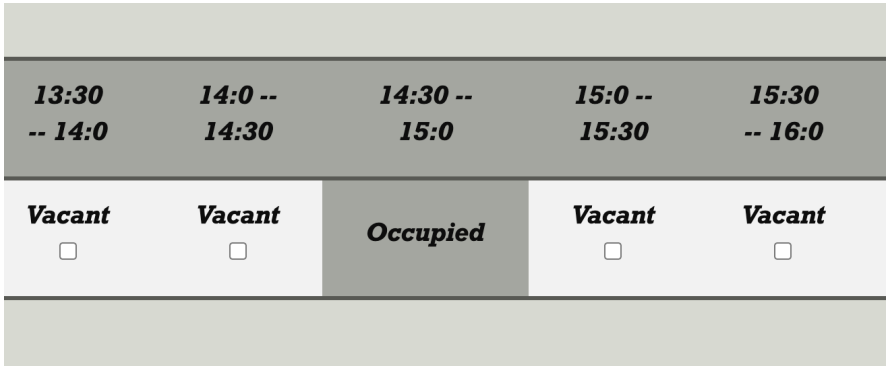
Machine with id 5:



Machine with id 6:



Machine with id 7:



Machine with id 8:

9:30 -- 10:0	10:0 -- 10:30	10:30 -- 11:0	11:0 -- 11:30	11:30 -- 12:0	12:0 -- 12:30
Vacant ☐	Occupied	Occupied	Occupied	Occupied	Vacant ☐

Result:
The test was passed. The test has also proven that the program sorts the time periods correctly, as the order in which they were stored in the database was not sequential.

***Test Nº 10*:**

Purpose: To test that with the appropriate input the booking would be generated for the time selected.

Procedure:

- Enter a page that displays washing machines for date 4/02/2022
- Select 3 slots for machine with id 3: slots between 6:00 and 7:30
- Press the “submit” button

Test Data:

- User table:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$DTYIxCMJEh0i9mHWEK3Ww.n…	<null>

- Machines table:

	id	type
1	1	1
2	2	1
3	3	1
4	4	1
5	5	2
6	6	2
7	7	2
8	8	2

Machine Types table:

	id	description
1	1	WASHING_MACHINE
2	2	DRYING_MACHINE

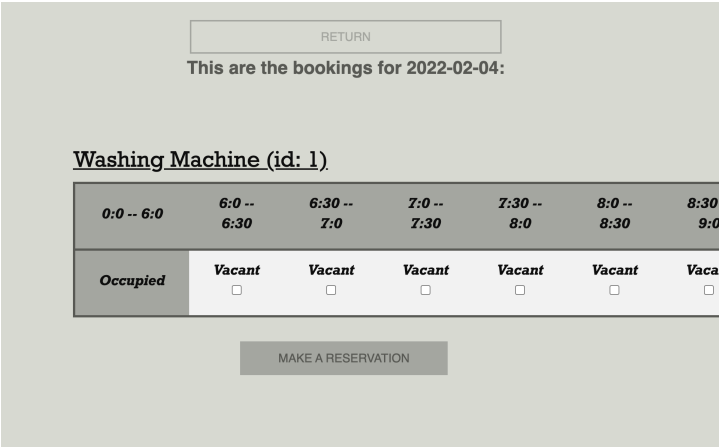
- Bookings table:

id	machine_id	user_id	start_time	end_time

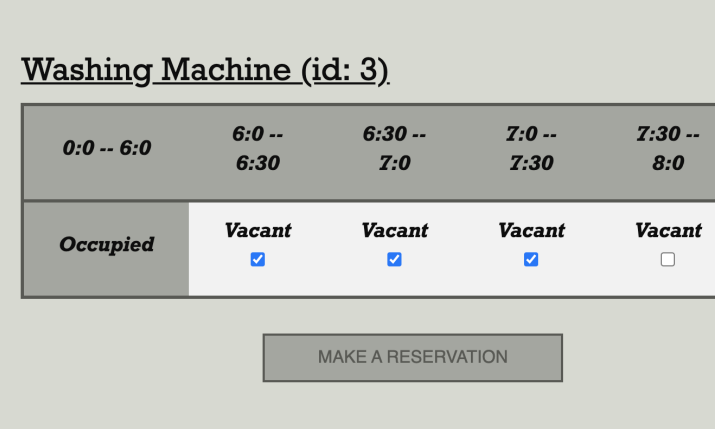
Expected Outcome: Upon selecting the slots mentioned above a new booking should be generated. It should have start_time=”2022 02 04 06:00”, finish_time=”2022 02 04 07:30”, user_id=60, machine_id=3

Actual Outcome:

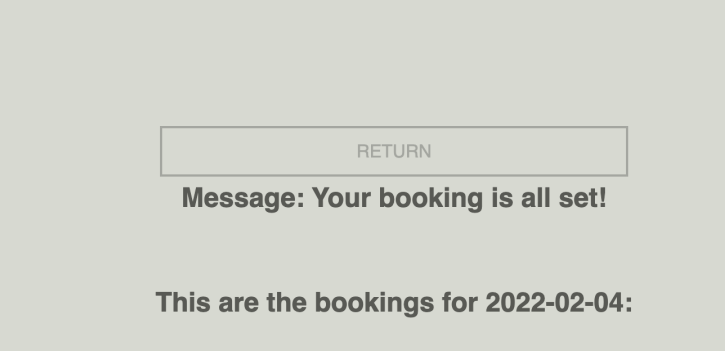
- Page that displays washing machine bookings for 2022/02/04:



- Selected time slots:



- Message on the screen upon submission:



- Booking table:

	id	machine_id	user_id	start_time	end_time
1	160	3	60	2022 02 04 06:00	2022 02 04 07:30

Result:
Validation mechanism works as expected as the booking was saved with the valid data inputted. The start and end times are set correctly, so are machine and user ids. Therefore, the system saves the bookings correctly.

Test Nº 11:

Purpose: To verify that users can cancel their bookings.

Procedure:

- After authentication navigate to a URL: http://localhost:8080/account/bookings.
- Press the “cancel” button under one of the bookings.

- Navigate back to the http://localhost:8080/account/bookings page to observe whether the booking cancelled still appears in the list of active bookings.

Test Data:

- Users table

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	<null>

- Bookings table

	id	machine_id	user_id	start_time ▲ 1	end_time
1	260	3	60	2022 02 08 06:00	2022 02 08 07:30
2	262	3	60	2022 02 08 06:00	2022 02 08 08:00

Expected Outcome:
(Time when this test is conducted: 2022/02/05 12:19)

Both bookings listed above should appear on the page http://localhost:8080/account/bookings under the "washing machine bookings" section. Beneath each of the bookings should be the "cancel" button.

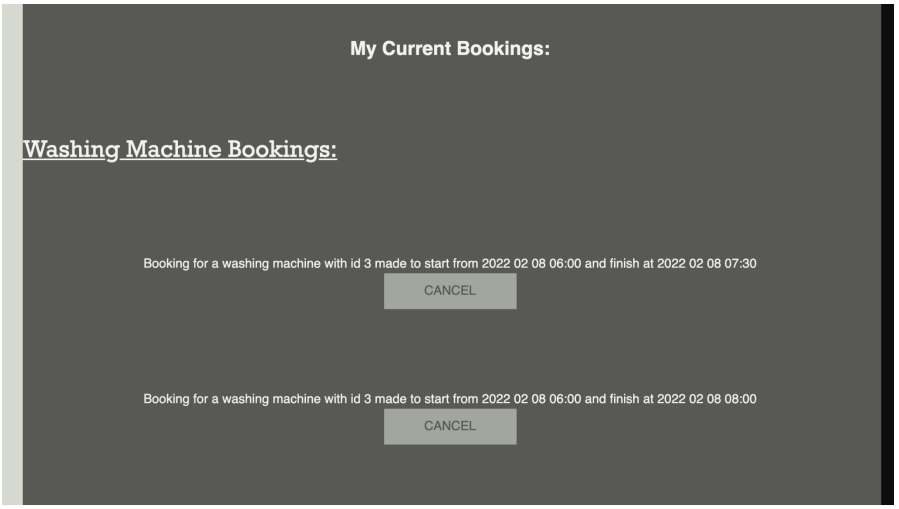
After pressing the button under the booking with id 262, the user should be redirected to a page where they should see a message telling that the booking was successfully cancelled.

After that the user should no longer see the booking in the "current bookings" section as the booking 262 should be deleted from the database.

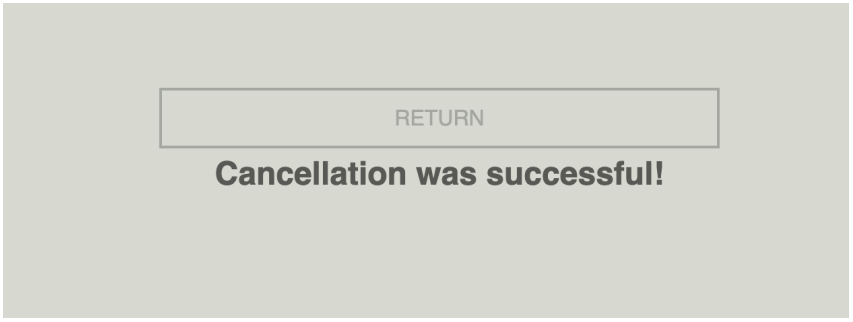
There also should be a message in the log files telling that a user with id 60 cancelled booking with id 262.

Actual Outcome:

- http://localhost:8080/account/bookings after logging into the portal:



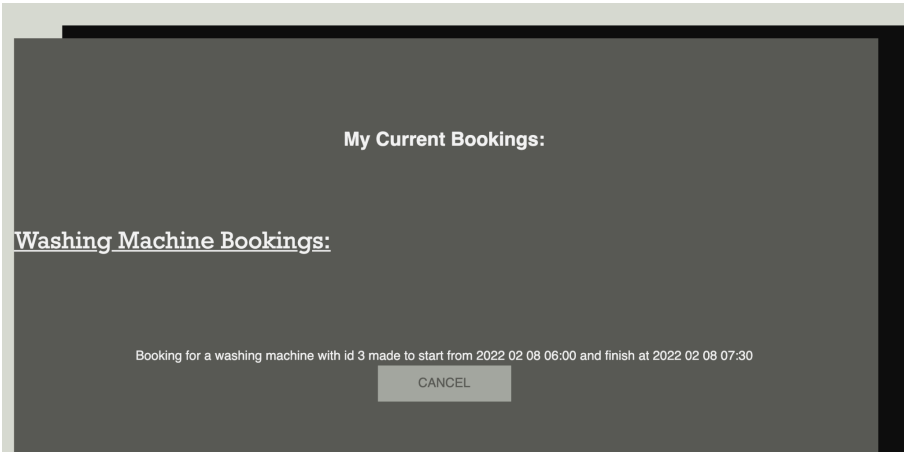
- User's screen after pressing the cancel button beneath the booking with id 262:



- Bookings in the database:

	id	machine_id	user_id	start_time ▲ 1	end_time
1	260	3	60	2022 02 08 06:00	2022 02 08 07:30

- User's screen capture of the "My current bookings" page:



- Trace.log:

```
2022-02-05 12:19:32 TRACE [com.adtimokhin.controller.BookingController.cancelBooking(BookingController.java:274)]:
 About to delete booking for machine id 3, for time 2022 02 08 06:00 - 2022 02 08 08:00 and by user 60
2022-02-05 12:19:32 TRACE [com.adtimokhin.controller.BookingController.cancelBooking(BookingController.java:276)]:
 Cancellation was successful!
```

Note: The test data for bookings are invalid. They are both linked to the same machine and they overlap in time. That is so because I have manually inserted the booking data into the database. That is another great example why validations are necessary - they act as an entry point for data, eliminating the need for future validations. As the data was entered manually into the database, those validations did not happen.

Result: The actual outcome has exactly followed the predicted behaviour.

Test № 12:

Purpose: To check that at 22:30 all bookings for a current date are deleted from the database and stored in a text file.

Test Data:
Machine bookings table:

	id ▲ 1	machine_id	user_id	start_time	end_time
1	39	7	60	2021 12 25 14:30	2021 12 25 15:00
2	40	8	60	2021 12 25 10:00	2021 12 25 12:00
3	41	6	60	2021 12 25 09:00	2021 12 25 11:00
4	42	5	60	2021 12 25 18:00	2021 12 25 20:00
5	43	5	60	2021 12 22 18:00	2021 12 22 19:00
6	44	8	60	2021 12 22 09:00	2021 12 22 09:30
7	45	8	60	2021 12 22 10:00	2021 12 22 12:30
8	46	7	60	2021 12 22 13:00	2021 12 22 14:00
9	47	6	60	2021 12 22 09:00	2021 12 22 10:00
10	50	6	60	2021 12 25 11:00	2021 12 25 12:00

Expected Outcome:

- At 22:30 all bookings made for the 22th of December (current date) should be removed from the table above. That includes bookings with ids: 43, 46, 47, 45, 44.
- A text file named bookings_2021_12_22.txt should be created
- Inside the file all deleted bookings must be stored in a format such as: `Booking{id=50, userId=8, MachineId=1, startDate='2021 11 06 09:45', endDate='2021 11 06 12:00'}`
- There should be messages in the trace.log file that trace that bookings for the current day were deleted from the database.

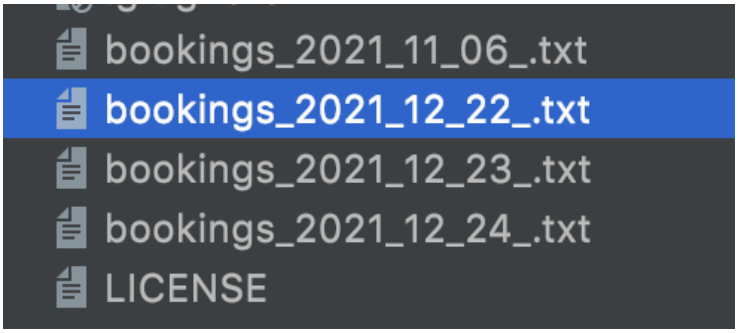
Actual Outcome:

State of the database at 2021-12-22 22:30:05:

	id	machine_id	user_id	start_time ▲ 1	end_time
1	41	6	60	2021 12 25 09:00	2021 12 25 11:00
2	40	8	60	2021 12 25 10:00	2021 12 25 12:00
3	50	6	60	2021 12 25 11:00	2021 12 25 12:00
4	39	7	60	2021 12 25 14:30	2021 12 25 15:00
5	42	5	60	2021 12 25 18:00	2021 12 25 20:00

(bookings with ids: 43, 46, 47, 45, 44 are now gone.)

Project's directory:



(A text file with the expected file name was generated).

Contents of the bookings_2021_12_22_.txt file:

```
Booking{id=43, userId=60, dryingMachineId=5, startDate='2021 12 22 18:00', endDate='2021 12 22 19:00'}
Booking{id=47, userId=60, dryingMachineId=6, startDate='2021 12 22 09:00', endDate='2021 12 22 10:00'}
Booking{id=46, userId=60, dryingMachineId=7, startDate='2021 12 22 13:00', endDate='2021 12 22 14:00'}
Booking{id=45, userId=60, dryingMachineId=8, startDate='2021 12 22 10:00', endDate='2021 12 22 12:30'}
Booking{id=44, userId=60, dryingMachineId=8, startDate='2021 12 22 09:00', endDate='2021 12 22 09:30'}
```

(All bookings deleted from the database are present in this file and all information that was stored in there is present in the file correctly).

trace.log file:

```
138
139
140 2021-12-22 22:30:00 TRACE [com.adtimokhin.util.backup.BackupUtil.saveBookingInformation(BackupUtil.java:69)]:
141 Initiating deletion of all bookings for 2021 12 22 22:30.
142 2021-12-22 22:30:01 TRACE [com.adtimokhin.util.backup.BackupUtil.saveBookingInformation(BackupUtil.java:102)]: All bookings were deleted.
143
```

(messages that tell that deletion of the bookings was initialised and completed for the 22th of December).

Result: The test was passed successfully. The system has behaved as expected.

Test № 13:

Purpose: To check that there appear logging messages during execution of methods that are annotated with @DebugMethod annotation.

Procedure: I will run the website and try to login. The controller method that is responsible for the GET login/ request would obtain the @DebugMethod annotation (for the purpose of the test).

Expected Outcome:
Before I will get a HTML page that I would request by the following URL: http://localhost:8080/login , there should appear a message in trace.log. The message should say:

About to execute method loginGet with parameters {...}

After I receive the page, another log entry should be generated. It should say:

Finished executing method loginGet

Actual Outcome:

- URL entered:

- New messages in the trace.log:

```
2022-02-05 18:24:19 TRACE
[com.adtimokhin.controller.aspect.DebugAspect.testExecution(DebugAspect.java:39)]: About to execute method loginGet with parameters
java.lang.Boolean,org.springframework.ui.Model,javax.servlet.http.HttpServletRequest
2022-02-05 18:24:19 TRACE
[com.adtimokhin.controller.aspect.DebugAspect.testExecution(DebugAspect.java:39)]: Finished executing method loginGet
```

Result: The predicted behaviour was validated.

Test № 14:

Purpose: To verify that users can access a page that would display **their** active bookings.

Procedure:

- After authentication navigate to a URL: http://localhost:8080/account/bookings
- Observe what bookings are displayed.

Test Data:

- User table:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$CY5e9UPwGhQFT6en2e3f0kl…	<null>

- Booking table:

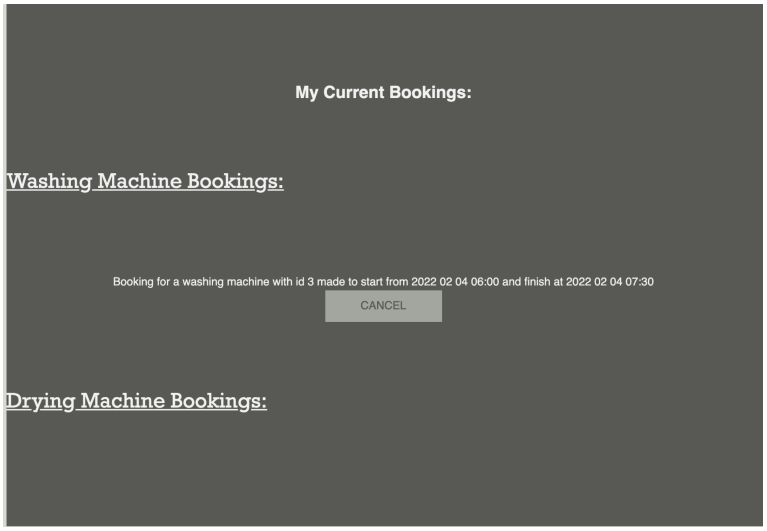
	id	machine_id	user_id	start_time ▲ 1	end_time
1	161	3	60	2022 01 04 06:00	2022 01 04 08:00
2	160	3	60	2022 02 04 06:00	2022 02 04 07:30

Note, the first booking is still active (the moment the test is conducted is 2022/02/03 13:13) and the second is not.

Expected Outcome: User must see a page that displays a booking information about a booking with id 160. No data should be displayed about booking with id 161.

Actual Outcome:

- http://localhost:8080/account/bookings:



Result:
Result meets the predictions. Data about the booking with id 160 is displayed in the correct format. The booking was correctly identified to be a booking for a washing machine. No data about a booking with id 161 is shown, as expected. Hence, the test passed successfully.

TEST Nº 15:

Purpose: To see that users can create Complaint objects.

Procedure:

- Add data from the test data section into the database
- Go to http://localhost:8080/complaint/
- Enter machine id 2
- Press submit button
- Observe whether a message that verifies that the complaint was created
- Check whether a complaint was created and stored in the database

Test Data:

- User table:

	id	name	email	password	email_token
1	60	Aleksandr Timokhin	adtimokhin@gmail.com	$2a$10$a9JCN7WnyhX27TkfVxozKebs…	<null>
2	76	Test User	test@test.com	example	<null>

- Booking table:

	id	washing_machine_id	user_id	start_time	end_time
1	49	2	60	2022 02 05 19:00	2022 02 05 19:30
2	162	2	76	2022 02 05 18:00	2022 02 05 19:00

- Complaint table:

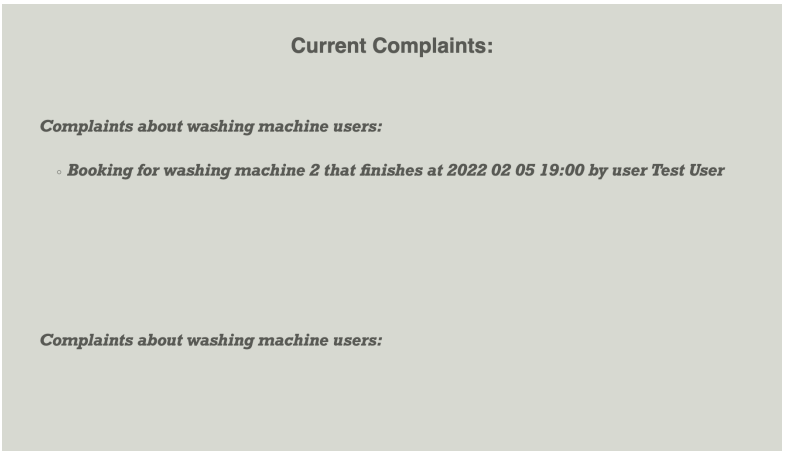
reporting_user_id	booking_id	id

Expected Outcome:
After authenticating as a user with id 60, I should navigate to http://localhost:8080/complaint/ and enter machine id 2. A complaint on user 76 should be generated and on my screen I should see a message that tells that the complaint was created.

Actual Outcome:

- http://localhost:8080/complaint/ :

- Message after pressing the “submit” button:



- Complaint table:

	id	reporting_user_id	booking_id
1	3	60	162

Result: Test was successfully passed.

Test № 16:

Purpose: To verify that users would be redirected to a special HTML error page when the URL entered does not match any of the controllers’ methods.

Procedure:

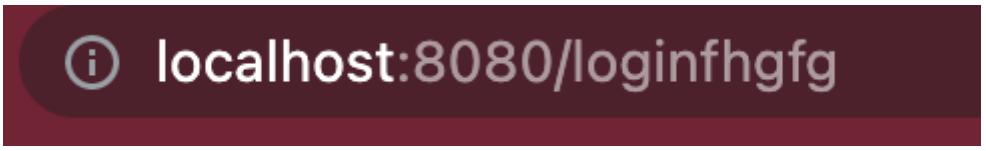
- Enter into URL a link to the server that should lead to no particular page
- Observe the message on the screen

Test Data: No special preparation is required.

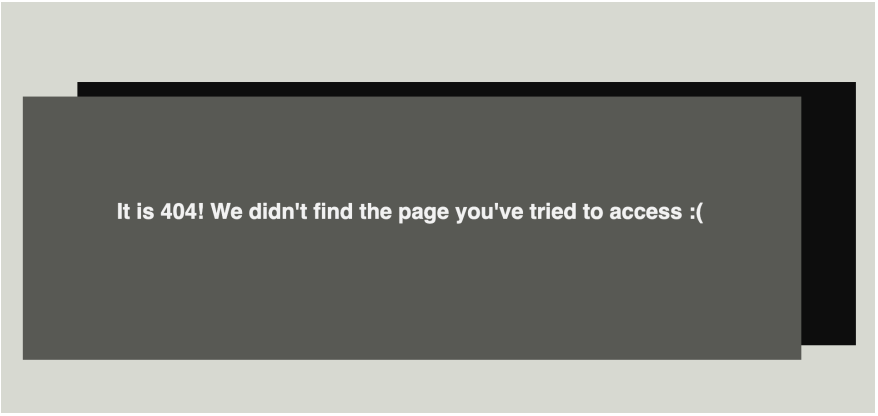
Expected Outcome: When I enter into the URL search bar: http://localhost:8080/loginfhgfg I should be redirected to a page that displays a message that tells that the page I attempted to visit simply does not exist.

Actual Outcome:

- URL entered:



- Page appeared on the screen:



Result: The test was passed.

Test Nº 17:

Purpose: To verify that users would be redirected to a special HTML error page when there is an internal server error.

Procedure:

- Trigger the system to produce an exception that is not elsewise handled in the program
- Observe what message appears on the screen.
- Observe what message is stored in the log files.

Test Data: I have added a special controller method into the AuthController class. Here is the code for that method:

```
@GetMapping("/error")
public String test() throws Exception {
    throw new Exception("Test error");
}
```

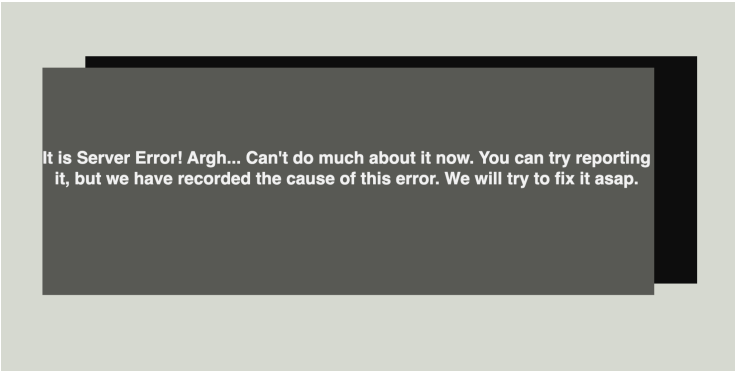
This method would trigger an exception with message: Test error
This method would be deleted after the test is finished executing.

Expected Outcome:
When a user enters localhost:8080/error an exception should be generated. The exception should be catched by the sampleExceptionHandler method inside the DefaultExceptionHandlerControlAdor class. The user should see a new page on their screen that tells them that an internal server error occurred. Simultaneously, a log message should be placed into the trace.log. The message should contain the message of the error and the full stack trace of the exception. The message should have a level of logging set to ERROR.

Actual Outcome:

- URL entered:

- Message on the user's screen



- Trace.log:

```
2022-02-05 16:54:09 ERROR
[com.adtimokhin.controller.DefaultExceptionHandlerControlAdvice.sampleExcep
tionHandler(DefaultExceptionHandlerControlAdvice.java:61)]: Internal server
error it was caused by test Exception.
Full stack
trace:[org.hibernate.internal.ExceptionConverterImpl.convert(ExceptionConve
rterImpl.java:167), … ]
```

The message is shortened because the full stack trace is 3 pages long.

Result:
The user sees the special error page, the appropriate log (with the correct time, logging level and message) is stored in the trace.log file. Hence, the test passed successfully.

TEST № 18:

Purpose: To validate that the website is protected from first-order SQL-injections.

Procedure:

- Enter into a form that will pass its inputs into a JPQL statement, data that would cause bookings table to be dropped.
- See if the table is dropped.

Test Data:
Data that would be inputted to login form:

Field	Value
email	adtimokhin@gmail.com; DROP TABLE Bookings
password	qwerty12345

Expected Outcome:
When I enter the test data into the login form, the user should not be found and the bookings table should not be dropped despite the fact that the query to the database would like this:

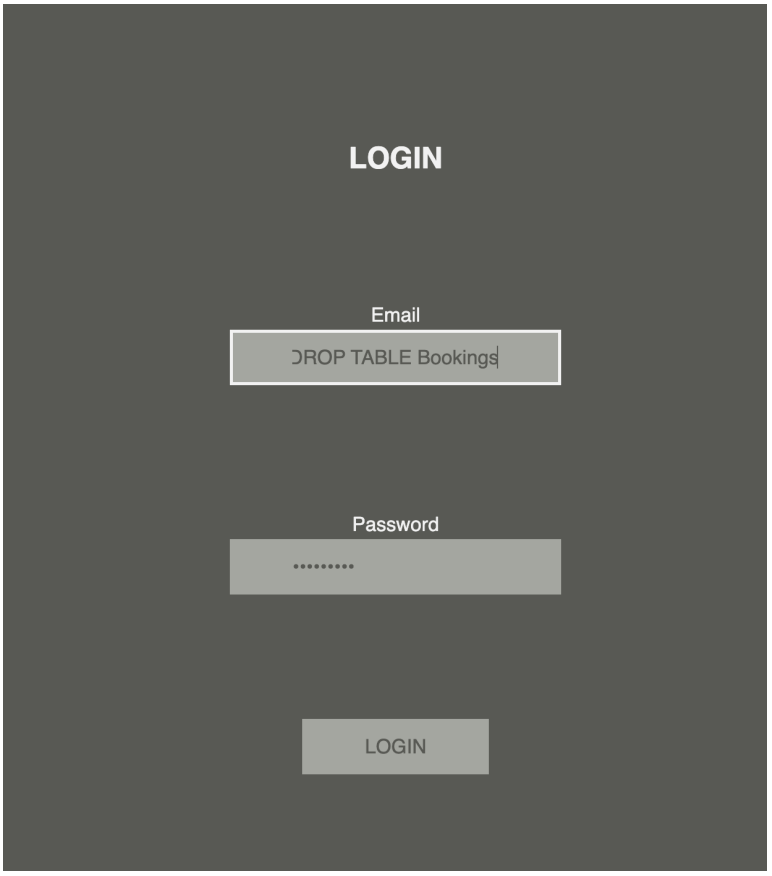
SELECT u FROM User u WHERE u.email = adtimokhin@gmail.com; DROP TABLE Booking

Note: Booking is one of the entities used in the application. It is linked to a table called booking_table. Therefore, the syntax above should drop the table booking_table.

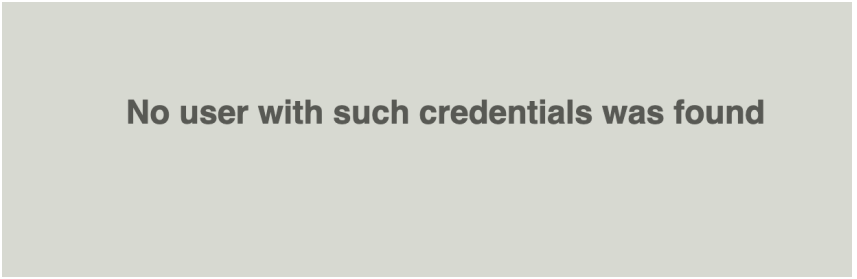
Actual Outcome:

- List of tables before any actions:

- The login form (filled in using the data from above):



- Message on the user's screen after pressing the login button:



- List of tables:

Result: The test was passed successfully.

EVALUATION SECTION

In my opinion, the project has been developed well. The tests conducted show that the main parts of the system work as expected.

Talking in general, the program manages to perform the main task - be an online platform where people can book time periods to use laundry machinery. I have performed a beta-test in my boarding house over a period of 3 days. The comments that I have received would be evaluated on page 210, but in general, the system managed to do well.

Evaluating the Success of the Solution

I would like to give a deeper evaluation of whether my objectives were met.

Objective	Met?	Comment
Users must be able to register on the platform	YES	Upon entering the index page the users are given an option to visit a sign up page. On the page, users must enter email, name, password and check-password. As shown by tests 2 and 5, appropriate error messages pop up if validations fail. An email is sent with the verification token to the email provided (refer to test 3). Therefore, this objective was fully met.
Users must be able to verify their email	YES	As shown by the test 4 - an algorithm that generates tokens works properly. The tokens are produced of the expected length and are unique. The results of test 3 prove that the tokens are saved in the database. The test 5 supports that users cannot access the booking functionality without verifying the email. By following the link the user is able to validate the email (test 6). Hence the objective is met.
Users must be able to login onto the platform in order to see private content.	YES	Users have access to the login button on the index page. This page contains email and password fields (as required). Upon the submission, the input is validated and the user sees an appropriate error message if the validation fails (test 5 shows that the user is not let onto the platform because of unverified email. That means that the test for their credentials was successful - the email verification is tested after the credentials being present in the database). The test 5 also proves that users receive error messages if the validation has failed. After validation the users are redirected to the actions/ page. As that page is only available to authenticated users (proven by test 1), the user authentication tokens are assigned to the users upon logging. All parts of the objective have been met.
Users must be able to check	YES	Proven by test 1, only authenticated users have access to the pages that display the timetables for washing and drying

washing and drying machine availability on any chosen day.		machines. Test 9 provides descriptive evidence that upon entering the timetable-viewing pages, the user can see the correctly constructed timetables for all machines of the right type and for the appropriate date. The same test shows that users have access to a separate page where users can select a date for timetables. Upon entering a new date the users are redirected back to the page with the timetables and the timetables are updated to show bookings for another date. As outlined in the design section, due to the fact that the user interface is designed so that they must select a date on a special page with an inbuilt date input form, the reason to additionally validate the format passed became redundant. All parts of the objective were
Users must be able to add a booking.	YES	As shown by tests 1 and 9, the authenticated user has access to a page that displays timetables for all washing or drying machines for a particular date. On that page the user has an option to select free slots. Under each timetable there is a submission button that sends the data inputted into the particular timetable for validation. Most of Tasks 5.2.* and 5.3.* became irrelevant as the decision to use checkboxes to select available slots eliminated any errors of users entering irrelevant data. Also, now users are simply unallowed to select slots that are booked by others. Test 10 has proven that the bookings are stored as expected when the valid data is inputted. Due to the design of the validation routine, all of the checks were performed for data validity, proving that the routine works properly. That test also has proven that the users get a message upon the submission of data. The bookings are also stored in the correct format. All the subtasks of the goal were met.
Users must be able to view their bookings.	YES	Test 11 proves that the users have access to a page that displays information about their bookings. The users are only shown those bookings that are still active (i.e. the end time of which has not yet passed). The objective is fully met.
Users must be able to cancel a booking	YES	Test 11 shows that users can press “cancel” buttons beneath their bookings on the “view my current bookings” page. After pressing the button, there is a check to see whether the user is still allowed to cancel the booking (users can cancel only if there is at least 120 minutes before the starting time of the booking). Test 11 shows how the booking is successfully cancelled. Upon the cancellation, the booking is removed from the database and appropriate log messages are generated and stored in the file called trace.log. After the deletion from the database, the user sees an appropriate message on the “my current bookings” page with an updated list of the current bookings.

Users must be able to report a misuse of the laundry machines.	YES	On a page that has the URL of http://localhost:8080/account/bookings there is a section that allows you to create reports and view existing ones. Test 15 proves that users have access to an HTML form that allows them to create a report. The same test shows that users' inputs (machine id) are validated and if there was a booking prior to their own the report would be created on that (previous) booking. The same test also validates that users receive a notice that tells whether the report was successfully generated. Hence, the objective 8 is fully met.
Users must be able to logout of the platform.	YES	There is a button in the account section of the website (URL http://localhost:8080/account/). It is labelled "Logout". After pressing this button the user is redirected to the index page. On the attempt to visit a page that is only available for authenticated users, the user is redirected to the login page. That means that the user loses their authentication token and thus is not authenticated. That behaviour is summarised in test 8. The test proves that all sub-objectives were achieved.
Users must be able to access relevant pages	YES	"view my bookings", "timetables", "main home"pages are only available to authenticated users and "login", "register", "index" are available to those who have not logged into the platform. Partially this is summarised in test 1.The settings for this behaviour are described in the SecurityConfig class.
Actions on the platform must be logged.	YES	Tests 12, 13, 16 ad 17 show that there are logging messages generated in the following cases: bookings are added and removed from the database; there is an internal server error; methods annotated with @DebugMethod would cause the system to produce messages before and their executions.
There should be appropriate exception handling	YES	As shown by tests 16 and 17, users are redirected to a special page when they attempt to enter a URL that is not linked to any controller method and are redirected to a separate page that informs the users that there happened an internal server error. Appropriate logging messages are recorded.
There must be some protection enabled.	YES	Every HTML form on the website contains a unique csrf token. As it can be seen from test 18, after a user entity is saved into the database their password is changed into a hash value. Thus, the passwords are encrypted. The test proves that on the attempt to login, the password passed in the login form is compared to the encrypted one and then the user is authenticated. Hence, the passwords are encrypted and are compared correctly. As described in the design section, SQL-injections are impossible due to the fact that JPA uses prepared statements and the users' input is treated not as a part of the query but

		rather a value. This is proven by the test 18
Actions must be asynchronous when possible.	YES	Methods that send emails and that place booking data into special text files are annotated with @Async annotations (they ensure that the methods are executed in separate threads).
Database content that is no longer relevant must be deleted automatically.	YES	Test 12 shows how these operations are conducted. At 22:01a routine is called that removes bookings that have date setted for today. All of the deleted bookings are stored in a special text file.
A database must be organised so that all data required is stored there.	YES	Code given in the design section is executed manually prior to launching the website. All tables necessary are generated (that can be seen on page 28)

User Feedback

I was able to run a beta test in my boarding house over weekends. Afterwards I was able to interview 3 people. Here is what they had to say:

How easy is the system to use?

- 'The website was easy to navigate. The home page for logged in users was very handy - all actions are in reach of a button! However, I found that on some pages the navigation buttons were missing. For example, there was no "go back" button when I ended up on the "this page does not exist" page. Apart from that, navigation on the website is highly intuitive.'

- ' Tables that contain available slots made a lot of sense. It did not require any effort to understand how to use the system. However, I found that some other actions were less obvious. I had to contact you personally for you (Aleksandr Timokhin) to tell me how to use the "report" feature of the website.'

- 'I really appreciated the fact that I can view bookings that I have made. It makes it so much easier to keep a track of when I should be using the machines. I think it would be even better if I got some kind of notification that my booking time is approaching.'

Did you manage to do on the website everything you hoped the system would do?

- 'Yes. The system does the most essential thing - it allows you to place bookings! I would say that there was a feature that did not work that well for me. After I left a complaint nobody from the staff showed up. I had to call them myself.'

- ‘It felt funny that you can enter almost any name you like. No one really checks that. Also, I can very easily set up two accounts for myself using different email addresses.’

- ‘The system worked just fine for me.’

Any suggestions?

- ‘The system is definitely helpful. But I would really appreciate it if it was made phone-friendly. When I try to access the website through my phone the interface is almost unusable - all buttons and other components are uncomfortably small.’

Analysing Users’ Commentaries:

The feedback received was almost positive. That made me confident that most of the features I have implemented are useful and work appropriately not only on my device.

Prior to running the beta-test I have warned everyone that the version of the program that we would be testing is very far from final. I consider that to be an important factor that decreased the volume of negative comments, limiting them to what is really bothering.

For me, the main take-aways from the interviews were:

- Placing/Cancelling bookings feature works well. That is promising as it is the main feature of the whole project.

- The Reporting system is not helpful at the state it is currently. First problem is that it is unintuitive. Users are unsure how it works. Second problem is that the reports that were created are not viewed by the staff members, making them useless.

- The interface can be improved to make it even more user-friendly. It was unfair to assume that real users would be using a laptop to access the portal. I was using the laptop to both create and to test my project as it saved time on switching between the devices. But the feedback I received clearly suggests that people do not face this issue and thus they require a well-developed interface for smaller screens.

- The comment that anyone can create as many accounts as they like was enlightening. The checks that I have put into place when users are registering prevent them from creating accounts with unexisting emails. But that might not be the issue.

- The design of the home screen for authenticated users is so simplistic that it is helpful to navigate through the website.

- Users do not enter URLs into the search bar to navigate through the website. While developing I was using this technique because I knew all possible URLs that I should be able to enter. It would be helpful to add navigation buttons that would take people to the home screen.

From this information I came up with some new potential improvements for the system.

Potential Improvements

Reporting system can be upgraded to be more helpful. The system may be upgraded so that When a new report is placed, staff members would get a notification through a letter sent to their emails. The system has the capability to be upgraded to perform the actions of sending emails to staff. However, for that purpose, a new feature must be implemented - authorities on the website. The system must distinguish between the staff and students.

Another upgrade can be a tutorial to the website that would be displayed to newly authorised users. A new table can be created that would store ids of those users that have not yet logged into the platform. And if they log in for the first time, their ids would be removed from that table and they would be redirected to another page with a text-tutorial.

I can also change the mechanism for setting up the reports. On the page where people need to specify the machine that they should be using but can't, I can add timetables for these machines for which a user has made bookings (for the current day). It would make it more visually intuitive how to use the report system.

Another improvement might be adding a third state for the time periods - passed. It would make it simpler for people to see which time periods are not available for the booking.

As it was discussed, my real users use phones to access the platform. Though I have used a CSS framework (Bootstrap 5) that I have configured to adapt to the phone screens, I probably did not do it well as people were complaining. An obvious improvement is to set the framework correctly and configure a new arrangement for elements on small screens. Another option, however, would also take care of the problem of notifications. That is creating IOS and Android applications. The issue with this option is that I would need to create two brand new applications.

The system that I have been using to validate users when they create their accounts does not work for the system where each account has a limited number of bookings per week - people are tempted to create new accounts. The solution to this is "closed registration". Before the system is launched at any new site, the people in charge should collect a list of names and valid email addresses. Users would no longer have access to the "sign up" option. When the system goes live emails with the credentials would be sent to all in the lists.

Another issue I see is that people have no way of changing their passwords. If someone wishes to change it they should have that option. That is not to say that currently there is no way for people to restore their accounts if they forget their passwords. An appropriate solution would be to create a system that sends validation tokens to the users to allow them to grant an access to a page where they can swap their passwords.

Another problem I see is that if people do not use their accounts, or simply never log in into them, it is not an efficient use of the memory in the database. Possible solution might be creating a table that stores user_id and a timestamp when the account was created or used for the last time. If two month pass since usage of the account the account might be deleted (with prior notification via email). However, this solution sounds unethical to me.

There are some improvements that would help the system to become more efficient.

One of such is creating a proper front-end, using a language such as JavaScript. Sending all data to users' devices in JSONs and then interpreting the data using JavaScript would decrease the load on the server. But that would require redeveloping the system into a REST API.

Using JPQL, Hibernate and JPA repositories is sensible and useful for the application of such sizes. As the complexity grows and the number of active users rises, however, usage of pure SQL becomes more appropriate. The reason being that JPQL, JPA, HQL all require to be translated to SQL.

Another possible improvement might include using a non-relational database, like mongoDB. File-based databases store data in JSONs and the structure of the tables is undefined. That means that the "columns" of data stored about a single entity can vary from another object in the same table. That might be helpful as the project grows to allow storing data of some unpredicted formats without restructuring the whole database.

Bibliography

Apache FreeMarker Manual. 2021. *macro, nested, return*. [online] Available at: <https://freemarker.apache.org/docs/ref_directive_macro.html> [Accessed 21 May 2021].

Baeldung.com. 2021. [online] Available at: <https://www.baeldung.com/hibernate-one-to-many> [Accessed 6 May 2021].

Bodnar, J., 2020. *Spring @DeleteMapping tutorial - using @DeleteMapping to map DELETE requests onto handlers*. [online] Zetcode.com. Available at: <https://zetcode.com/spring/deletemapping/> [Accessed 3 June 2021].

Blinov and Romanchik, 2007. *Java Industrial Programming*. [ebook] Minsk: UniversalPress, pp.1-704. Available at: <https://careers.epam.by/content/dam/epam/by/book_epam_by/Java.Programming.pdf> [Accessed 3 June 2021].

Cockroachlabs.com. 2021. *SQL Performance Best Practices | CockroachDB Docs*. [online] Available at: <https://www.cockroachlabs.com/docs/v20.2/performance-best-practices-overview.html> [Accessed 8 July 2021].

Comptia.org. 2021. *Tech Hiring Stands Out in December Jobs Report*. [online] Available at: <https://www.comptia.org/newsroom/press-releases/2021/01/08/tech-hiring-stands-out-in-december-jobs-report> [Accessed 28 October 2020].

DB-Engines. 2021. *DB-Engines Ranking*. [online] Available at: <https://db-engines.com/en/ranking> [Accessed 8 July 2021].

Docs.jboss.org. 2004. *Chapter 7. Collection mapping*. [online] Available at: <https://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/collections.html> [Accessed 14 July 2021].

Duarte, A., 2020. *Understanding Inversion of Control and Dependency Injection - DZone Java*. [online] dzone.com. Available at: <https://dzone.com/articles/understanding-inversion-of-control-and-dependency> [Accessed 19 June 2021].

Fiverr.com. (2010). *Freelance Services Marketplace and Businesses*. [online] Available at: <https://www.fiverr.com/> [Accessed 9 September 2021].

Gb.ru. *GeekBrains - Educational portal*. [online] Available at: <https://gb.ru/> [Accessed 28 October 2020].

GitHub. 2021. *GitHub - wmff/topjava23: Java Enterprise: Maven/ Spring/ Security/ JPA(Hibernate)/ REST(Jackson)/ Bootstrap(CSS)/ jQuery*. [online] Available at: <https://github.com/wmff/topjava23> [Accessed 15 July 2021].

Haase, M. and Wakefield, S., 2011. *Why are there no PUT and DELETE methods on HTML forms?*. [online] Software Engineering Stack Exchange. Available at: <https://softwareengineering.stackexchange.com/questions/114156/why-are-there-no-put-and-delete-methods-on-html-forms> [Accessed 18 June 2021].

Habr. 2017. *Spring MVC — Basic Principles*. [online] Available at: <https://habr.com/ru/post/336816/> [Accessed 23 June 2021].

Hexlet.io. n.d. *Module 2. Lesson 4. Packets in Java. / Introduction in Java*. [online] Available at: <https://ru.hexlet.io/courses/java_101/lessons/java_packages/theory_unit> [Accessed 3 June 2021].

Hibernate. n.d. *Hibernate. Everything data. - Hibernate*. [online] Available at: <https://hibernate.org/> [Accessed 28 May 2021].

Htmlbook.ru. (2002). *Home | htmlbook.ru*. [online] Available at: <http://htmlbook.ru/> [Accessed 13 September 2021].

Indeed.com. n.d. *Java With Spring Aop jobs*. [online] Available at: <https://www.indeed.com/jobs?q=Java+With+Spring+Aop&vjk=882c3cb20a1ecc8f> [Accessed 5 December 2021].

Itjobswatch.co.uk. 2021. *Spring Jobs, Skill Sets & Salary Benchmarking*. [online] Available at: <https://www.itjobswatch.co.uk/jobs/uk/spring.do#job_vacancy_trend> [Accessed 5 December 2021].

Javarush.ru. n.d. [online] Available at: <https://javarush.ru/> [Accessed 3 June 2021].

Java For Beginners. Lesson 1: JDK и Hello World., [online video], alishev, Published 6 January 2015, <https://www.youtube.com/watch?v=ziOQ8wkmnSE&list=PLAma_mKffTOSUkXp26rgdnC0PicnmnDak>, accessed [20 November 2020].

Journaldev.com. 2014. *Spring MVC Exception Handling - @ControllerAdvice, @ExceptionHandler, HandlerExceptionResolver - JournalDev*. [online] Available at: <https://www.journaldev.com/2651/spring-mvc-exception-handling-controlleradvice-exceptionhandler-handlerexceptionresolver> [Accessed 14 June 2021].

Kainulainen, P., 2012. *Spring Data JPA Tutorial Part Three: Custom Queries with Query Methods*. [online] Petrikainulainen.net. Available at: <https://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-three-custom-queries-with-query-methods/> [Accessed 9 August 2021].

King, P., n.d. *Apache Maven WAR Plugin – Usage*. [online] Maven.apache.org. Available at: <https://maven.apache.org/plugins/maven-war-plugin/usage.html> [Accessed 12 August 2021].

Logicbig.com. 2017. *JPA + Hibernate - Persisting a Collection of basic Java types by using @ElementCollection*. [online] Available at: <https://www.logicbig.com/tutorials/java-ee-tutorial/jpa/element-collection.html> [Accessed 9 July 2021].

Mihalcea, V., 2020. *A beginner's guide to entity state transitions with JPA and Hibernate - Vlad Mihalcea*. [online] vladmihalcea.com. Available at: <https://vladmihalcea.com/a-beginners-guide-to-jpa-hibernate-entity-state-transitions/> [Accessed 9 July 2021].

Minto, B., 1991. *The pyramid principle*. London: Minto International.

Patel, V., 2013. *Spring MVC Cookie example. Spring Http Cookie Tutorial*. [online] ViralPatel.net. Available at: <https://www.viralpatel.net/spring-mvc-cookie-example/#:~:text=Spring%20does%20not%20provide%20any,it's%20just%20piece%20of%20cake> [Accessed 9 August 2021].

Poutsma, A., Deleuze, S. and Clozel, B., n.d. *HttpStatus (Spring Framework 5.3.13 API)*. [online] Docs.spring.io. Available at: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpStatus.html> [Accessed 23 August 2021].

Proselyte.net. 2021. *App examples*. [online]. *PROSELYTE*.Available at: <https://proselyte.net/video/applications/> [Accessed 10 November 2020].

Proselyte.com, 2016. Spring Handbook (complete course). [online] *.PROSELYTE*. Available at: <https://proselyte.net/tutorials/spring-tutorial-full-version/> [Accessed 10 January 2021].

Sierra, K. and Bates, B., 2008. *Head First Java*. 2nd ed. Sebastopol: O'Reilly Media, Inc.

SINGH, R., 2017. *JPA / Hibernate ElementCollection Example with Spring Boot*. [online] CalliCoder. Available at: <https://www.callicoder.com/hibernate-spring-boot-jpa-element-collection-demo/> [Accessed 21 May 2021].

StackShare. 2021. *Why developers like Java*. [online] Available at: <https://stackshare.io/java> [Accessed 6 May 2021].

Tiobe.com 2021, index | TIOBE - The Software Quality Company [online] Available at: <https://www.tiobe.com/tiobe-index/> [Accessed 3 June 2021].

Traversy Media (2017). *Traversy Media*. *YouTube*. Available at: <https://www.youtube.com/watch?v=5GcQtLDGXy8> [Accessed 18 Aug. 2021].

Writingcenter.uagc.edu. n.d. *Writing Clearly & Concisely | UAGC Writing Center*. [online] Available at: <https://writingcenter.uagc.edu/writing-clearly-concisely> [Accessed 3 June 2021].

W3schools.com. n.d. *HTML input value Attribute*. [online] Available at: <https://www.w3schools.com/tags/att_input_value.asp> [Accessed 8 July 2021].

Youtube.com. *doIT - YouTube*. [online] Available at: <https://www.youtube.com/channel/UCwMl9L2VNAR0upPrkhAo_Ig> [Accessed 20 November 2020].

Youtube.com. *Amigoscode - YouTube*. [online] Available at: <https://www.youtube.com/user/djdjalas> [Accessed 3 December 2020].

Youtube.com. *alishev - YouTube*. [online] Available at: <https://www.youtube.com/c/alishevN> [Accessed 3 December 2020].

Youtube.com. 2021. *Spring Boot Full Stack with Angular | Full Course [2021] [NEW]*. [online] Available at: <https://www.youtube.com/watch?v=Gx4iBLKLVHk&t=1s> [Accessed 3 December 2020].

Youtube.com. 2007. *JavaBrainsChannel - Youtube*. [online] Available at: <https://www.youtube.com/c/JavaBrainsChannel/playlists> [Accessed 3 June 2021].

Youtube.com. n.d. *Part 04 - 07 - Custom Error Page Global Exception Handling Spring Mvc*. [online] Available at: <https://www.youtube.com/watch?v=SsOEmclaJ3A> [Accessed 7 July 2021].

Youtube.com. n.d. *IntelliJ IDEA. Adding a Project to Git and GitHub*. [online] Available at: <https://www.youtube.com/watch?v=mf2-MOl0VXY> [Accessed 11 August 2021].

Youtube.com. n.d. *How to use Tomcat server in IntelliJ IDEA*. [online] Available at: <https://www.youtube.com/watch?v=29Xp3V3Ip08> [Accessed 12 August 2021].

Youtube.com. 2020. *Bootstrap 5 tutorial - crash course for beginners in 1.5H*. [online] Available at: <https://www.youtube.com/watch?v=c9B4TPnak1A> [Accessed 13 October 2021].

Youtube.com. 2020. *Javascript Full Course for Beginners to Advanced*. [online] Available at: <https://www.youtube.com/watch?v=dOnAC2Rr-6A> [Accessed 10 October 2021].