#Oblig 2

####Nicolai Molstad, Emilie Martin og Erik Myhre

Implementing task 1

We started with implenting the API. Where we add all the routes for:

- Users
- Chat-rooms
- Room users
- Messages

As an example we will show one of them, as they are quite similar.

```java
package org.emnmem.oblig2.controller;

import ...


@CrossOrigin
@RestController
@RequestMapping("/api/users")
public class UsersController {

    Logger logger = LoggerFactory.getLogger(UsersController.class);

    @Autowired
    UserService userService;

    @PostMapping("/addOne")
    public ResponseEntity<User> addUser(@RequestBody UserDto user) {
        logger.info(user.toString());
        if (user.getUsername().equals("") || user.getPassword().equals("")){
            return new ResponseEntity<>(null, HttpStatus.BAD_REQUEST);
        }
        User userResponse = userService.addOne(user);
        return new ResponseEntity<>(userResponse, HttpStatus.OK);
    }

    @PostMapping("/getByUser")
    public ResponseEntity<User> getByUsername(@RequestBody User user) {
        var userOut : Optional<User> = userService.getByUsername(user);
        return userOut.map(value -> new ResponseEntity<>(value, HttpStatus.OK)).orElseGet(()
                -> new ResponseEntity<>(null, HttpStatus.BAD_REQUEST));

    }

    @GetMapping("/getAll")
    public ResponseEntity<List<User>> getAll() { return new ResponseEntity<>(userService.getAll(), HttpStatus.OK); }
}

```

@RequestMapping is routing all calls for api/users to this controller. @AutoWired is dependency injecting the service for the user table in the database. @PostMapping is defines an endpoint for putting data into the API. @GetMapping is defining an endpoint for getting the data from the API.

We of course also had to have the files from the folder model, service and repository as well. In model we have just made the objects:

- Message
- Room
- RoomUser
- RoomUserId
- User

Again for the example I will only show a picture of UserService.

```java
@Service
@Transactional
public class UserService {

    @Autowired
    UserRepository userRepository;

    public User addOne(UserDto user) {
        String password = BCrypt.hashpw(user.getPassword(), BCrypt.gensalt( log_rounds: 4));
        User setUser = new User();
        setUser.setUsername(user.getUsername());
        setUser.setPassword(password);

        return userRepository.save(setUser);
    }

    public List<User> getAll() { return userRepository.findAll(); }

    public User getById(String id) {
        try {
            Optional<User> user = userRepository.findById(Integer.parseInt(id));
            return user.orElse( other: null);
        } catch (Exception e) {
            return null;
        }
    }

    public Optional<User> getByUsername(User user) {
        return userRepository.findAll()
            .stream()
            .filter(user1 -> (user.getUsername().equals(user1.getUsername()) &&
                        BCrypt.checkpw(user.getPassword(), user1.getPassword()))).findFirst();



    }

    public User deleteById(String id) {
        User result = getById(id);
        userRepository.deleteById(Integer.parseInt(id));
        return result;
```

@Transactional guarantees serializing of the database.

For the Repository.

```java
package org.emnmem.oblig2.repository;

import ...

@Repository
public interface UserRepository extends JpaRepository<User, Integer> {

    User save(UserDto user);
}

```

This is the connection to the database. The JPARepository interface has predefined methods for connecting to the database.

Implementing Task 2

We implemented the "client program" via the *oblig 2* folder. The *oblig 2* folder contains the entirety of the frontend part of the project.

To implement this part we used ReactJS. The entry point for the application is App.js. We have made the components for the

- Frontpage
- Login
- Register
- Rooms
- Room/Id

The site has the basic functions you would excpect from the names of the components.

We then moved on to the bots. ClientBot and WordAnalysing are the files we made to implement this. WordAnalysing is heavily inspired from portfolio 1 and thus makes the bots just a simple replybot. In Clientbot we have made 4 bots with different names and using the WordAnalysing to each create their own unique replies.

Implementing task 3

To do this we figured we needed websockets.

```java
package org.emnmem.oblig2.websocket;

import ...

@Controller
public class WebsocketController {

    @MessageMapping("/chat.send")
    @SendTo("/topic/public")
    public PushDto sendMessage(@Payload PushDto room) { return room; }

}
```

The chatting room connects to the websocket. The room subscribes to the /topic/public. For each time the user sends their message; it sends the room number to the websocket. All users in the same room then fetches the messages from the API.

Running

Running the program requires the dependencies in the pom.xml file. Maven uses these files to run and build the program.

The dependencies used in backend are:

- Springboot (server)
- Lombok (making constructors and getters and setters)
- h2database (database)
- bcrypt (hashing the password)

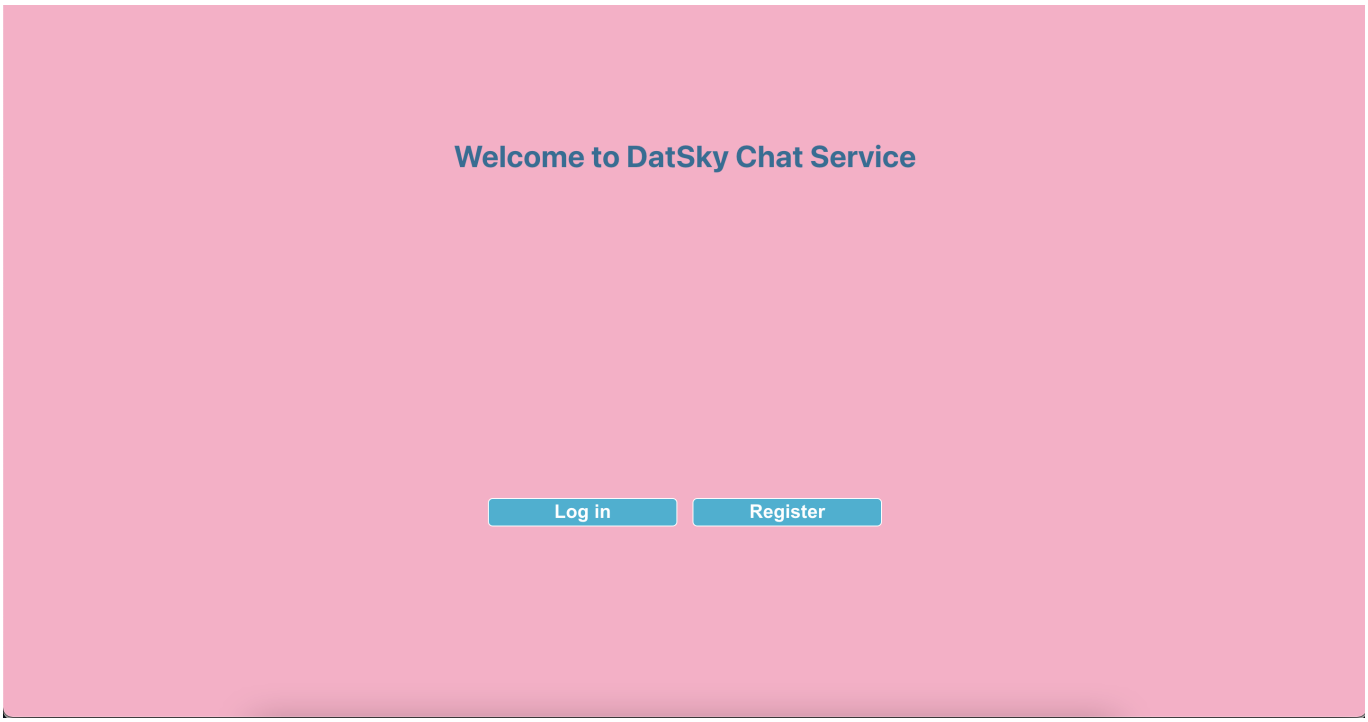
The dependencies used in frontend are:

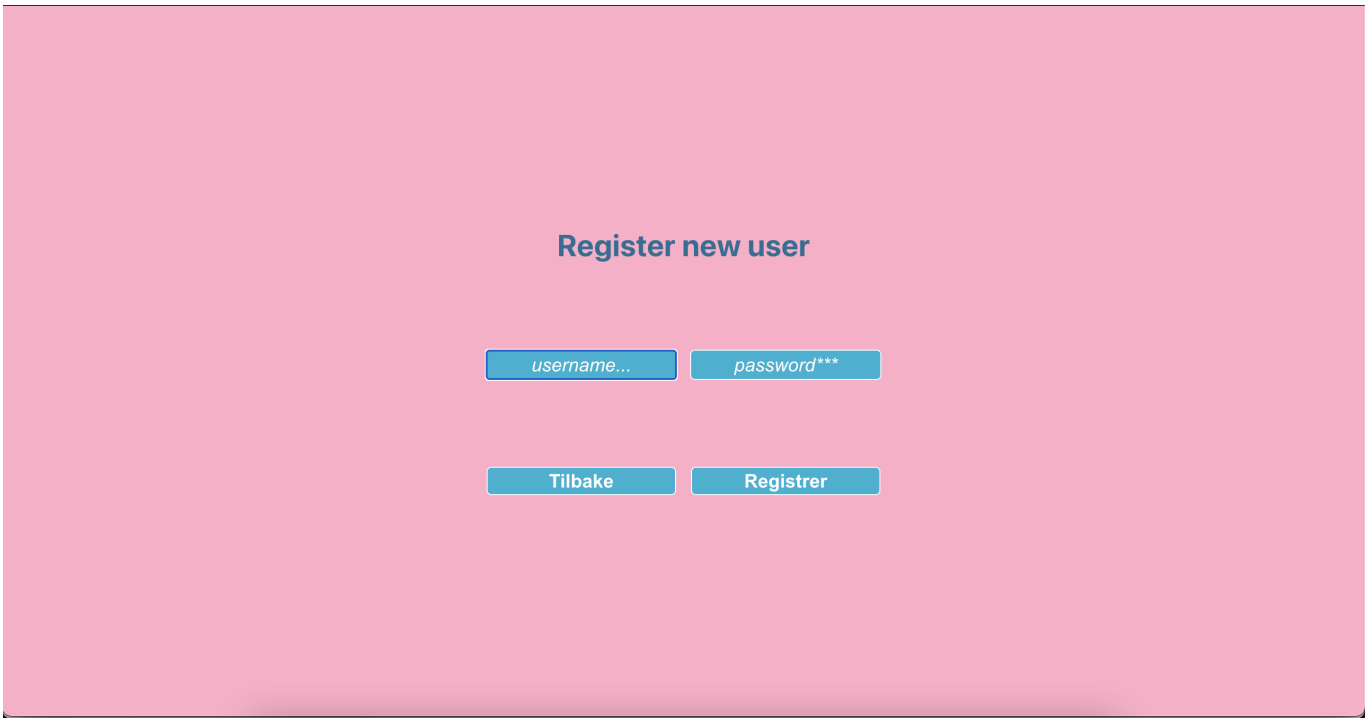
- StompJS (connecting to the websocket)
- SockJS-Client (connecting to websocket)
- React-router-dom (redirecting)

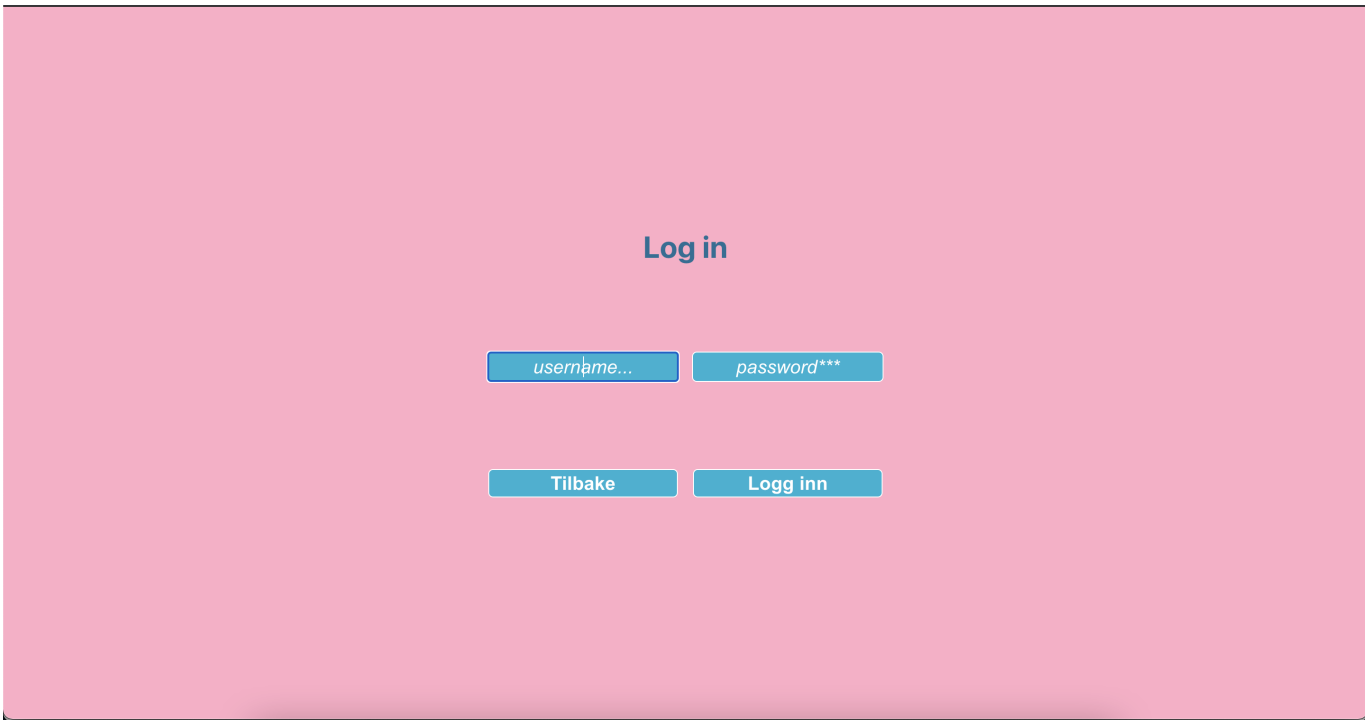
To build and run the project we are using maven to create a .jar. The command we are using is mvn clean install. Java runs the .jar with the command java -jar *.jar. You will need to have maven installed to do this. We have premade the .jar file. We have made the docker file to make an image, and in the start.sh script you can automatically buid the image and run a container based on that image

Outputs

Outputs from browsers


Welcome to DatSky Chat Service
Log in
Register


Register new user
username...
password***
Tilbake
Registrer


Log in
username...
password***
Tilbake
Logg inn


Room name
Register
Log out
All rooms
Eriks fantastiske rom


Register to Eriks fantastiske rom
Tilbake
Register

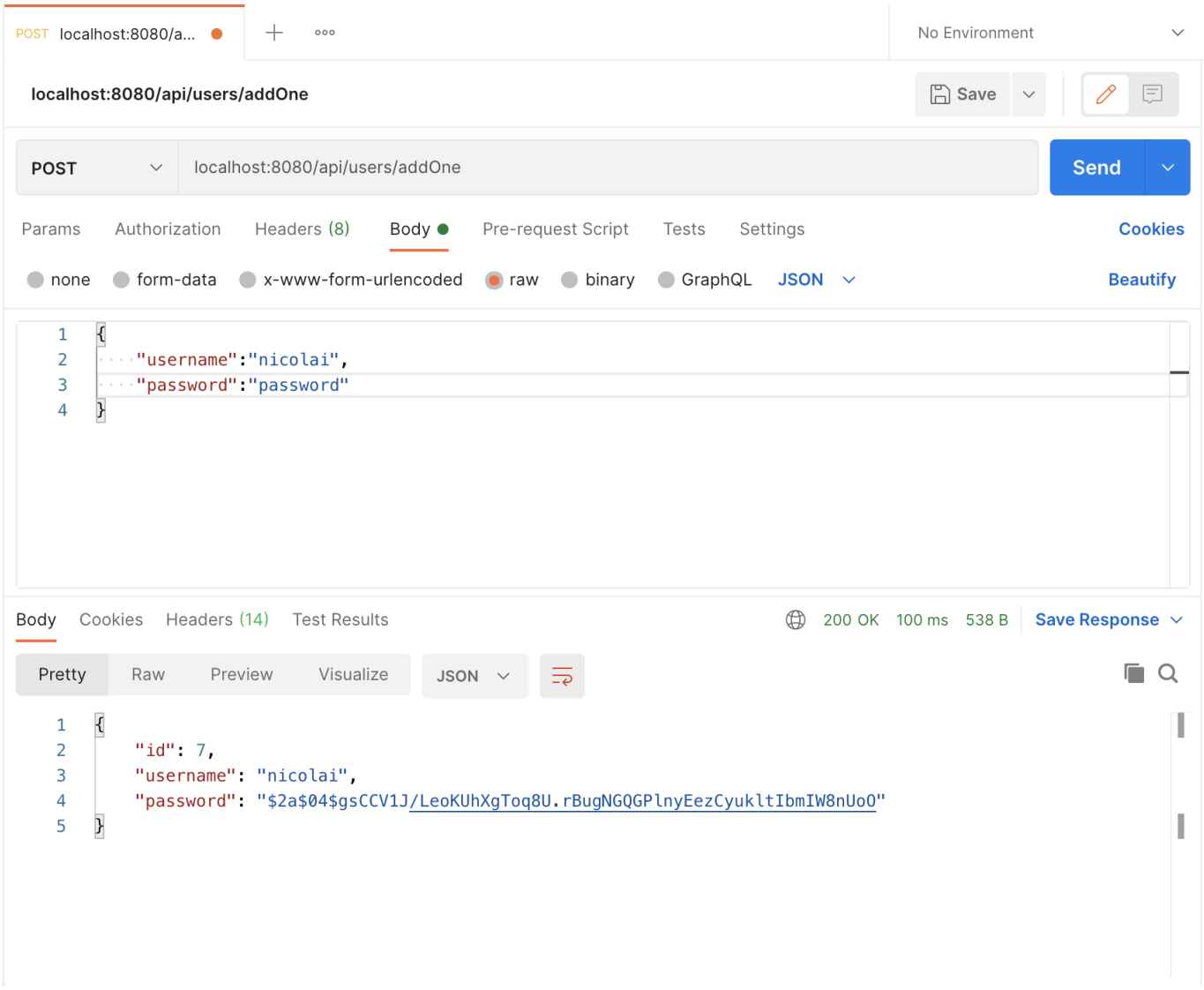

Eriks fantastiske rom
Tilbake
message
👆

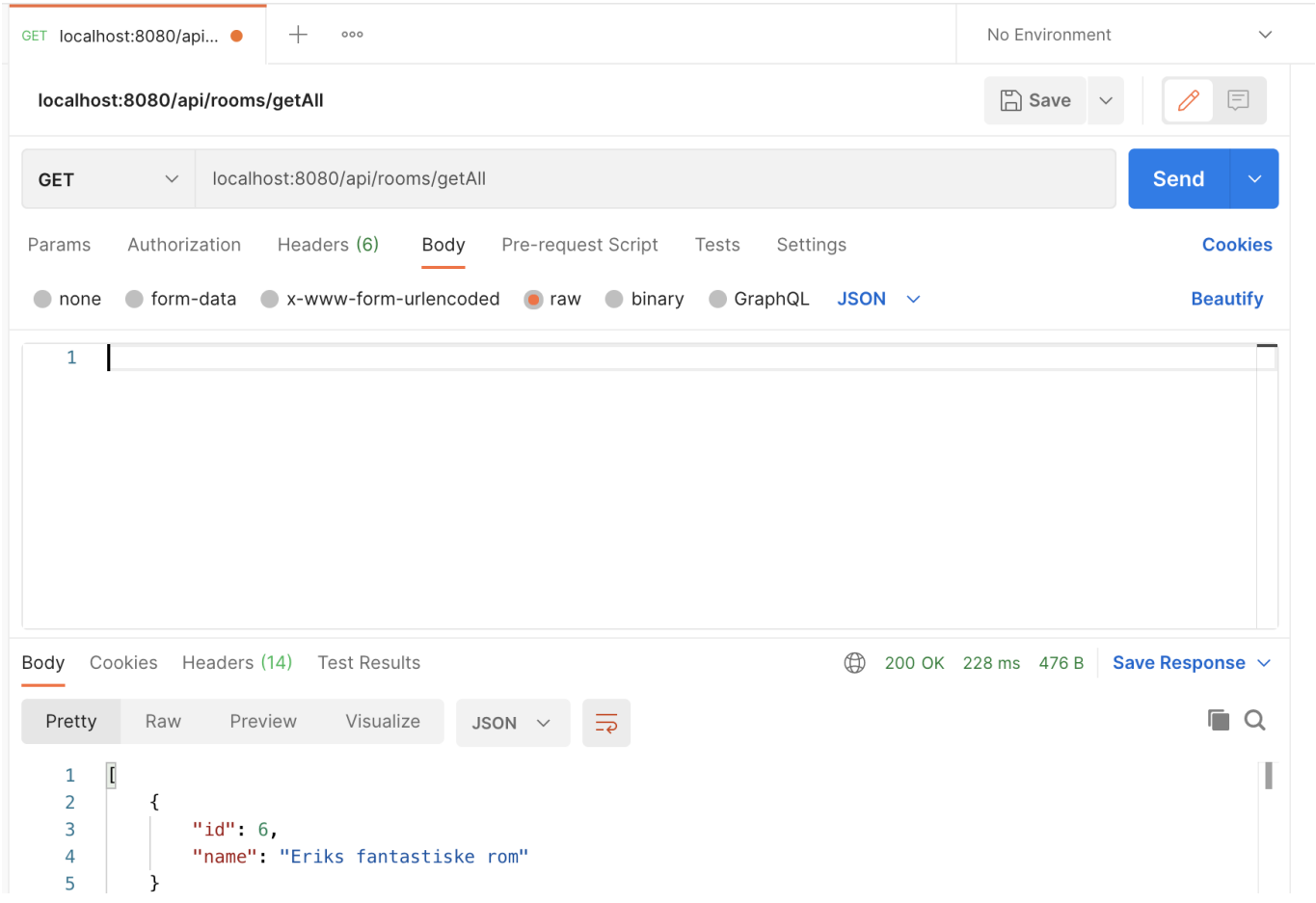
Outputs from console

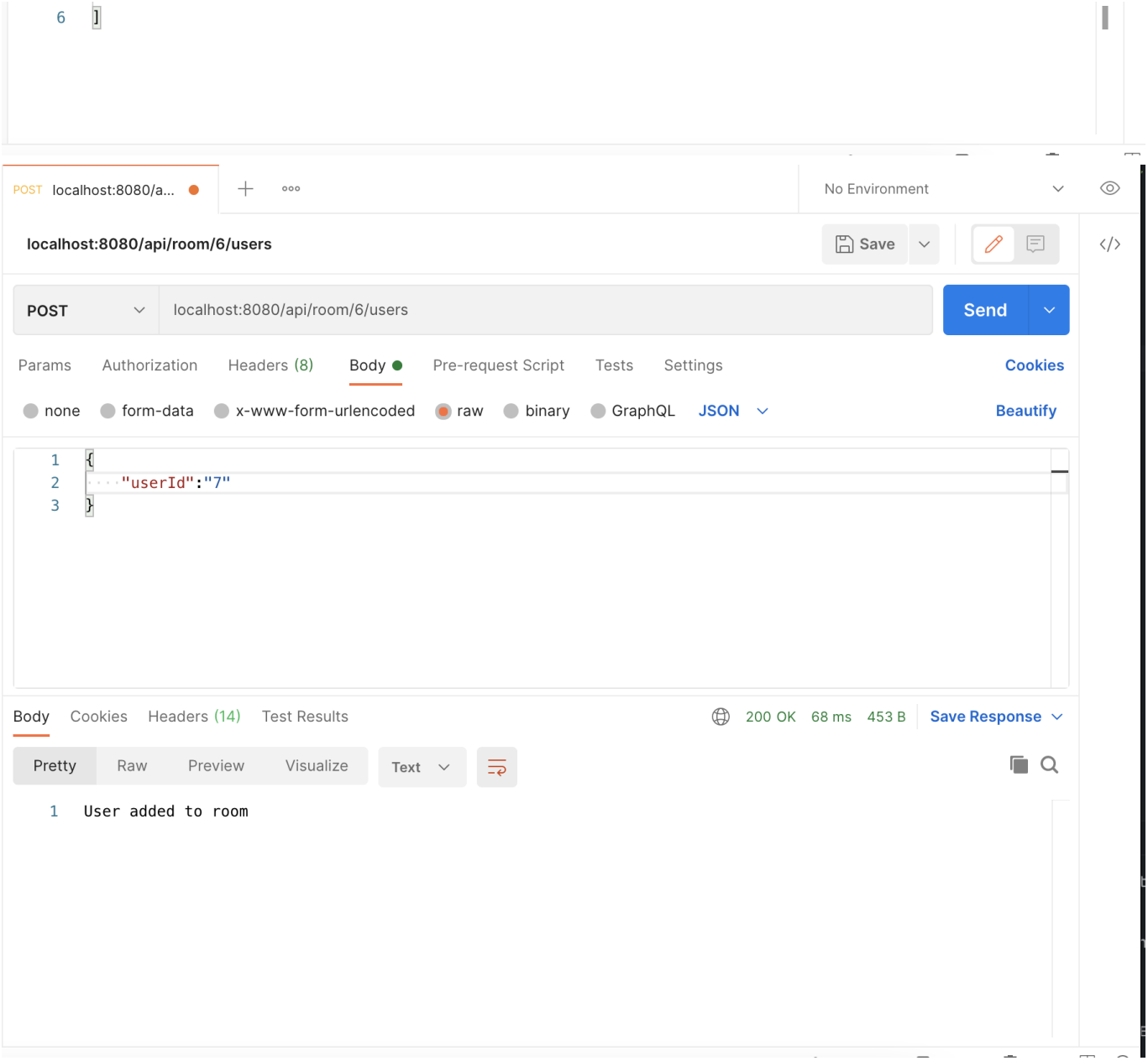
```
/Library/Java/JavaVirtualMachines/jdk-11.0.7.jdk/Contents/Home/bin/java -Dfile.encoding=UTF-8 -jar /Users/nicolaimolstad/OneDrive - OsloMet/DATA 2410 - Datanettverk og skytjenester/oblig2/tc

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.4.3)

2021-05-05 13:23:01.561  INFO 43111 --- [           main] org.emnmem.oblig2.Oblig2Application      : Starting Oblig2Application v0.0.1-SNAPSHOT using Java 11.0.7 on MacBook-Pro with PID 4311
2021-05-05 13:23:01.564  INFO 43111 --- [           main] org.emnmem.oblig2.Oblig2Application      : No active profile set, falling back to default profiles: default
2021-05-05 13:23:02.780  INFO 43111 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-05-05 13:23:03.115  INFO 43111 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 323 ms. Found 4 JPA repository interfaces.
2021-05-05 13:23:03.982  INFO 43111 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2021-05-05 13:23:04.002  INFO 43111 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-05-05 13:23:04.002  INFO 43111 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.43]
2021-05-05 13:23:04.079  INFO 43111 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-05-05 13:23:04.079  INFO 43111 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2441 ms
2021-05-05 13:23:04.148  INFO 43111 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2021-05-05 13:23:04.462  INFO 43111 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2021-05-05 13:23:04.529  INFO 43111 --- [           main] o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:testdb'
2021-05-05 13:23:04.721  INFO 43111 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2021-05-05 13:23:04.796  INFO 43111 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.28.Final
2021-05-05 13:23:05.000  INFO 43111 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2021-05-05 13:23:05.174  INFO 43111 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2021-05-05 13:23:05.931  INFO 43111 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platfc
2021-05-05 13:23:05.944  INFO 43111 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2021-05-05 13:23:06.269  WARN 43111 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be perform
```

```
2021-05-05 13:23:07.045  INFO 43111 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'clientInboundChannelExecutor'
2021-05-05 13:23:07.051  INFO 43111 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'clientOutboundChannelExecutor'
2021-05-05 13:23:07.064  INFO 43111 --- [           main] o.s.s.c.ThreadPoolTaskScheduler          : Initializing ExecutorService 'messageBrokerTaskScheduler'
2021-05-05 13:23:07.182  INFO 43111 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'brokerChannelExecutor'
2021-05-05 13:23:07.579  INFO 43111 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframework.security.web.context.request.async.WebA
2021-05-05 13:23:07.805  INFO 43111 --- [           main] o.s.b.a.w.s.WelcomePageHandlerMapping    : Adding welcome page: class path resource [static/index.html]
2021-05-05 13:23:08.037  INFO 43111 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2021-05-05 13:23:08.038  INFO 43111 --- [           main] o.s.m.s.b.SimpleBrokerMessageHandler     : Starting...
2021-05-05 13:23:08.038  INFO 43111 --- [           main] o.s.m.s.b.SimpleBrokerMessageHandler     : BrokerAvailabilityEvent[available=true, SimpleBrokerMessageHandler [org.springframework.m
2021-05-05 13:23:08.038  INFO 43111 --- [           main] o.s.m.s.b.SimpleBrokerMessageHandler     : Started.
2021-05-05 13:23:08.059  INFO 43111 --- [           main] org.emnmem.oblig2.Oblig2Application      : Started Oblig2Application in 7.38 seconds (JVM running for 8.242)
```

Outputs from API


POST localhost:8080/a...
No Environment
localhost:8080/api/users/addOne
Save
POST
localhost:8080/api/users/addOne
Send
Params Authorization Headers (8) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
{
"username":"nicolai",
"password":"password"
}
Body Cookies Headers (14) Test Results
200 OK 100 ms 538 B
Save Response
Pretty Raw Preview Visualize JSON
{
"id": 7,
"username": "nicolai",
"password": "$2a$04$gsCCV1J/LeoKUhXgToq8U.rBugNGQGPlnyEezCyukltIbmIW8nUoO"
}


GET localhost:8080/api...
No Environment
localhost:8080/api/rooms/getAll
Save
GET
localhost:8080/api/rooms/getAll
Send
Params Authorization Headers (6) Body Pre-request Script Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (14) Test Results
200 OK 228 ms 476 B
Save Response
Pretty Raw Preview Visualize JSON
[
{
"id": 6,
"name": "Eriks fantastiske rom"
}



Other stuff

After we implemented all the most important features we added some quality of life improvements to the look and use of the site using CSS and javascript. An example of this would be being able to just click enter from having written your password to automatically click the login/register button.