Software engineering lab FS22 - Group 21

Milestone 1 – Report

Jordi Küffer (Teamlead) – 20-714-051
Jasmin Hochuli – 20-705-711
Luisa Stückelberger – 20-711-412
Christoph Bachmann – 20-701-090
Lars Bösch – 18-921-981

Tests

Unit-Test

```
import { UserValidator } from '../main/validation/UserValidator'

function generatePW(length: number): string {
  let pw = "";
  for ( let i = 0; i < length; i++ ) {
      pw += "1"
  }
  return pw;
}


describe( name: 'Validate PostUser', fn: () => {

  let invalidDateRange = {
    "password": generatePW( length: 6),
    "firstName": "test",
    "lastName": "test",
    "description": "test",
    "biography": "test",
    "tags": "test",
    "pictureReferences": ["test"],
    "birthday": "2019-06-22",
    "email": "test@test.ch",
    "phoneNumber": "+41795233087",
    "gender": "MALE",
    "moveInDate": "1999-06-23",
    "moveOutDate": "1999-06-22"
  }
  test( name: 'test invalid date range', fn: () => {
    let res = UserValidator.validatePostUser(invalidDateRange);
    let expected = "Errors:\nmoveInDate must be before moveOutDate\n" +
        "Mandatory fields are: firstName,lastName,birthday,email,phoneNumber,password\n" +
        "Optional fields are: description,biography,tags,pictureReferences,gender,moveInDate,moveOutDate,isComplete"
    expect(res.validationFoundErrors()).toBe( expected: true);
    expect(res.toString()).toEqual(expected);
  })
```

Description:

This Unit-Test tests the implementation of the UserValidator class. First, we defined a json body where the moveInDate is after the moveOutDate. This will then be passed to the static UserValidator.validatePostUser() method, which returns an instance of the ValidationReport class (This class is tested separately by a different unit test). The validation report is then saved in the ‘res’ variable.
After that, two assertions assert that the report does indeed contain errors and that the error is the expected one (by comparing it to a predefined string).

This test is easy to read and understand since it uses descriptive names and is relatively small and focuses only on one task.
It will help us to discover new bugs regarding the date validation by throwing an error if it would suddenly be possible to create Users with invalid date ranges.
The selected test is a good example of a unit test, because it focuses on one method only which does exactly one task and returns an easy to check ‘return-type’

Integration Test

```
test( name: '1 Test Valid Add UserProfile Request', fn: () => {
    const expected_response = {
        profileId: "123",
        firstName: "test",
        lastName: "test",
        description: "",
        biography: "",
        tags: [],
        pictureReferences: [],
        matches: {},
        creationDate: new Date(StubInputs.getCurrentDateStr()),
        onlineStatus: "ONLINE",
        birthday: "1999-06-22T00:00:00.000Z",
        email: "test@test.ch",
        phoneNumber: "0795553030",
        gender: "NOT SET",
        isSearchingRoom: true,
        isAdvertisingRoom: false,
        moveInDate: new Date(NaN),
        moveOutDate: new Date(NaN),
        flatId: "",
        isComplete: false
    }

    const user_repo = new ValidMockUserRepository();
    const flat_repo = new ValidMockFlatRepository();
    const ds = new UserProfileDataService(user_repo, flat_repo, jest.fn());

    return ds.addUserProfile(StubInputs.getValidUserPostBody()).then(
        (response : string ) => {
            console.log(response);
            expect(JSON.stringify(response)).toEqual(JSON.stringify(expected_response));
        }
    );
});
```

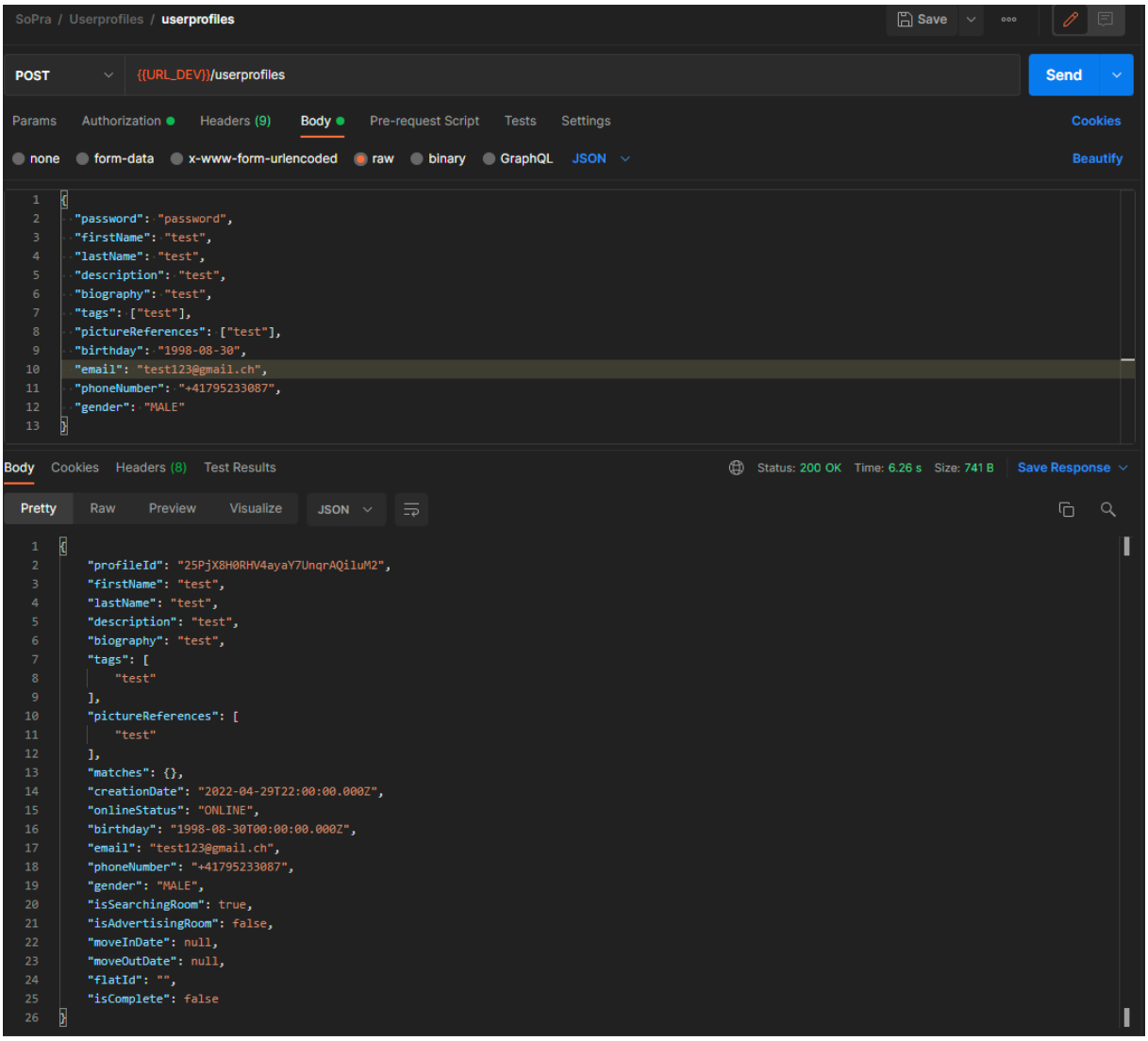
```
export class ValidMockUserRepository implements UserRepository {
    constructor() {
        this.collection_name = "";
        this.database = null;
    }

    collection_name: string;
    database: any;

    addProfile(user_to_add: UserProfile): Promise<string> {
        console.log("Entered Mock addUserProfile");
        return Promise.resolve( value: "Successfully added " + user_to_add.email);
    }
```

This integration test examines the combined behavior of the UserProfileConverter, the UserValidator integrated in the UserProfileDataService. First there is a json which defines the expected response from the method call which is carried out. Then there are mock repositories instantiated for the dependency injection. Last but not least the call is carried out and the actual response is compared to the expected response.

This test examines all methods which act inside the UserProfileDataService and do not affect external resources It will fail and alert the developers if the interaction between the Classes and Methods change or request process is altered in a way that effects the response. Due to the examination of multiple, interacting components which are covered by this test, we consider this as a typical example of an integration test.

Rest Interface Test

Due to the fact that we are building our backend with Firebase cloud functions, we decide to implement our REST interface tests via Postman. There is no other simple way to examine the access points of the cloud functions in a meaningful way. An export of our complete Postman Suit is stored in the assets of the roomeight-backend repository.

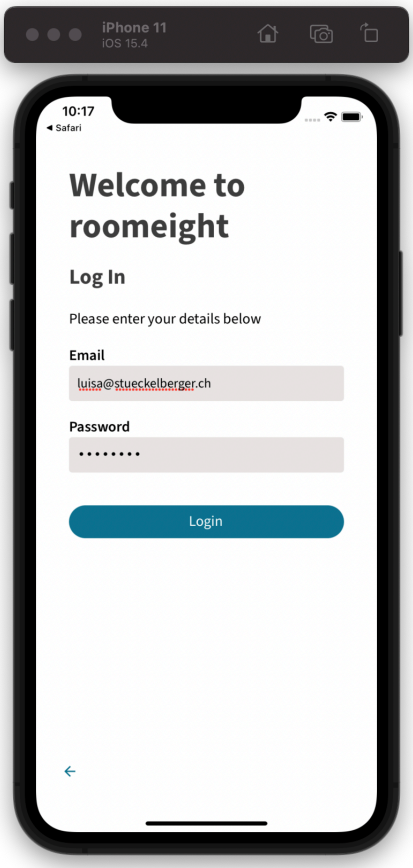


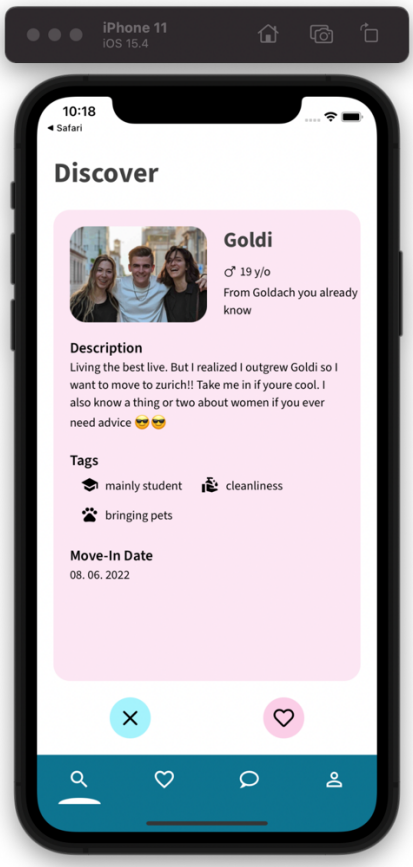
By pressing the send button the request with the defined body is carried out. One can quickly identify if the returned response is a valid one or if an error is thrown. Therefore this test lets us know if something unexpected is returned at any of the endpoints. If any of the endpoints are broken in a way it will be visible here immediately. This is a good example of a REST interface test because the request is sent in a realistic way and the whole endpoint is tested.

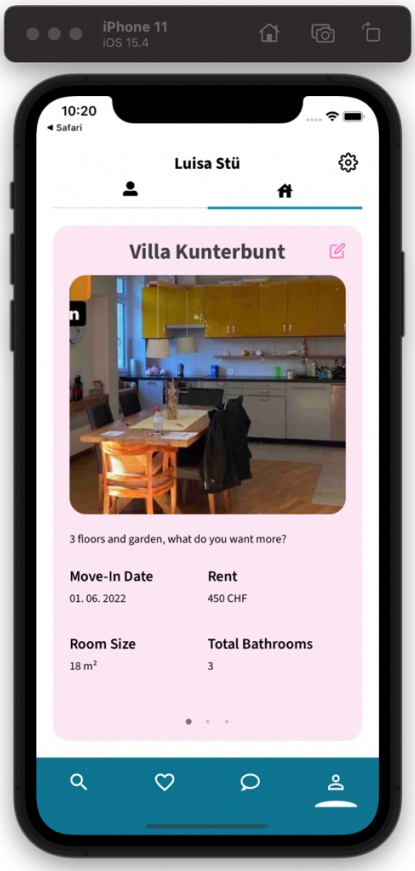
Database Layout

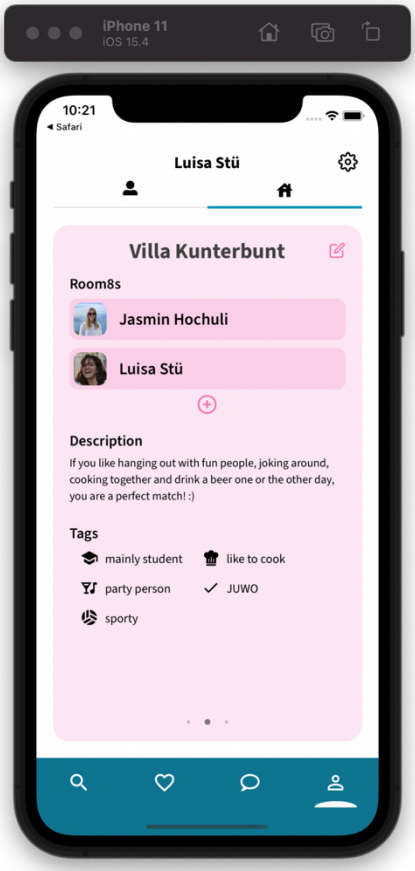
```
Tables:
      user-profiles(
                        biography: string,
                        birthday: Timestamp,
                        creationDate: Timestamp,
                        description: string,
                        email: string,
                        firstName: string,
                        flatId: string,
                        gender: string,
                        isAdvertisingRoom: boolean,
                        isComplete: boolean,
                        isSearchingRoom: boolean,
                        lastName: string,
                        likes: Array<string>,
                        matches: Array<string>,
                        moveInDate: Timestamp,
                        moveOutDate: Timestamp,
                        onlineStatus: string,
                        phoneNumber: string,
                        pictureReferences: Array<string>,
                        profileId: string,
                        tags: Array<string>,
                        viewed: Array<string>
                    ),

      flat-profile(
                        address: string,
                        biography: string,
                        creationDate: Timestamp,
                        description: string,
                        likes: Array<{likedUser: string, likes: Array<string>}>
                        matches: Array<string>
                        moveInDate: Timestamp,
                        moveOutDate: Timestamp,
                        name: string,
                        numberOfBaths: number,
                        numberOfRoomates: number,
                        onlineStatus: string,
                        permanent: boolean,
                        pictureReferences: Array<string>,
                        profileId: string,
                        rent: number,
                        roomMates: Array<string>,
                        roomSize: number,
                        tags: Array<string>
                    )
```

Screenshots from the Frontend









Lars Bösch, Jordi Küffer, Jasmin Hochuli, Christoph Bachmann