Code Refactoring

CSE2115 Software Engineering Methods Group 08b

Y2Q2 2021/2022

1 CodeMR Report Before Refactoring

Most of the metrics did not show any very-high or high problems in the project. The major issue within the code involved coupling, which can be seen in the pie-chart in the figure below (Figure 1.1). The project has 21.3% of medium-high coupling. Hence the decision was made to focus on this metric for most of the method and class restructurings performed for this assignment.

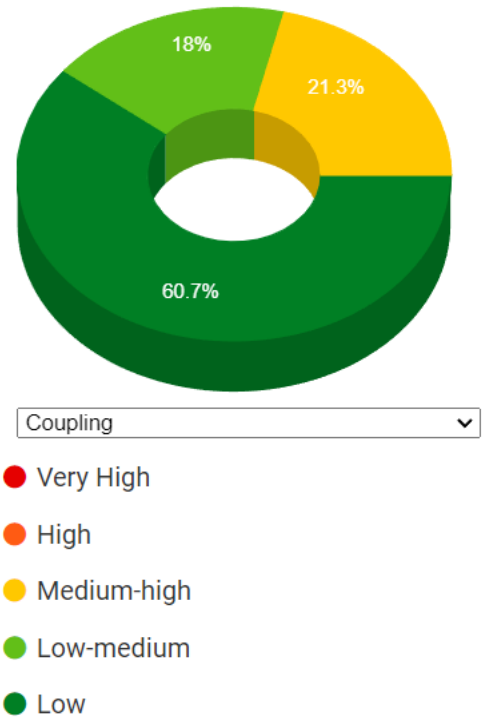

Figure 1.1: Caption

2 Method Level Refactoring

2.1 updateBooking()

Name	Complexity	Coupling
updateBooking(Booking, Long, String): boolean	low	very-high

Figure 2.1: updateBooking() report before refactoring

The CodeMR report has found that the `updateBooking()` (Figure 5.1) method has a very high coupling level (Figure 2.1). Using IntelliJ's refactoring tool, it was possible to easily notice and fix this issue, which mainly came down to duplicate code. The extracted method is called `sendPutBookingRequest()` (Figure 5.2). After that operation the coupling of `updateBooking()` went down to medium-high (Figure 2.2).

Name	Complexity	Coupling
m updateBooking(Booking, Long, String): boolean	low	medium-high

Figure 2.2: updateBooking() report after first refactoring

Another improvement made was moving the booking validating operations to a separate method: The method `validateBooking()` (Figure 5.3). After this change CodeMR shows low-medium coupling level for `updateBooking()` (Figure 2.3). The refactored method can be seen on Figure 5.1. Both newly created methods have been tested.

Name	Complexity	Coupling
m updateBooking(Booking, Long, String): boolean	low	low-medium

Figure 2.3: updateBooking() report after refactoring

2.2 getMyGroup()

Name	Complexity	Coupling
m getMyGroup(Long, String, String): Group	low	medium-high

Figure 2.4: getMyGroup() report before refactoring

In the CodeMR report it can be seen that the `getMyGroup()` (Figure 5.5) method has medium-high coupling (Figure 2.4). The reason for this high coupling came down to code duplication. Using the refactoring tool the new method `getGroupRequest()` (Figure 5.6) was extracted and created. Appropriate changes were applied to `getMyGroup()` (Figure 5.7). After generating another CodeMR report, it can be seen that the coupling for `getMyGroup()` is now low (Figure 2.5). After this, some tests were added for the `getGroupRequest()` method to ensure proper functionality.

Name	Complexity	Coupling
m getMyGroup(Long, String, String): Group	low	low

Figure 2.5: getMyGroup() report after refactoring

2.3 getBooking()

Name	Complexity	Coupling
checkIfRoomsConnected( ): Str	low	low
deleteBooking(String, Long, S	low	medium-high
getAllBookings(String): List	low	low
getBooking(Long, String): Bo	low	very-high
getFutureBookings(String): Li	low	low
getMyBookingsChrono(String	low	low

Figure 2.6: getBooking() report before refactoring

The CodeMR report shows that `getBooking()` method in the main gateway has very high coupling (Figure 2.6). IntelliJ did not recommend any changes, but by extracting the method, as seen in Figure 5.11, a big difference can still be made. The coupling of the method changes from very-high to low-medium (Figure 2.7). Figure 5.11 shows this new method, which is called `sendGetBookingRequest()`. Figure 5.10 shows what the code before refactoring.

Name	Complexity	Coupling
BookingController	low-medium	medium-high
addBooking(Booking, String	low	low
addBookingForGroup(Bookir	low	low
checkIfRoomsConnected( ): St	low	low
deleteBooking(String, Long, S	low	medium-high
getAllBookings(String): List	low	low
getBooking(Long, String): Bc	low	low-medium
getFutureBookings(String): L	low	low
getMyBookingsChrono(String	low	low

Figure 2.7: getBooking() report after refactoring

2.4 deleteBooking()

Name	Complexity	Coupling	Size	Lack of Cohesio
deleteBooking(String, Long, S	low	medium-high	low	low

Figure 2.8: deleteBooking() report before refactoring

The CodeMR report alerted to the `deleteBooking()` method in main gateway, which had medium-high coupling (Figure 2.8). By extracting some duplicate code, a new helper method `sendDeleteBookingRequest()` was created. Upon doing so, the coupling of the method changed from medium-high to low-medium, as shown in Figure 2.9. Figure 5.12 shows what the method looked like before refactoring, while Figure 5.13 shows the function after the changes have been implemented.

Name	Complexity	Coupling	Size	Lack of Cohesion
deleteBooking(String, Long, S	low	low-medium	low	low

Figure 2.9: deleteBooking() report after refactoring

2.5 getBuilding()

Name	Complexity	Coupling	Size	Lack of Cohesion
getBuilding(int, String): Build	low	very-high	low	low

Figure 2.10: getBuilding() report before refactoring

In the CodeMR report it can be seen that the `getBuilding()` method in main gateway had very high coupling(Figure 2.10). By extracting some duplicate code, the helper method `sendGetBuildingRequest()` was created. Upon doing so, the coupling of the method changes from very high to low-medium, as shown in Figure 2.11. Figure 5.14 shows what the method looked like before, while Figure 5.15 showcases the changes.

Name	Complexity	Coupling	Size	Lack of Cohesion
getBuilding(int, String): Build	low	low-medium	low	low

Figure 2.11: getBuilding() report after refactoring

3 Class Level Refactoring

3.1 MainRoomController

The CodeMR report showed that the coupling was medium-high in the MainRoomController class in the main gateway. This meant the class was too interdependent on the booking and room controller classes.

Splitting these methods into a separate class would solve this coupling. Figure 5.14 depicts the issue in the class, the highlighted code is what causes the high coupling/inter-dependency, as it uses methods from the room controller and booking controller classes.

Hence, a new class called SearchController was created which contain these methods to reduce this coupling. The high coupling before refactoring can be seen in Figure 3.1, whereas the coupling after the refactoring can be seen in Figure 3.2.

Name	Complexity	Coupling	Size	Lack of Cohes...
MainRoomController	low-medium	medium-high	low-medium	low
Authorization	low	low-medium	low	low
BuildingController	low	medium-high	low	low
GroupController	low	low-medium	low-medium	low
HelloController	low	low	low	low
RoleController	low	low	low	low

Figure 3.1: MainRoomController report before refactoring

Name	Complexity	Coupling	Size	Lack of Cohesio
template				
nl.tudelft.sem.template	low	low	low	low
nl.tudelft.sem.template.controllers	low	low-medium	medium-high	low
Authorization	low	low-medium	low	low
BookingController	low-medium	medium-high	low-medium	low
BuildingController	low	medium-high	low	low
GroupController	low	low-medium	low-medium	low
HelloController	low	low	low	low
MainRoomController	low	low-medium	low-medium	low

Figure 3.2: MainRoomController report after refactoring

3.2 BuildingController

The report further highlighted medium-high coupling in the classes, this can be seen in Figure 3.3. After further inspection, it could be seen that most of the methods in this class caused high coupling, thus a splitting was required.

The class would be split into two parts, equally splitting the methods into both classes. Thus a SecondBuildingController class was created to contain some of these methods. The splitting of the class has solved the medium-high coupling, which is now a low-medium coupling as shown in Figure 3.4.

Name	Complexity	Coupling	Size	Lack of Cohesion
template				
nl.tudelft.sem.template	low	low	low	low
nl.tudelft.sem.template.controllers	low	low-medium	medium-high	low
Authorization	low	low-medium	low	low
BookingController	low-medium	medium-high	low-medium	low
BuildingController	low	medium-high	low-medium	low

Figure 3.3: BuildingController report before refactoring

Name	Complexity	Coupling	Size	Lack of Cohesion
template				
nl.tudelft.sem.template	low	low	low	low
nl.tudelft.sem.template.controllers	low	medium-high	medium-high	low
Authorization	low	low-medium	low	low
BookingController	low-medium	medium-high	low-medium	low
BuildingController	low	low	low	low
GroupController	low	low-medium	low-medium	low
HelloController	low	low	low	low
MainRoomController	low	low-medium	low-medium	low
RoleController	low	low	low	low
RoomController	low	low	low	low
SearchController	low-medium	low-medium	low-medium	low
SecondBuildingController	low	low-medium	low	low

Figure 3.4: BuildingController report after refactoring

3.3 BookingController

Name	Complexity	Coupling
BookingController	low-medium	medium-high

Figure 3.5: BookingController report before refactoring.

The CodeMR report showed that `BookingController` class had a medium-high level of coupling (Figure 3.5). The methods with mapping "mybookings" (Figure 5.17) were extracted to a separate controller class named `myBookingController` (Figure 5.18). The tests for these methods are also moved from `BookingControllerTest` to a new class `MyBookingsControllerTest` class. With that, the coupling for both classes goes down to low-medium (Figure 3.6).

Name	Complexity	Coupling
BookingController	low	low-medium

Name	Complexity	Coupling
MyBookingsController	low	low-medium

Figure 3.6: BookingController and MyBookingsController report after refactoring.

3.4 UserController

Name	Complexity	Coupling
UserController	low	low-medium
HelloController	low	low
RoleController	low	low
SecurityController	low	low

Figure 3.7: UserController report before refactoring.

After improving code smells of all classes in the current version of the project we turned to our previous refactoring efforts. We found that when we initially implemented the user microservice the UserController class coupling was low-medium. As you can see in Figure 3.7 the UserController Coupling is low-medium. In order to reduce it to low we extracted repeating code into a separate method. In Figure 3.8 you can see the CodeMR report after the refactoring of the UserController class. In Figure 5.19 you can see the `getList()` method that was extracted from our code.

Name	Complexity	Coupling
HelloController	low	low
MainRoomController	low-medium	medium-high
RoleController	low	low
RoomController	low	low
SecurityController	low	low
UserController	low	low

Figure 3.8: UserController report after refactoring.

3.5 Room

Name	Complexity	Coupling	Size
nl.tudelft.sem.template.objects	low	low	low-medium
Booking	low-medium	low	low-medium
Building	low	low	low-medium
Group	low	low	low-medium
Role	low	low	low
Room	low-medium	low	low-medium
User	low	low	low-medium

Figure 3.9: Room report before refactoring.

When generating the CodeMR report on our code we noticed low-medium complexity of our Room class. In Figure 3.9. When looking into why this was happening we noticed we had written methods that were never used in our application. After reviewing the methods in Figure 5.20 we decided that removing them would not affect the fuctionality of the class. After refactoring we generated the CodeMR report again and in Figure 3.10 you can see the Room class complexity has been reduced to low

Name	Complexity	Coupling	Size
nl.tudelft.sem.template.objects	low	low	low-medium
Booking	low-medium	low	low-medium
Building	low	low	low-medium
Group	low	low	low-medium
Role	low	low	low
Room	low	low	low-medium
User	low	low	low-medium

Figure 3.10: Room report after refactoring.

4 CodeMR Report After Refactoring

After refactoring the 5 methods and 5 classes, we managed to reduce the coupling and the complexity of our project. Since we decided to f we managed to reduce coupling by 6 percent, bringing it to a total of 15.3% medium-low coupling. This can be seen in the pie-chart in the figure below (Figure 4.1).

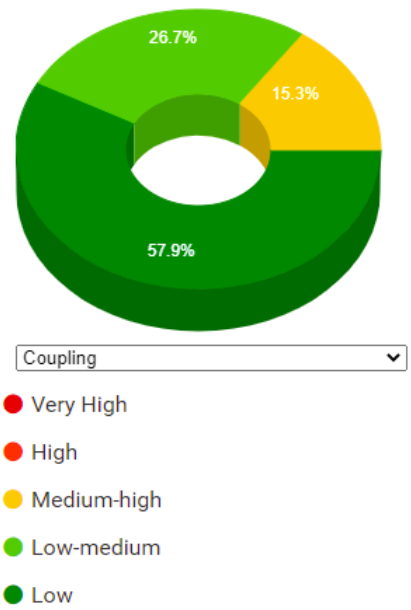

Figure 4.1: CodeMR report after refactoring

5 Appendixes

5.1 Appendix A

```java
public boolean updateBooking(@RequestBody Booking booking, @PathVariable("id") Long id,
                             @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    try {
        Validator handler = validatorCreator(token);
        boolean isValid = handler.handle(booking);
        if (isValid) {
            String uri = "http://localhost:8083/bookings/".concat(String.valueOf(id));
            HttpHeaders headers = new HttpHeaders();
            headers.add(HttpHeaders.AUTHORIZATION, token);
            HttpEntity<Booking> entity = new HttpEntity<>(booking, headers);
            restTemplate.exchange(uri, HttpMethod.PUT, entity, void.class);
            return true;
        }
        return false;

    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.1: updateBooking() method before refactoring

```java
private boolean sendPutBookingRequest(@RequestBody Booking booking,
                                      @RequestHeader(HttpHeaders.AUTHORIZATION) String token,
                                      String uri) throws HttpClientErrorException {
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<Booking> entity = new HttpEntity<>(booking, headers);
    restTemplate.exchange(uri, HttpMethod.PUT, entity, void.class);
    return true;
}
```

Figure 5.2: sendPutBookingRequest() extracted method

```java
private boolean validateBooking(Booking booking, String token)
        throws InvalidBookingException, InvalidRoomException, BuildingNotOpenException {
    Validator handler = validatorCreator(token);
    return handler.handle(booking);
}
```

Figure 5.3: validateBooking() extracted method

```java
public boolean updateBooking(@RequestBody Booking booking, @PathVariable("id") Long id,
                             @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    try {
        if (validateBooking(booking, token)) {
            String uri = "http://localhost:8083/bookings/".concat(String.valueOf(id));
            return sendPutBookingRequest(booking, token, uri);
        }
        return false;

    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.4: updateBooking() method after refactoring

5.2 Appendix B

```java
private Group getGroup(@RequestHeader(HttpHeaders.AUTHORIZATION) String token, String uri) {
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);
    try {
        ResponseEntity<Group> res = restTemplate.exchange(uri,
                HttpMethod.GET, entity, Group.class);
        return res.getBody();
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.5: getMyGroup() method before refactoring

```java
Group getGroupRequest(@RequestHeader(HttpHeaders.AUTHORIZATION) String token, String uri) {
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);
    try {
        ResponseEntity<Group> res = restTemplate.exchange(uri,
                HttpMethod.GET, entity, Group.class);
        return res.getBody();
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.6: getGroupRequest() extracted method

```java
public Group getMyGroup(@PathVariable("id") Long id,
                        @PathVariable("secretaryId") String secretaryId,
                        @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8081/secretary/getMyGroup/"
                + id.toString() + "/"
                + secretaryId;
    return getGroupRequest(token, uri);
}
```

Figure 5.7: getMyGroup() method after refactoring

```java
/**
 * Returns a specific booking with respect to its id.
 *
 * @param id the id of the booking we want.
 * @return the booking we are searching for.
 */
@GetMapping("/getBooking/{id}")
@ResponseBody
public Booking getBooking(@PathVariable("id") Long id,
                          @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8083/getBooking/".concat(String.valueOf(id));
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);

    try {
        ResponseEntity<Booking> res = restTemplate
            .exchange(uri, HttpMethod.GET, entity, Booking.class);
        return res.getBody();
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.8: getBooking() method before refactoring

```java
 */
@GetMapping("/getBooking/{id}")
@ResponseBody
public Booking getBooking(@PathVariable("id") Long id,
                          @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8083/getBooking/".concat(String.valueOf(id));
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    return sendGetBookingRequest(headers, uri);
}

protected Booking sendGetBookingRequest(HttpHeaders headers, String uri) {
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);

    try {
        ResponseEntity<Booking> res = restTemplate
                .exchange(uri, HttpMethod.GET, entity, Booking.class);
        if (res.getBody() == null) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        } else {
            return res.getBody();
        }
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, e.getMessage());
    }
}
```

Figure 5.9: getBooking() method after refactoring

5.3 Appendix C

```java
/**
 * Returns a specific booking with respect to its id.
 *
 * @param id the id of the booking we want.
 * @return the booking we are searching for.
 */
@GetMapping("/getBooking/{id}")
@ResponseBody
public Booking getBooking(@PathVariable("id") Long id,
                          @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8083/getBooking/".concat(String.valueOf(id));
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);

    try {
        ResponseEntity<Booking> res = restTemplate
            .exchange(uri, HttpMethod.GET, entity, Booking.class);
        return res.getBody();
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.10: getBooking() method before refactoring

```java
 */
@GetMapping("/getBooking/{id}")
@ResponseBody
public Booking getBooking(@PathVariable("id") Long id,
                          @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8083/getBooking/".concat(String.valueOf(id));
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    return sendGetBookingRequest(headers, uri);
}

protected Booking sendGetBookingRequest(HttpHeaders headers, String uri) {
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);

    try {
        ResponseEntity<Booking> res = restTemplate
                .exchange(uri, HttpMethod.GET, entity, Booking.class);
        if (res.getBody() == null) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND);
        } else {
            return res.getBody();
        }
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, e.getMessage());
    }
}
```

Figure 5.11: getBooking() method after refactoring

5.4 Appendix D

```java
/**
 * Deletes your booking from the system as an employee.
 *
 * @param id the id of the booking to delete.
 * @return true if successfully deleted, else false.
 */
@DeleteMapping("/deleteBooking/{userId}/{id}")
@ResponseBody
public boolean deleteBooking(@PathVariable(userIdPath) String userId,
                             @PathVariable("id") Long id,
                             @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8083/myBookings/".concat(userId + "/" + String.valueOf(id));
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);
    try {
        restTemplate.exchange(uri, HttpMethod.DELETE, entity, void.class);
        return true;
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.12: deleteBooking() method before refactoring

```java
/**
 * Deletes your booking from the system as an employee.
 *
 * @param id the id of the booking to delete.
 * @return true if successfully deleted, else false.
 */
@DeleteMapping("/deleteBooking/{userId}/{id}")
@ResponseBody
public boolean deleteBooking(@PathVariable(userIdPath) String userId,
                             @PathVariable("id") Long id,
                             @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8083/myBookings/".concat(userId + "/" + String.valueOf(id));
    try {
        return sendDeleteBookingRequest(token, uri);
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}

/**
 * Helper method for sending delete request for a booking.
 * @param token the user's authorization token
 * @param uri address to send the request to
 * @return true if request is successful
 * @throws HttpClientErrorException when response was not 200 OK
 */
private boolean sendDeleteBookingRequest(@RequestHeader(HttpHeaders.AUTHORIZATION) String token,
                                         String uri) throws HttpClientErrorException {
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);
    restTemplate.exchange(uri, HttpMethod.DELETE, entity, void.class);
    return true;
}
```

Figure 5.13: deleteBooking() method after refactoring

5.5 Appendix E

```java
/** Returns a specific building with respect to its id.
 *
 * @param id        the id of the building we want.
 * @param token     the token of the user
 * @return the building we are searching for.
 */
@GetMapping("/getBuilding/{id}")
@ResponseBody
public Building getBuilding(@PathVariable("id") int id,
                            @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8082/getBuilding/".concat(String.valueOf(id));

    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "body", headers);

    try {
        ResponseEntity<Building> res = restTemplate
                .exchange(uri, HttpMethod.GET, entity, Building.class);
        return res.getBody();
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.14: getBuilding() method before refactoring

```java
/** Returns a specific building with respect to its id.
 *
 * @param id        the id of the building we want.
 * @param token     the token of the user
 * @return the building we are searching for.
 */
@GetMapping("/getBuilding/{id}")
@ResponseBody
public Building getBuilding(@PathVariable("id") int id,
                            @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    String uri = "http://localhost:8082/getBuilding/".concat(String.valueOf(id));

    try {
        return sendGetBuildingRequest(token, uri);
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}

/**
 * Helper method for sending get requests for buildings.
 *
 * @param token the user's authorization token
 * @param uri the address to send request to
 * @return the building if it exists
 * @throws HttpClientErrorException otherwise
 */
protected Building sendGetBuildingRequest(@RequestHeader(HttpHeaders.AUTHORIZATION)
                                          String token,
                                          String uri) throws HttpClientErrorException {
    HttpHeaders headers = new HttpHeaders();
    headers.add(HttpHeaders.AUTHORIZATION, token);
    HttpEntity<String> entity = new HttpEntity<>( body: "body", headers);
    ResponseEntity<Building> res = restTemplate
        .exchange(uri, HttpMethod.GET, entity, Building.class);
    return res.getBody();
}
```

Figure 5.15: getBuilding() method after refactoring

5.6 Appendix F

```java
public List<Room> availableRooms(@RequestParam String date, @RequestParam String startTime,
                                 @RequestParam String endTime,
                                 @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {
    ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.registerModule(new JavaTimeModule());


    LocalTime realStartTime = LocalTime.parse(startTime);
    LocalTime realEndTime = LocalTime.parse(endTime);
    if (realEndTime.isBefore(realStartTime)) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                "End time is before start time");
    }
    LocalDate realDate = LocalDate.parse(date);
    if (realDate.isBefore(LocalDate.now())) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                "Date is in the past");
    }
    if (realDate.isEqual(LocalDate.now()) && realStartTime.isBefore(LocalTime.now())) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
                "Start time is in the past");
    }
    List<Room> allRooms = objectMapper.convertValue(roomController.getRooms(token),
            new TypeReference<List<Room>>() {
            });
    List<Booking> allBookings = objectMapper.convertValue(bookingController
                    .getFutureBookings(token),
            new TypeReference<List<Booking>>() {
```

Figure 5.16: MainRoomController code snippet

5.7 Appendix G

```
@PutMapping("/myBookings/{userId}/{id}")
@ResponseBody
public void updateMyBooking(@RequestBody Booking booking,
                            @PathVariable(userIdPath) String userId,
                            @PathVariable("id") Long id,
                            @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {...}


@DeleteMapping("/myBookings/{userId}/{id}")
@ResponseBody
public void deleteBooking(@PathVariable(userIdPath) String userId,
                          @PathVariable("id") Long id,
                          @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {...}

@GetMapping("/myBookings/default/{userId}")
@ResponseBody
public List<Booking> getMyBookingsDefault(@PathVariable(userIdPath) String userId,
                                          @RequestHeader(HttpHeaders
                                                  .AUTHORIZATION) String token) {...}

@GetMapping("/myBookings/chrono/{userId}")
@ResponseBody
public List<Booking> getMyBookingsChrono(@PathVariable(userIdPath) String userId,
                                         @RequestHeader(HttpHeaders
                                                 .AUTHORIZATION) String token) {...}

@GetMapping("/myBookings/location/{userId}")
@ResponseBody
public List<Booking> getMyBookingsLocation(@PathVariable(userIdPath) String userId,
                                           @RequestHeader(HttpHeaders
                                                   .AUTHORIZATION) String token) {...}
```

Figure 5.17: BookingController code snippet before recapturing

```
@Controller
@RequestMapping(value = "/myBookings")
public class MyBookingsController {

    @Autowired
    private transient BookingService bookingService;

    @Autowired
    private transient Authorization auth;

    private static final String userIdPath = "userId";

    @PutMapping("/{userId}/{id}")
    @ResponseBody
    public void updateMyBooking(@RequestBody Booking booking,
                                @PathVariable(userIdPath) String userId,
                                @PathVariable("id") Long id,
                                @RequestHeader(HttpHeaders.AUTHORIZATION) String token) {...}
```

Figure 5.18: MyBookingsController code snippet before refactoring

5.8 Appendix H

```java
private List getList(String uri, HttpHeaders headers, RestTemplate restTemplate) {
    HttpEntity<String> entity = new HttpEntity<>( body: "", headers);
    try {
        ResponseEntity<List> res = restTemplate.exchange(uri,
                HttpMethod.GET, entity, List.class);
        return res.getBody();
    } catch (HttpClientErrorException e) {
        throw new ResponseStatusException(e.getStatusCode(), e.toString());
    } catch (Exception e) {
        throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR, "");
    }
}
```

Figure 5.19: UserController getList() method.

5.9 Appendix I

```java
public void setRoomNumber(int roomNumber) {
    this.roomNumber = roomNumber;
}

public void setName(String name) {
    this.name = name;
}

public void setCapacity(int capacity) {
    this.capacity = capacity;
}

public void setEquipment(Map<String, String> equipment) {
    this.equipment = equipment;
}

public void setAvailable(String available) {
    this.available = available;
}
```

Figure 5.20: Methods removed from Room class