Group 5A / Assignment 3

Mutation Testing Report

Task 1: Automated Mutation Testing

1.1: AllergyController

```
     @throws Exception Bad Request if user isn't found in database.
      */
     @PostMapping("/addAllergies")
     public ResponseEntity addAllergy(@RequestBody AllergiesRequestModel ar) throws Exception {
1        if (us.findByNetId(new NetId(am.getNetId())).isPresent()) {
             AppUser user = us.findByNetId(new NetId(am.getNetId())).get();
1            user.addAllergies(ar.getAllergies());
             us.save(user);
1            return ResponseEntity.ok().build();
         } else {
             throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Invalid token");
         }

     }

     /**
      * Get mapping for all allergies using token.
      *
      * @return returns Response Entity with the response model which is a
      * @throws Exception if user isn't found in database.
      */
     @GetMapping("/getAllergies")
     public ResponseEntity<AllergiesResponseModel> getAllergies() throws Exception {
         String netId = am.getNetId();

1        if (us.findByNetId(new NetId(am.getNetId())).isPresent()) {
             AppUser user = us.findByNetId(new NetId(am.getNetId())).get();
             List<Long> allergies = user.getAllergies();
1            return ResponseEntity.ok(new AllergiesResponseModel(allergies));

         } else {
             throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Invalid token");
         }
     }
```

Name	Line Coverage	Mutation Coverage
AllergiesController.java	25% 4/16	0% 0/5

(Before)

We chose to test the allergyController class as it has all the functionalities for the allergies which are a main part of our software. We added a bunch of new tests which brought the mutation coverage up to 100%. Here are the tests and the results:

```java
@Test
void addAllergyOk() throws Exception{
  AllergiesRequestModel ar = new AllergiesRequestModel();
  List<Long> allergies = new ArrayList<>(List.of(1L, 2L));
  ar.setAllergies(allergies);

  ResponseEntity r = allergiesController.addAllergy(ar);

  assertThat(r.getStatusCode()).isEqualTo(HttpStatus.OK);
}

@Test
void addAllergyFail() throws Exception{
  when(mockAuthenticationManager.getNetId()).thenReturn("NoUser");
  when(mockJwtTokenVerifier.getUserIdFromToken(anyString())).thenReturn("NoUser");
  when(userRepository.findByNetId(new NetId("NoUser"))).thenReturn(Optional.empty());

  AllergiesRequestModel ar = new AllergiesRequestModel();
  List<Long> allergies = new ArrayList<>(List.of(1L, 2L));
  ar.setAllergies(allergies);

  assertThatThrownBy( () -> {
    ResponseEntity r = allergiesController.addAllergy(ar);
  }).isInstanceOf(ResponseStatusException.class).hasMessageContaining("400 BAD_REQUEST \"Invalid token\"");

}
```

```java
@Test
void addAllergySuccessful() throws Exception{
  AllergiesRequestModel ar = new AllergiesRequestModel();
  List<Long> allergies = new ArrayList<>(List.of(5L, 6L));
  ar.setAllergies(allergies);

  ResponseEntity r = allergiesController.addAllergy(ar);

  ResponseEntity<AllergiesResponseModel> result = allergiesController.getAllergies();
  List<Long> real = new ArrayList<>(List.of(1L, 2L, 3L, 5L, 6L));
  assertThat(result.getBody().getAllergies()).isEqualTo(real);
}

@Test
void getAllergiesOk() throws Exception {
  ResponseEntity<AllergiesResponseModel> r = allergiesController.getAllergies();
  assertThat(r.getStatusCode()).isEqualTo(HttpStatus.OK);
}

@Test
void getAllergiesList() throws Exception {
  ResponseEntity<AllergiesResponseModel> r = allergiesController.getAllergies();
  List<Long> real = new ArrayList<>(List.of(1L, 2L, 3L));
  assertThat(r.getBody().getAllergies()).isEqualTo(real);
}
```

```java
@Test
void getAllergiesNoUserFound() throws Exception {
  when(mockAuthenticationManager.getNetId()).thenReturn("NoUser");
  when(mockJwtTokenVerifier.getUserIdFromToken(anyString())).thenReturn("NoUser");
  when(userRepository.findByNetId(new NetId("NoUser"))).thenReturn(Optional.empty());

  assertThatThrownBy( () -> {
    ResponseEntity<AllergiesResponseModel> e = allergiesController.getAllergies();
  }).isInstanceOf(ResponseStatusException.class).hasMessageContaining("400 BAD_REQUEST \"Invalid token\"");
 }
}
```

```java
@PostMapping("/addAllergies")
public ResponseEntity addAllergy(@RequestBody AllergiesRequestModel ar) throws Exception {
    if (us.findByNetId(new NetId(am.getNetId())).isPresent()) {
        AppUser user = us.findByNetId(new NetId(am.getNetId())).get();
        user.addAllergies(ar.getAllergies());
        us.save(user);
        return ResponseEntity.ok().build();
    } else {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Invalid token");
    }

}

/**
 * Get mapping for all allergies using token.
 *
 * @return returns Response Entity with the response model which is a
 * @throws Exception if user isn't found in database.
 */
@GetMapping("/getAllergies")
public ResponseEntity<AllergiesResponseModel> getAllergies() throws Exception {
    String netId = am.getNetId();

    if (us.findByNetId(new NetId(am.getNetId())).isPresent()) {
        AppUser user = us.findByNetId(new NetId(am.getNetId())).get();
        List<Long> allergies = user.getAllergies();
        return ResponseEntity.ok(new AllergiesResponseModel(allergies));

    } else {
        throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Invalid token");
    }
}
```

Name	Line Coverage		Mutation Coverage	
AllergiesController.java	100%	16/16	100%	5/5

(After)

1.2: BasketService

After looking through the PITest report the following class stood out with the following mutants:

Name	Line Coverage		Mutation Coverage	
BasketService.java	73%	37/51	45%	10/22

40	1. replaced return value with null for nl/tudelft/sem/template/basket/services/BasketService::getBasket → KILLED
56	1. negated conditional → KILLED
58	1. Replaced double addition with subtraction → SURVIVED
60	1. removed call to commons/BasketInfo::setPrice → SURVIVED
64	1. removed call to commons/BasketInfo::setPrice → NO_COVERAGE
77	1. removed call to nl/tudelft/sem/template/basket/services/BasketService::calculatePrice → SURVIVED
83	1. negated conditional → KILLED
88	1. removed call to nl/tudelft/sem/template/basket/services/BasketService::calculatePrice → NO_COVERAGE
104	1. negated conditional → KILLED
105	1. removed call to commons/BasketInfo::setPrice → SURVIVED
106	1. removed call to commons/BasketInfo::setCoupon → KILLED
107	1. replaced boolean return with false for nl/tudelft/sem/template/basket/services/BasketService::applyCouponToBasket → KILLED
109	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
110	1. removed call to commons/BasketInfo::setPrice → NO_COVERAGE
111	1. removed call to commons/BasketInfo::setCoupon → NO_COVERAGE
112	1. replaced boolean return with false for nl/tudelft/sem/template/basket/services/BasketService::applyCouponToBasket → NO_COVERAGE
113	1. replaced boolean return with true for nl/tudelft/sem/template/basket/services/BasketService::applyCouponToBasket → KILLED
124	1. removed call to commons/BasketInfo::setCoupon → KILLED
125	1. removed call to nl/tudelft/sem/template/basket/services/BasketService::calculatePrice → SURVIVED
137	1. removed call to commons/BasketInfo::setStoreId → NO_COVERAGE
138	1. replaced return value with "" for nl/tudelft/sem/template/basket/services/BasketService::setStorePreference → NO_COVERAGE

This class is the BasketService class. It takes care of the user's basket: Adding and removing pizzas, applying a coupon and calculating the price of the basket. These are all very important and therefore the class should be well tested. As can be seen above, the BasketService class has decent line coverage, but the mutation coverage falls behind a bit. However, most of the coverage comes from integration testing currently. To properly test this class, we created a new test class for testing the BasketService class: the BasketServiceTests class. This class contains 8 new tests that extensively test the functionality of BasketService. See below the coverages.of the class after the new test class has been added:

BasketService.java	92%	47/51	95%	21/22

Mutations

40	1. replaced return value with null for nl/tudelft/sem/template/basket/services/BasketService::getBasket → KILLED
56	1. negated conditional → KILLED
58	1. Replaced double addition with subtraction → KILLED
60	1. removed call to commons/BasketInfo::setPrice → KILLED
64	1. removed call to commons/BasketInfo::setPrice → NO_COVERAGE
77	1. removed call to nl/tudelft/sem/template/basket/services/BasketService::calculatePrice → KILLED
83	1. negated conditional → KILLED
88	1. removed call to nl/tudelft/sem/template/basket/services/BasketService::calculatePrice → KILLED
104	1. negated conditional → KILLED
105	1. removed call to commons/BasketInfo::setPrice → TIMED_OUT
106	1. removed call to commons/BasketInfo::setCoupon → KILLED
107	1. replaced boolean return with false for nl/tudelft/sem/template/basket/services/BasketService::applyCouponToBasket → KILLED
109	1. changed conditional boundary → KILLED 2. negated conditional → KILLED
110	1. removed call to commons/BasketInfo::setPrice → KILLED
111	1. removed call to commons/BasketInfo::setCoupon → KILLED
112	1. replaced boolean return with false for nl/tudelft/sem/template/basket/services/BasketService::applyCouponToBasket → KILLED
113	1. replaced boolean return with true for nl/tudelft/sem/template/basket/services/BasketService::applyCouponToBasket → KILLED
124	1. removed call to commons/BasketInfo::setCoupon → KILLED
125	1. removed call to nl/tudelft/sem/template/basket/services/BasketService::calculatePrice → KILLED
137	1. removed call to commons/BasketInfo::setStoreId → KILLED
138	1. replaced return value with "" for nl/tudelft/sem/template/basket/services/BasketService::setStorePreference → TIMED_OUT

Below are the PITest report code coverages before and after the addition of the new tests:

Before:

```
48      public void removeBasket(String customerId) {
49          baskets.remove(customerId);
50      }
51
52      public void calculatePrice(String customerId) {
53          Basket basket = baskets.get(customerId);
54          double price = 3.0;
55
56 1        if (basket.getBasketInfo().getCoupon() == null) {
57              for (Pizza p : basket.getBasketInfo().getPizzas()) {
58 1              price += p.getPrice();
59              }
60 1            basket.getBasketInfo().setPrice(price);
61          } else {
62              price = basket.getBasketInfo().getCoupon().calculatePrice(basket.getBasketInfo()
63                  .getPizzas());
64 1            basket.getBasketInfo().setPrice(price);
65          }
66      }
67
68      /**
69       * Adds the passed pizza to the basket of the customer.
70       *
71       * @param customerId ID of the owner of the basket
72       * @param pizza      the pizza to be added
73       */
74      public void addPizzaToBasket(String customerId, Pizza pizza) {
75          Basket basket = baskets.get(customerId);
76          basket.getBasketInfo().getPizzas().add(pizza);
77 1        calculatePrice(customerId);
78      }
79
80      public void removePizzaFromBasket(String customerId, String pizzaName) {
81          Basket basket = baskets.get(customerId);
82          for (Pizza pizza : basket.getBasketInfo().getPizzas()) {
83 1            if (pizza.getName().equals(pizzaName)) {
84                  basket.getBasketInfo().getPizzas().remove(pizza);
85                  return;
86              }
87          }
88 1        calculatePrice(customerId);
89      }
90
98       */
99      public boolean applyCouponToBasket(String customerId, Coupon coupon) {
100         Basket basket = baskets.get(customerId);
101         Coupon curr = basket.getBasketInfo().getCoupon();
102         double newPrice = coupon.calculatePrice(basket.getBasketInfo().getPizzas());
103
104 1       if (curr == null) {
105 1           basket.getBasketInfo().setPrice(newPrice);
106 1           basket.getBasketInfo().setCoupon(coupon);
107 1           return true;
108         } else {
109 2           if (newPrice < basket.getBasketInfo().getPrice()) {
110 1               basket.getBasketInfo().setPrice(newPrice);
111 1               basket.getBasketInfo().setCoupon(coupon);
112 1               return true;
113 1           } else return false;
114         }
115     }
116
117     /**
118      * Removes a coupon from basket and re-calculates the price.
119      *
120      * @param customerId ID of the owner of the basket
121      */
122     public void removeCouponFromBasket(String customerId) {
123         Basket basket = baskets.get(customerId);
124 1       basket.getBasketInfo().setCoupon(null);
125 1       calculatePrice(customerId);
126     }
127
128     /**
129      * Sets the store preference of the customer to the basket's field.
130      *
131      * @param customerId the owner of the basket's id
132      * @param storeId    the id of the store to be ordered from
133      * @return Message
134      */
135     public String setStorePreference(String customerId, int storeId) {
136         Basket basket = baskets.get(customerId);
137 1       basket.getBasketInfo().setStoreId(storeId);
138 1       return "Store preference saved.";
139     }
140 }
```

After:

```
47      /
48      public void removeBasket(String customerId) {
49          baskets.remove(customerId);
50      }
51
52      public void calculatePrice(String customerId) {
53          Basket basket = baskets.get(customerId);
54          double price = 3.0;
55
56 1        if (basket.getBasketInfo().getCoupon() == null) {
57              for (Pizza p : basket.getBasketInfo().getPizzas()) {
58 1              price += p.getPrice();
59              }
60 1            basket.getBasketInfo().setPrice(price);
61          } else {
62              price = basket.getBasketInfo().getCoupon().calculatePrice(basket.getBasket
63                  .getPizzas());
64 1            basket.getBasketInfo().setPrice(price);
65          }
66      }
67
68      /**
69       * Adds the passed pizza to the basket of the customer.
70       *
71       * @param customerId ID of the owner of the basket
72       * @param pizza      the pizza to be added
73       */
74      public void addPizzaToBasket(String customerId, Pizza pizza) {
75          Basket basket = baskets.get(customerId);
76          basket.getBasketInfo().getPizzas().add(pizza);
77 1        calculatePrice(customerId);
78      }
79
80      public void removePizzaFromBasket(String customerId, String pizzaName) {
81          Basket basket = baskets.get(customerId);
82          for (Pizza pizza : basket.getBasketInfo().getPizzas()) {
83 1            if (pizza.getName().equals(pizzaName)) {
84                  basket.getBasketInfo().getPizzas().remove(pizza);
85                  break;
86              }
87          }
88 1        calculatePrice(customerId);
89      }
90
98       */
99      public boolean applyCouponToBasket(String customerId, Coupon coupon) {
100         Basket basket = baskets.get(customerId);
101         Coupon curr = basket.getBasketInfo().getCoupon();
102         double newPrice = coupon.calculatePrice(basket.getBasketInfo().getPizzas());
103
104 1       if (curr == null) {
105 1           basket.getBasketInfo().setPrice(newPrice);
106 1           basket.getBasketInfo().setCoupon(coupon);
107 1           return true;
108         } else {
109 2           if (newPrice < basket.getBasketInfo().getPrice()) {
110 1               basket.getBasketInfo().setPrice(newPrice);
111 1               basket.getBasketInfo().setCoupon(coupon);
112 1               return true;
113 1           } else return false;
114         }
115     }
116
117     /**
118      * Removes a coupon from basket and re-calculates the price.
119      *
120      * @param customerId ID of the owner of the basket
121      */
122     public void removeCouponFromBasket(String customerId) {
123         Basket basket = baskets.get(customerId);
124 1       basket.getBasketInfo().setCoupon(null);
125 1       calculatePrice(customerId);
126     }
127
128     /**
129      * Sets the store preference of the customer to the basket's field.
130      *
131      * @param customerId the owner of the basket's id
132      * @param storeId    the id of the store to be ordered from
133      * @return Message
134      */
135     public String setStorePreference(String customerId, int storeId) {
136         Basket basket = baskets.get(customerId);
137 1       basket.getBasketInfo().setStoreId(storeId);
138 1       return "Store preference saved.";
139     }
140 }
```

Below are all the new tests which were added:

```java
@BeforeEach
public void setup() {
    Ingredient ingredient1 = new Ingredient( name: "ingredient1",  price: 5.00);
    Ingredient ingredient2 = new Ingredient( name: "ingredient1",  price: 10.00);
    Ingredient ingredient3 = new Ingredient( name: "ingredient1",  price: 2.50);
    List<Ingredient> ingredientsPizza1 = new ArrayList<>(List.of(ingredient1, ingredient2));
    pizza1 = new Pizza( name: "testPizza1", ingredientsPizza1);
    List<Ingredient> ingredientsPizza2 = new ArrayList<>(List.of(ingredient1, ingredient3));
    pizza2 = new Pizza( name: "testPizza2", ingredientsPizza2);

    basketService = new BasketService();
}

@Test
public void removeBasketTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");

    // act
    basketService.removeBasket( customerId: "testCustomer");

    // assert
    assertThat(basketService.getBasket( customerId: "testCustomer")).isNull();
}
```

```java
@Test
public void addPizzaToBasketTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");

    // act
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza1);

    // assert pizza was added
    List<Pizza> pizzasInBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPizzas();
    assertThat(pizzasInBasket).containsExactly(pizza1);
    // assert price was updated
    double priceOfBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPrice();
    assertThat(priceOfBasket).isEqualTo(21.00);
}

@Test
public void removePizzaFromBasketTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza1);

    // act
    basketService.removePizzaFromBasket( customerId: "testCustomer",  pizzaName: "testPizza1");

    // assert pizza has been removed
    List<Pizza> pizzasInBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPizzas();
    assertThat(pizzasInBasket).hasSize(0);
    // assert price was updated
    double priceOfBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPrice();
    assertThat(priceOfBasket).isEqualTo(3.00);
```

```java
@Test
public void applyCouponNoPriorCouponTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza1);
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza2);

    Coupon coupon = new Coupon( code: "coup10",  type: 'D',  rate: 50.0,  limitedTime: false);

    // act
    basketService.applyCouponToBasket( customerId: "testCustomer", coupon);

    // assert coupon is applied
    Coupon couponInBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getCoupon();
    assertThat(couponInBasket).isEqualTo(coupon);

    // assert price is updated
    double priceOfBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPrice();
    assertThat(priceOfBasket).isEqualTo(17.25);
}
```

```java
@Test
public void applyCheaperCouponTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza1);
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza2);

    Coupon expensiveCoupon = new Coupon( code: "coup11",  type: 'D',  rate: 25.0,  limitedTime: false);
    Coupon cheaperCoupon = new Coupon( code: "coup10",  type: 'D',  rate: 50.0,  limitedTime: false);
    basketService.applyCouponToBasket( customerId: "testCustomer", expensiveCoupon);

    // act
    basketService.applyCouponToBasket( customerId: "testCustomer", cheaperCoupon);

    // assert correct coupon is set
    Coupon currentCouponApplied = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getCoupon();
    assertThat(currentCouponApplied).isEqualTo(cheaperCoupon);

    // assert price is correct
    double priceOfBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPrice();
    assertThat(priceOfBasket).isEqualTo(17.25);
}
```

```java
@Test
public void applyMoreExpensiveCouponTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza1);
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza2);

    Coupon expensiveCoupon = new Coupon( code: "coup11",  type: 'D',  rate: 25.0,  limitedTime: false);
    Coupon cheaperCoupon = new Coupon( code: "coup10",  type: 'D',  rate: 50.0,  limitedTime: false);
    basketService.applyCouponToBasket( customerId: "testCustomer", cheaperCoupon);

    // act
    basketService.applyCouponToBasket( customerId: "testCustomer", expensiveCoupon);

    // assert correct coupon is set
    Coupon currentCouponApplied = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getCoupon();
    assertThat(currentCouponApplied).isEqualTo(cheaperCoupon);

    // assert price is correct
    double priceOfBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPrice();
    assertThat(priceOfBasket).isEqualTo(17.25);
}
```

```java
@Test
public void removeCouponTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza1);
    basketService.addPizzaToBasket( customerId: "testCustomer", pizza2);

    Coupon coupon = new Coupon( code: "coup11",  type: 'D',  rate: 25.0,  limitedTime: false);
    basketService.applyCouponToBasket( customerId: "testCustomer", coupon);

    // act
    basketService.removeCouponFromBasket( customerId: "testCustomer");

    // assert correct coupon is set
    Coupon currentCouponApplied = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getCoupon();
    assertThat(currentCouponApplied).isNull();

    // assert price is correct
    double priceOfBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getPrice();
    assertThat(priceOfBasket).isEqualTo(31.50);
}
```

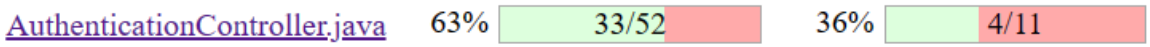
```java
@Test
public void setStorePreferenceTest() {
    // arrange
    basketService.createBasket( customerId: "testCustomer");

    // act
    basketService.setStorePreference( customerId: "testCustomer",  storeId: 2);

    // assert
    int storePreferenceInBasket = basketService.getBasket( customerId: "testCustomer").getBasketInfo().getStoreId();
    assertThat(storePreferenceInBasket).isEqualTo(2);
}
```

1.3: AuthenticationController

Running Pittest for the first time gives us the following result:



The AuthenticationController class has the necessary REST endpoints, as well as the logic needed for authentication tasks like registering and logging in. Since authentication is a fundamental component of our app, we decided that it is important to reinforce this class in terms of testing as well as mutation killing. This the original feedback given by pittest:

```
    @Autowired
    public AuthenticationController(AuthenticationManager authenticationManager,
                                    JwtTokenGenerator jwtTokenGenerator,
                                    JwtUserDetailsService jwtUserDetailsService,
                                    RegistrationService registrationService) {
        this.authenticationManager = authenticationManager;
        this.jwtTokenGenerator = jwtTokenGenerator;
        this.jwtUserDetailsService = jwtUserDetailsService;
        this.registrationService = registrationService;
    }

    /**
     * Endpoint for authentication.
     *
     * @param request The login model
     * @return JWT token if the login is successful
     * @throws Exception if the user does not exist or the password is incorrect
     */
    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponseModel> authenticate(
            @RequestBody AuthenticationRequestModel request)
            throws Exception {

        try {
            authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(
                            request.getNetId(),
                            request.getPassword()));
        } catch (DisabledException e) {
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "INVALID_CREDENTIALS", e);
        }

        final UserDetails userDetails = jwtUserDetailsService.loadUserByUsername(request.getNetId());
        final String jwtToken = jwtTokenGenerator.generateToken(userDetails);
        return ResponseEntity.ok(new AuthenticationResponseModel(jwtToken));
    }

    @PostMapping("/register")
    public ResponseEntity register(@RequestBody RegistrationRequestModel request) throws Exception {
        try {
            NetId netId = new NetId(request.getNetId());
            Password password = new Password(request.getPassword());
            //role is default customer on this endpoint
            UserRole userRole = new UserRole("customer");
            registrationService.registerUser(netId, password, userRole);
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
        }

        return ResponseEntity.ok().build();
    }

    @PostMapping("/registerStoreOrManager")
    public ResponseEntity<String> registerStore(@RequestBody RegistrationSpecialRequestModel request) throws Exception {
        try {
            NetId netId = new NetId(request.getNetId());
            Password password = new Password(request.getPassword());
            if (!(request.getUserRole().equals("store") || request.getUserRole().equals("manager"))) {
                return ResponseEntity.badRequest().body("invalid role!");
            }
            UserRole userRole = new UserRole(request.getUserRole());
            registrationService.registerUser(netId, password, userRole);
        } catch (Exception e) {
            throw new ResponseStatusException(HttpStatus.BAD_REQUEST, e.getMessage());
        }

        return ResponseEntity.ok().build();
    }
```

```
    @GetMapping("/allStores")
    public ResponseEntity<String> getAllStores() {
        StringBuilder sb = new StringBuilder();
        sb.append("Below are all available stores:\n");
        for (AppUser store : jwtUserDetailsService.getAllStores()) {
            sb.append(store.getId()).append("- ").append(store.getNetId()).append("\n");
        }
        sb.append("In order to choose the store you wish to order from, please make the following POST request:\n")
              .append("http://localhost:8083/api/basket/setStore/{storeID}\n")
              .append("where storeID is the number next to store name on the list above.");
        return ResponseEntity.ok(sb.toString());
    }

    /**
     * GET endpoint that receives the customer input storeID and verifies if there exists a store with such storeID.
     *
     * @param storeId the customer input for preferred store
     * @return TRUE if there is, and vice-versa
     */
    @GetMapping("/verify/{storeID}")
    public ResponseEntity<Boolean> verifyStoreId(@PathVariable("storeID") int storeId) {
        Boolean result = false;
        for (AppUser user : jwtUserDetailsService.getAllStores()) {
            if (user.getId() == storeId) {
                result = true;
            }
        }
        if (result) {
            return ResponseEntity.ok(true);
        } else {
            return ResponseEntity.ok(false);
        }
    }
}
```

Mutations

84	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::authenticate → KILLED
106	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::register → SURVIVED
114	1. negated conditional → KILLED 2. negated conditional → KILLED
115	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::registerStore → KILLED
123	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::registerStore → SURVIVED
142	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::getAllStores → NO_COVERAGE
155	1. negated conditional → NO_COVERAGE
159	1. negated conditional → NO_COVERAGE
160	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::verifyStoreId → NO_COVERAGE
162	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::verifyStoreId → NO_COVERAGE

At first glance we see that the main mutation that this class is vulnerable against is replacing the returned response with null. Thus, a null check test was added for every method:

```
@Test
public void registerStoreOrManagerNullCheck() throws Exception {
    // Arrange
    final NetId testUser = new NetId("SomeUser");
    final Password testPassword = new Password("password123!");
    final HashedPassword testHashedPassword = new HashedPassword("hashedTestPassword");
    final UserRole testUserRole = new UserRole("fakeRole");
    when(mockPasswordEncoder.hash(testPassword)).thenReturn(testHashedPassword);

    RegistrationSpecialRequestModel model = new RegistrationSpecialRequestModel();
    model.setNetId(testUser.toString());
    model.setPassword(testPassword.toString());
    model.setUserRole(testUserRole.toString());

    // Assert
    assertThat(authenticationController.registerStore(model)).isNotNull();
}
```

```
@Test
public void getStoresNullCheck() throws Exception {
    AppUser store1 = new AppUser(new NetId("Delft"), new HashedPassword("test"),
            new UserRole("manager"), List.of(0L));
    AppUser store2 = new AppUser(new NetId("Rotterdam"), new HashedPassword("test2"),
            new UserRole("manager"), List.of(0L));
    List<AppUser> stores = new ArrayList<>(List.of(store1, store2));
    when(mockJwtUserDetailsService.getAllStores()).thenReturn(stores);

    assertThat(authenticationController.getAllStores()).isNotNull();
}
```

```java
@Test
public void registerNullCheck() throws Exception {
    // Arrange
    final NetId testUser = new NetId("SomeUser");
    final Password testPassword = new Password("password123!");
    final HashedPassword testHashedPassword = new HashedPassword("hashedTestPassword");
    final UserRole testUserRole = new UserRole("customer");
    when(mockPasswordEncoder.hash(testPassword)).thenReturn(testHashedPassword);

    RegistrationRequestModel model = new RegistrationRequestModel();
    model.setNetId(testUser.toString());
    model.setPassword(testPassword.toString());

    // Assert
    assertThat(authenticationController.register(model)).isNotNull();
```

Then, additional tests were added to the getAllStores() and verifyStoreId() methods, not only because these classes were a bit neglected testing wise, but also to kill the respective mutants that negated the conditionals.

```java
@Test
public void getStoresValid() throws Exception {
    AppUser store1 = new AppUser(new NetId("Delft"), new HashedPassword("test"),
            new UserRole("manager"), List.of(0L));
    List<AppUser> stores = new ArrayList<>(List.of(store1));
    when(mockJwtUserDetailsService.getAllStores()).thenReturn(stores);

    ResultActions resultActions = mockMvc.perform(get("/allStores")
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", "Bearer MockedToken"));
    resultActions.andExpect(status().isOk());
    MvcResult result = mockMvc.perform(get("/allStores")
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", "Bearer MockedToken")).andReturn();
    String responseBody = result.getResponse().getContentAsString();
    assertThat(responseBody).isEqualTo("Below are all available stores:\n" +
            "0- Delft\n"
            + "In order to choose the store you wish to order from, please make the following POST request:\n"
            "http://localhost:8083/api/basket/setStore/{storeID}\n" +
            "where storeID is the number next to store name on the list above.");
}
```

```java
@Test
public void verifyStoresValid() throws Exception {
    AppUser store1 = new AppUser(new NetId("Delft"), new HashedPassword("test"),
            new UserRole("manager"), List.of(0L));
    AppUser store2 = new AppUser(new NetId("Rotterdam"), new HashedPassword("test2"),
            new UserRole("manager"), List.of(0L));
    List<AppUser> stores = new ArrayList<>(List.of(store1, store2));
    when(mockJwtUserDetailsService.getAllStores()).thenReturn(stores);

    ResultActions resultActions = mockMvc.perform(get("/verify/0")
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", "Bearer MockedToken"));
    resultActions.andExpect(status().isOk());
    MvcResult result = mockMvc.perform(get("/verify/0")
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", "Bearer MockedToken")).andReturn();
    String responseBody = result.getResponse().getContentAsString();
    assertThat(responseBody).isEqualTo("true");
}
```

```java
@Test
public void verifyStoresInvalid() throws Exception {
    AppUser store1 = new AppUser(new NetId("Delft"), new HashedPassword("test"),
            new UserRole("manager"), List.of(0L));
    AppUser store2 = new AppUser(new NetId("Rotterdam"), new HashedPassword("test2"),
            new UserRole("manager"), List.of(0L));
    List<AppUser> stores = new ArrayList<>(List.of(store1, store2));
    when(mockJwtUserDetailsService.getAllStores()).thenReturn(stores);

    ResultActions resultActions = mockMvc.perform(get("/verify/23")
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", "Bearer MockedToken"));
    resultActions.andExpect(status().isOk());
    MvcResult result = mockMvc.perform(get("/verify/23")
            .contentType(MediaType.APPLICATION_JSON)
            .header("Authorization", "Bearer MockedToken")).andReturn();
    String responseBody = result.getResponse().getContentAsString();
    assertThat(responseBody).isEqualTo("false");
}

@Test
public void verifyStoresNullCheck() throws Exception {
    AppUser store1 = new AppUser(new NetId("Delft"), new HashedPassword("test"),
            new UserRole("manager"), List.of(0L));
    AppUser store2 = new AppUser(new NetId("Rotterdam"), new HashedPassword("test2"),
            new UserRole("manager"), List.of(0L));
    List<AppUser> stores = new ArrayList<>(List.of(store1, store2));
    when(mockJwtUserDetailsService.getAllStores()).thenReturn(stores);

    assertThat(authenticationController.verifyStoreId(0)).isNotNull();
    assertThat(authenticationController.verifyStoreId(23)).isNotNull();
}
```

After the additional testing we get this from pitest's report:

AuthenticationController.java	100%	52/52	100%	11/11

Mutations

84	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::authenticate → TIMED_OUT
106	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::register → TIMED_OUT
114	1. negated conditional → KILLED 2. negated conditional → KILLED
115	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::registerStore → KILLED
123	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::registerStore → KILLED
142	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::getAllStores → KILLED
155	1. negated conditional → KILLED
159	1. negated conditional → KILLED
160	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::verifyStoreId → KILLED
162	1. replaced return value with null for nl/tudelft/sem/template/authentication/controllers/AuthenticationController::verifyStoreId → KILLED

1.4: RestService (Order microservice)

As shown below, the RestService in the Order microservice was barely tested and therefore caught none of the mutants.

Pit Test Coverage Report

Package Summary

nl.tudelft.sem.template.order.services

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	17%	3/18	0%	0/8	0%	0/0

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
RestService.java	17%	3/18	0%	0/8	0%	0/0

Pit Test Coverage Report

Package Summary

nl.tudelft.sem.template.order.services

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	61%	11/18	50%	4/8	80%	4/5

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
RestService.java	61%	11/18	50%	4/8	80%	4/5

To solve this, a ResServiceTest class was created to test the methods in the RestService. This new class has five tests and ended up killing 4/8 mutants. The tests relate to both methods in the RestService:

```
no usages  Izzy van der Giessen
@Test
public void unlimitedCouponTest() {
    Coupon coupon = new Coupon();
    String token = "Token";

    ResponseEntity<String> res = restService.removeCoupon(new RemoveCouponRequestModel(coupon), token);
    HttpStatus h = res.getStatusCode();
    assertThat(h).isEqualTo(HttpStatus.OK);
}

no usages  Izzy van der Giessen
@Test
public void limitedCouponTest() {
    Coupon coupon = new Coupon( code: "CODE",  limitedTime: true);
    String token = "Token";

    ResponseEntity<String> res = restService.removeCoupon(new RemoveCouponRequestModel(coupon), token);
    HttpStatus h = res.getStatusCode();
    assertThat(h).isEqualTo(HttpStatus.BAD_REQUEST);
}

no usages  Izzy van der Giessen
@Test
public void limitedCouponTestWithMock() {
    RestTemplateBuilder restTemplateBuilder = mock(RestTemplateBuilder.class);
    RestTemplate restTemplate = mock(RestTemplate.class);
    when(restTemplateBuilder.build()).thenReturn(restTemplate);
    RestService rest = new RestService(restTemplateBuilder);
    Coupon coupon = new Coupon( code: "CODE",  limitedTime: true);
    String token = "Token";

    when(restTemplate.exchange(anyString(), any(HttpMethod.class), any(HttpEntity.class), any(Class.class))).thenReturn(ResponseEntity.ok( body: ""));
    ResponseEntity<String> res = rest.removeCoupon(new RemoveCouponRequestModel(coupon), token);
    String h = res.getBody();
    assertThat(h).isEqualTo( expected: "");
}
```

```
@Test
public void getWrongBasketTest() {
    assertThat(restService.getBasket( token: "Token").getStatusCode()).isEqualTo(HttpStatus.BAD_REQUEST);
}

no usages  Izzy van der Giessen
@Test
public void getMockBasketTest() {
    RestTemplateBuilder restTemplateBuilder = mock(RestTemplateBuilder.class);
    RestTemplate restTemplate = mock(RestTemplate.class);
    when(restTemplateBuilder.build()).thenReturn(restTemplate);
    RestService rest = new RestService(restTemplateBuilder);
    when(restTemplate.exchange(anyString(), any(HttpMethod.class), any(HttpEntity.class), any(Class.class))).thenReturn(ResponseEntity.ok( body: ""));

    assertThat(rest.getBasket( token: "Token").getStatusCode()).isEqualTo(HttpStatus.OK);
}
```

Task 2: Manual Mutation Testing

Selected core domain: Basket Microservice

For task 2, the goal is to inject mutants manually in 4 critical classes within the Basket microservice, and kill them by creating test cases for each mutant. Essentially, this test case is able to “kill” these mutants by *passing* without the mutant, and *fail* when the mutant is in place.

This part of the report will proceed as follows for each case:

- Critical class
- The reason behind its criticality
- The injection of the mutant
- The killer test case

2.1: CouponRepoController

CouponRepoController is one of the most critical components of the microservice as it serves the connection between the coupon repository and the rest of the system. Essentially, its functionality is a must for any sort of presence of data related to coupons.

The CouponRepoController’s *getCoupon()* method is a GET-endpoint to retrieve a coupon’s details being passed the code. It’s critical for the coupon-application ability of the customer. The tests have shown that the last if-clause is not properly tested, since when the line is replaced by `if(coupon != null)`, all tests pass.

The mutation is not only in the line that has the if-statement, but also in the line below it, since changing the condition within the if-statement changes the else case too. Therefore, 2 tests are created for cases where the **coupon** variable is null and not-null. (BAD_REQUEST is expected when the **coupon** variable is null, and OK is expected when it's not-null). The link to the commit can be found here.

2.2: CouponService

In addition to searching for null coupons, we thought that it is good to test if adding null coupons to the database works as expected. We chose CouponService class, which is a service class that is more focused on handling the coupons. It communicates with the coupon database directly, and therefore plays a crucial part on whether the functionalities related to coupons can provide the desired service. The tested method is **couponInvalid()**: it checks if the activation code of the coupon follows the required format. This method is critical because if this method returns an incorrect result, then the coupons that are saved into the repository will be faulty. Also, the customers will not be able to apply the coupons they wish to.

The mutant that was injected was removing the exclamation mark for the regex matching.

```
public boolean couponInvalid(Coupon c) { return !c.getCode().matches( regex: "(?i)^[A-Z]{4}[0-9]{2}$"); }
```

This would make the method return false for data that is invalid. We thought that this was a mistake that can be made, because this method has to return **true** for **invalid** methods, which is not intuitive; a regular checker would return false for an invalid input. The commit regarding this mutation can be found here.

2.3: BasketController

BasketController is responsible for providing the customer with total functionality and control over their basket. One of the crucial requirements for the basket is setting the store to order from. This functionality is dealt with *setStorePreference()* method of the controller.

```
POST endpoint to set the preferred store to be ordered from. Verifies the validity of the input
storeID by calling user
Params: storeId – the ID of the preferred store
Returns: Message OK or BAD_Request
@PostMapping("/setStore")
public ResponseEntity<String> setStorePreference(@RequestBody int storeId) {
    if (!restService.verifyStoreId( storeId: storeId)) {
        return ResponseEntity.badRequest().body( body: "Invalid storeID. Please try again.");
    } else {
        return ResponseEntity.ok( body: basketService.setStorePreference( customerId: authManager.getNetId(), storeId: storeId));
    }
}
```

This method includes an if statement that selects the proper response entity based on the validity of the customer-input storeID. This clause is a part of the method that is not yet tested and can cause great harm (ordering from non-existing stores) if it doesn't go as planned. A potential mutation would be for the ! (not) to be removed, which would return an "Invalid storeID…" message for existing stores, and worse, would place orders to non-existing stores. In order to check this, a killer test is placed that expects a BAD_REQUEST only if the restService.verifyStoreId() actually responds with a **false**. The test fails with mutation in place, and passes when mutation is not present. The link to the test can be found here.

2.4: PizzaService

PizzaService is another service class that handles the Pizzas. Similar to CouponService, this class manages and validates the pizza database. The mutated method is invalid(), which is another checker method that verifies a given Pizza. This is crucial to have for the stores, because it makes sure that the customers do not see any problematic data instances.

The mutation that was made was changing logical OR to bitwise OR operators:

```
public Boolean invalid(Pizza pizza) {
    return exists(pizza) | pizza == null | pizza.getName().isEmpty() | pizza.getIngredients().isEmpty();
```

The program fails for the mutated method if pizza is null, because after obtaining true for "pizza == null", the method still tries to run the third comparison and throws NullPointerException. We thought that this can also be a possible issue that can be addressed, because although the two statements are logically equal, they are treated differently by the compiler. This is also very critical as it is only observed during run-time, leading to program failure. The commit regarding this mutation can be found here.