

# MealController

Initial mealcontroller mutation score test: 63%

The first optimization we can make is to remove the method call to `checkProductIdList()`. We can see that this method does not provide additional functionality, as all products are also verified in the `checkAvailablePortions()` method. That is also why the mutant that removed this method call survived, as it did not change the outcome of the program. We call such mutants “equivalent mutants”. Removing this unnecessary method call improved the mutation score from 63% to 71%. The commit to this change can be found [here](#).

Next, in the method `checkAvailablePortions()`, there is a conditional statement that checks whether or not there are enough portions to accommodate the portions each user needs. From the mutation test report, it appears that we did not have boundary tests on this, so changing the `<` to `<=` did not change test results. This has been resolved by adding a boundary test. We created a new product containing 11 portions and created a meal consisting of 3 persons each consuming 4 portions of the product. We also modified an existing test case where two persons consume 5 portions from a 10-portions product. This improved mutation score from 71% to 86%. The commit implementing this change can be found by following [this link](#).

PIT reports one more mutant not being killed. This mutant changed the message that was passed with an exception. However, this same calculation was done in an `if`-statement right above. So, we could easily kill this mutant by putting this calculation in a variable instead of calculating it twice inline. This improvement brought the mutation score up to 100%. This small adjustment can be found in [this commit](#).

Mutation score after modifications: 100%

# PortionServiceImpl

Initial mutation score: 0%

Commit:

<https://gitlab.ewi.tudelft.nl/cse2115/2020-2010/7-student-house-food-management/op27-sem54/-/commit/db5567f2732568b3900659cacc8c1fdb4d9bd5d>

The `nl.tudelft.sem.sem54.fridge.service.PortionServiceImpl` class was untested, and so we introduced 4 tests to thoroughly test the method. Included was full branch coverage of the method. For testing the loop, we used the strategy of doing the loop zero times, one time, and multiple times.

Current mutation score: 100%

## ProductServiceImpl

Initial mutation score: 0%

Commit:

<https://gitlab.ewi.tudelft.nl/cse2115/2020-2010/7-student-house-food-management/op27-sem54/-/commit/68bcdc965a493529188fb8c1d7073a92da9c408c>

The `nl.tudelft.sem.sem54.fridge.service.ProductServiceImpl` class was untested, and so we introduced 15 tests to thoroughly test the different methods. Included was boundary testing (see `findById_lessZero` on line 71) and full branch coverage of each method as well. After the first pass of writing tests, only one mutant remained unkilld in the `save` test on line 42, which was then killed in the [final commit](#).

Current mutation score: 100%

## ProductTransactionServiceImpl

Initial mutation score: 0%

Commit:

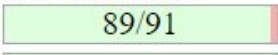
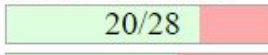
<https://gitlab.ewi.tudelft.nl/cse2115/2020-2010/7-student-house-food-management/op27-sem54/-/commit/70873af39368aa81831fd27ec11e6744a415ee99>

The `nl.tudelft.sem.sem54.fridge.service.ProductTransactionServiceImpl` class was untested, and so we introduced 3 tests to thoroughly test the class. Included was full branch coverage of the methods. We also tested for certain exceptions to be thrown.

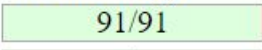
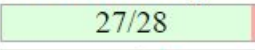
Current mutation score: 100%

# FridgeController

Before:

Name	Line Coverage	Mutation Coverage
<a href="#">FridgeController.java</a>	98%  89/91	71%  20/28

After:

Name	Line Coverage	Mutation Coverage
<a href="#">FridgeController.java</a>	100%  91/91	96%  27/28

With the FridgeController class being a relatively important chunk of our application, we were obliged to improve its mutation coverage. To do that, we added some new tests and also improved the quality of existing tests. We covered some edge cases that we skipped out initially. Our existing tests of the web layer were improved by adding new assertions and increasing the functionality of our mocked classes (`@Spy` instead of `@Mock`). This greatly improved our testing coverage. We also stumbled upon one equivalent mutant while improving our test cases. Multiplication is replaced with division but in our case that does not give us any useful information. (The red line should be 200)

199	1	<code>.setCredits(product.getCreditValue())</code>
200		<code>.setPortions(portionsUndo * (-1))</code>
191	2.	removed call to nl/tudelft/sem/sem54/fridge/domain/Product::setPortionsLeft → KILLED
199	1.	Replaced integer multiplication with division → SURVIVED
205	1.	removed call to nl/tudelft/sem/sem54/fridge/domain/ProductTransaction::setRevert → KILLED

# FridgeServiceImpl

Initial mutation score: 0%

Current mutation score: 100%

Running our mutation tool against our suite we noticed that previously unproblematic code could indeed be dangerous. A few of our previously untested classes turned out to have a low mutation score. `FridgeServiceImpl` was one of them. We added tests that covered all branches and cases of the service.

# PermissionService

Initial mutation score: 0%

Current mutation score: 75%

Another problematic class was the `PermissionService`. It is a very important class, another result of our previously mentioned refactoring. For improving the mutation score of this class we added a new test suite with an almost complete line coverage (93%). A lot of these tests proved quite useful, despite them doing more fool-proofing of our code than adding tests that cover unnoticed edge cases. The most important part is that they make sure any personal assumptions we've made about our code will be asserted in the test suite. Those are things that either we will surely forget or other people (working on the project) would not know.

Name	Line Coverage	Mutation Coverage
<a href="#">FridgeServiceImpl.java</a>	0% <div>0/8</div>	0% <div>0/4</div>
<a href="#">LoggedInUserService.java</a>	100% <div>7/7</div>	100% <div>2/2</div>
<a href="#">PermissionService.java</a>	0% <div>0/15</div>	0% <div>0/4</div>

Name	Line Coverage	Mutation Coverage
<a href="#">FridgeServiceImpl.java</a>	100% <div>8/8</div>	100% <div>4/4</div>
<a href="#">LoggedInUserService.java</a>	100% <div>7/7</div>	100% <div>2/2</div>
<a href="#">PermissionService.java</a>	93% <div>14/15</div>	75% <div>3/4</div>