Software Test Documentation: Test Plan

for

Metu NCC Library Management

**Prepared by Bartu Can PALAMUT 2386225**

**Doğukan AKDAĞ 2452928**

**Ataberk Türk ATAĞ 2015915**

**Eren ÖZTÜRK 2385656**

**Middle East Technical University**

**Computer Engineering**

**Supervised by İdil CANDAN**

**07.04.2024**

Contents

[1 Introduction 2](#_gjdgxs)

[2 Test items 2](#_30j0zll)

[3 Features to be tested 2](#_1fob9te)

[4 Features not to be tested 2](#_3znysh7)

[5 Approach 3](#_2et92p0)

[5.1 Unit Testing 3](#_tyjcwt)

[5.1.1 Test Procedure 3](#_z3pfpxe4o9sh)

[5.1.2 Test Cases 3](#_7meamium7ph)

[5.2 Integration Testing 5](#_3dy6vkm)

[5.2.1 Test Procedure 5](#_a5gy9xpfg1dv)

[5.2.2 Test Cases 6](#_xeswqb7s41sm)

[5.3 System Testing 8](#_1t3h5sf)

[5.4 Performance Testing 9](#_4d34og8)

[5.5 Usability Testing 11](#_2s8eyo1)

[6 Item pass/fail criteria 11](#_17dp8vu)

[7 Schedule 12](#_3rdcrjn)

[7.1 Milestones and Tasks 12](#_26in1rg)

[7.2 Gantt Chart 14](#_lnxbz9)

[8 References 14](#_35nkun2)

# Introduction

The software product is a cross-platform application that is designed for management of a library, especially for the METU NCC. It serves as a centralised platform for managing library resources, rooms, and Copy Card purposes.

Outline of Functions:

* Book Management
  + Add, edit and delete books and ebooks.
* Room Management
  + Add, edit and delete rooms.
  + Reserve a timeslot of rooms.
* Course Management
  + Add, edit and delete course.
  + Add and delete course material to/from the course.
  + Add and delete students to/from the course.
* Copy Card Management
  + Add receipt for copy card balance.
* Registration Process
  + Create an account with METU mail.

# Test items

* Book Service
* Book Queue Service
* Course Service
* Room Service
* Room Reservations Service
* Registration Service
* Copy Card Service
* Recommendation System

# Features to be tested

We are planning to test services related to Book, Book queue, Course, Room, Room reservations, Registration, Copy Card, and Recommendation system. We are planning to divide that features for different test methods. For example, book, book queue, room reservations and registration will be tested in the integration test section. However, creating room, course and copy card features will be tested in the unit test section. In the system testing part, we will test all features we tested in unit test and integration test, and recommendation system. In the performance testing part, we will test some list operations with a large number of users to check system performance. Lastly, we will do a usability test with the SUS method with real end users.

# Features not to be tested

We are planning to not test Spring Security because that library is not our focus and tested by Spring already. Also, we are planning to not test Login operation because it is easy login operation. Moreover, we are planning to not test external services like Google API and OpenLibrary API because that services’ maintenance is not our responsibility.

# Approach

## Unit Testing

### Test Procedure

We will do unit testing on functions related to room, course, and copy card features on the backend. We decided to use JUnit and JaCoCo for testing purposes. JUnit is a framework that helps us to write more reliable and efficient tests. JaCoCo is a framework that generates code coverage reports. First, we selected Doğukan as a tester for unit testing. We identified the test cases and then we gave the test cases to Doğukan. We prepared our test cases in detail. Each test case has detailed steps’s explanation, and test data, so Doğukan can prepare unit tests for each test case easily. Doğukan prepared our tests independently of other units by using mocked or stubbed dependencies. This prevents interference from external factors and facilitates easier debugging. Also, Doğukan put assertions to the tests to check and validate expected behaviour of each test. Moreover, Bartu prepared a Continuous Integration (CI) pipeline by using GitHub Action to automate test execution on each commit on the master branch in GitHub repository. Thus, for each new development, every commit merged to the main branch is tested, and if the tests are successful, the microservice is automatically deployed to the production environment.

### Test Cases

| ID | Description | Steps | Test Data | Pre-condition | Expected Output |
| --- | --- | --- | --- | --- | --- |
| 1 | Trying to create a room with valid request body | 1-Call the createRoom method  2-Assert that method doesn’t return null | valid CreateRoom request body | User should be authenticated with LIBRARIAN role | StatusDTO Object which contains “S” success message |
| 2 | Trying to create a room with invalid request body | 1-Call the createRoom method  2-Assert that method throws exception code | invalid CreateRoom request body | User should be authenticated with LIBRARIAN role | Invalid Request Exception |
| 3 | Trying to generate QR code for a room with valid room id | 1-Call the generateQRcodeForRoom method  2-Assert that method doesn’t return null | valid room id | User should be authenticated with LIBRARIAN role | StatusDTO Object which contains “S” success message |
| 4 | Trying to generate QR code for a room with invalid room id | 1-Call the generateQRcodeForRoom method  2-Assert that method throws exception | invalid room id | User should be authenticated with LIBRARIAN role | Room Not Found Exception |
| 5 | Trying to create course with valid request body | 1-Call the createCourse method  2-Assert that method doesn’t return null | valid CreateCourse request body | User should be authenticated with LECTURER role | StatusDTO Object which contains “S” success message |
| 6 | Trying to create course with invalid request body | 1-Call the createCourse method  2-Assert that method throws exception code | invalid CreateCourse request body | User should be authenticated with LECTURER role | Invalid Request Exception |
| 7 | Trying to update course with valid id and request body | 1-Call the updateCourse method  2-Assert that method doesn’t return null | valid course id and CreateCourse request body | User should be authenticated with LECTURER role | StatusDTO Object which contains “S” success message |
| 8 | Trying to update course with invalid id | 1-Call the updateCourse method  2-Assert that method throws exception code | invalid course id | User should be authenticated with LECTURER role | Invalid Request Exception |
| 9 | Trying to update course with invalid request body | 1-Call the updateCourse method  2-Assert that method throws exception code | invalid CreateCourse request body | User should be authenticated with LECTURER role | Course Not Found Request Exception |
| 10 | Trying to get courses | 1-Call the getCourses method  2-Assert that method doesn’t return null |  |  | CourseDTO List |
| 11 | Trying to upload CoureMaterial | 1-Call the uploadCourseMaterial method  2-Assert that method doesn’t return null | Multipart File, course id, file name | User should be authenticated with LECTURER role | StatusDTO Object which contains “S” success message |
| 12 | Trying to get CourseMaterial | 1-Call getCourseMaterial method  2-Assert that method doesn’t return null | course material id |  | CourseMaterial Object |
| 13 | Trying to add student to a course | 1-Call addStudentToCourse method  2-Assert that method doesn’t return null | student id, course id | User should be authenticated with LECTURER role | StatusDTO Object which contains “S” success message |
| 14 | Trying to upload receipt | 1-Call createReceiptHistory method  2-Assert that method doesn’t return null | image id |  | StatusDTO Object which contains “S” success message |
| 15 | Trying to upload receipt with invalid image id | 1-Call createReceiptHistory method  2-Assert that method throws Image Not Found exception | image id |  | Image Not Found Exception |
| 16 | Trying to approve receipt | 1-Call approveReceipt method  2-Assert that method doesn’t return null | receipt id,  balance | User should be authenticated with LIBRARIAN role | StatusDTO Object which contains “S” success message |
| 17 | Trying to get receipts | 1-Call getReceipt method  2-Assert that method doesn’t return null |  | User should be authenticated with LIBRARIAN role | ReceiptDTO List Object |

## Integration Testing

### Test Procedure

We will do integration testing on APIs related to book, room reservations, and registration features on the backend. We decided to use JUnit and JaCoCo for testing purposes. JUnit is a framework that helps us to write more reliable and efficient tests. JaCoCo is a framework that generates code coverage reports. First, we selected Bartu as a tester for integration testing. We identified the test cases and then we gave the test cases to Bartu. We prepared our test cases in detail. Each test case has detailed steps’s explanation, and test data, so Bartu can prepare integration tests for each test case easily. Firstly, Bartu configured the H2 database for integration tests. H2 is a relational database management system written in Java. It runs on memory, so at the end of the tests, all data will be deleted. Then, Bartu created some data for the database at the beginning of the integration tests to use in the tests. Like unit testing, in the integration testing Bartu used independent units. For example, for not sending email, Bartu mocked mail service. Also since Bartu used the transactional feature of Spring Boot, after each test run, the data generated in that test is not permanently committed to the database. The data created for the database initialization phase actually works before all tests work. This created data is committed to the H2 database. In this way, the data we will use for each test case is created, we perform our test, and after the test case is completed, the data that was created by the services is deleted without being permanently committed to the database. This tests also ran by Continuous Integration (CI) pipeline that has been explained in the unit test part[⁽⁵˙¹⁾](#_tyjcwt).

### Test Cases

| ID | Description | Steps | Test Data | Pre-condition | Expected Output |
| --- | --- | --- | --- | --- | --- |
| 1 | Trying to register with metu mail. | 1-Call registration endpoint. | Registration informations that contains email = “e238622@metu.edu.tr” | The email must not be exist in the database. | Successful message. HTTP Status code = 200 |
| 2 | Trying to register with mail that is not metu mail. | 1-Call registration endpoint. | Registration informations that contains email = “bartucankk@gmail.com” | - | “Only metuians can register this application.” message. HTTP Status code = 500 |
| 3 | Trying to create a book process. | 1- Check the database of how many books exist.  2- Call create book endpoint.  3- Check the database book is created. | Request body that contains sample book information. | - | Successful message. HTTP Status code = 200 |
| 4 | Trying to create a book process with invalid informations | 1- Check the database of how many books exist.  2- Call create book endpoint.  3- Check the database book is not created. | Request body that contains sample book information. | - | “Invalid Request” error message. HTTP Status code = 500 |
| 5 | Trying to borrow a book to a user. | 1- Check if the book is not available in the database.  2- Call borrow book endpoint with book id and user id.  3- Check if the database book is not available. | Request body that contains sample book id and sample user id. | Database should have at least one user and book. | Successful message. HTTP Status code = 200 |
| 6 | Trying to borrow a book that not available and user is not in the queue. | 1- Check if the book is not available in the database.  2- Call borrow book endpoint with book id and user id.  3- Check if the database book is not available again.  4- Check if the book is not available again in the database  5- Check database book is not borrowed to user. | Request body that contains sample book id and sample user id. | Database should have at least two user and one book. Book should be borrowed to another user. That user should be not in the queue | “BOOK\_NOT\_AVAILABLE” error message. HTTP Status code = 500 |
| 7 | Trying to enter queue for borrowed - not available - book | 1- Check if the book is not available in the database.  2- Call enqueue book endpoint with book id and user id.  3- Check if the database book is not available again.  4- Check if the book is not available again in the database  5- Check database user is enqueued for the book. | Request body that contains sample book id. | Database should have at least two users and one book. Book should be borrowed to user-1. Second user, user-2, should be not in the queue. | Successful message. HTTP Status code = 200 |
| 8 | Trying to take back a book from the user. | 1- Check if the book is not available in the database.  2- Call take back book endpoint with book id and user id.  3- Check the database. Book should be reserved for the next person in the queue. | Request body that contains sample book id and sample user id. | Database should have at least two user and one book. Book should be borrowed by user 1. And second user should be in the queue for that book. | Successful message. HTTP Status code = 200 |
| 9 | Trying to make a reservation for a room slot. | 1- Check the database to get a sample room slot id that will be used for step 2.  2- Call make reservation API endpoint.  3- Check the database reservation is created. | Room and its time slots. | Room should be created. Request users must not have any reservation. | Successful message. HTTP Status code = 200 |
| 10 | Trying to make a reservation for a room slot but s/he has already two reservation. | 1- Check the database to get a sample room slot id that will be used for step 2.  2- Call make reservation API endpoint.  3- Check the database one of the reservation is not created. | Room and it's time slots. | Request user should have two reservations. | “Max Reservation Reached” error message. HTTP Status code = 500 |

## System Testing

Without using any extra tools or methods, we personally tested the system. First, we created certain test scenarios, which we subsequently sent to our tester, Ataberk. The goal of system testing is to evaluate the complete integrated system in order to confirm that it satisfies the established criteria and functions as intended. As a result, we concentrated on evaluating the user and librarian as well as the mobile application. For every component, two system tests were created. Our Test Plan report contains specifics about these scheduled tests.

1. Book Create/ Book Borrow/ Takeback/ Queue
2. Room Create/ Room Reservation
3. Copy card receipt
4. Course Create/ Material Create/ Student Join/
5. Recommendation with machine learning

Scenarios:

1. At the Middle East Technical University campus in Northern Cyprus, İdil works as a librarian. She is supposed to be entering recently delivered books into the system, but since she doesn't know about the option that lets her enter books directly using their ISBNs, she enters the books by hand. After logging into her librarian account, she selects the side tab to view the book list and launches the 'new book' button at the bottom to begin manually inputting each book. She eventually realises that this approach takes a lot of time, so she asks if there's a quicker way and finds that she can input books with simply their ISBN digits. She eagerly inputs the ISBN number and presses the "fill" button to carry out the next step. If the ISBN number is input correctly, the system verifies the creation of the book; nevertheless, İdil observes that when some ISBN numbers are typed incompletely or erroneously, the system notifies her of the error.
2. After a hectic midterm period, Şükrü, a student at the Middle East Technical University campus in Northern Cyprus, is hoping to locate a book to read for some alone time. But he would rather not physically look for it at the library since he's tired. He launches the MLM app, enters into his account, and begins perusing the books the system has to offer. He uses the search function to see if the books he's interested in are currently on the library's shelf, and then he clicks the "borrow" button. He then makes his way to the library to retrieve the book, returning it on the appointed day.
3. Sükrü wants to check out another book after returning this one within the allotted time. He does, however, have to join a waiting list because every book he is interested in is already checked out. When Sükrü’s turn comes, the system emails him to let him know that he needs to pick up the book within a day or else he would lose his spot in line. In the meanwhile, Sükrü starts reading the e-book edition. To get the book, Sükrü returns to the library. But when the time comes to return it, Sükrü unintentionally misses the deadline, and the system notifies him via email that he must return the book once more.
4. Enver works as a librarian at the Middle East Technical University campus in Northern Cyprus. There are study rooms accessible for groups, however the photos of the participants must be input into the system. Enver clicks the three lines on the left to bring up the menu, login into his librarian account, and click on the "create room." In order to register the rooms in the system, he inputs the name of the room and uploads the photos he took. The system sends out a confirmation message if the rooms are accurately registered. If there is a problem, though, the system won't let the room get registered and will instead provide an error notice.
5. Kemal Kılıç, a student at the Middle East Technical University campus in Northern Cyprus, would like to utilise a library room with his buddies to do a project for class. He uses the accommodation reservation option on the main menu after logging into his MLM system account to do this. He completes the accommodation reservation by selecting one of the times the system provides for that particular day. Kemal and his buddies head to the reserved room and use the Room Connection button to scan the QR code on the door to obtain access to use it for the length they requested after receiving a confirmation message for the reservation.
6. Meryem has to credit a student's copy card because she works as a librarian at the Middle East Technical University campus in Northern Cyprus. But before anything else, she has to confirm that the money has been received. He selects 'Receipt Management' from the main menu in order to view the user-uploaded receipt. She loads the credit onto the copied card after verifying the payment.

## Performance Testing

We are planning to use JMeter for performance testing. JMeter helps us to make a large number of requests in a specific time interval. We first identified the test cases and tester. We selected the tester as Eren. Then we prepared test scenarios related to getting operations of book, room, and course features. Eren will prepare test scripts with JMeter according to the test cases. We did not make any automation for performance testing, so every week, Eren will run these scripts and make a report. We will use our performance testing results to compare our application’s services under different attempts. Furthermore, if our server cannot handle these number of attempts, we will make some performance improvements to get a reliable and fast system.

We conducted four test cases:

* We run get book service every 1 seconds with 5000 different requests.
* We run get room service every 1 seconds with 5000 different requests.
* We run get course service every 1 seconds with 5000 different requests.
* We run get recommendation service every 1 seconds with 5000 different requests.

| ID | Description | Steps | Test Data | Pre-condition | Expected Output |
| --- | --- | --- | --- | --- | --- |
| 1 | Run getBooks service every 1 second with 5000 different request | - | Production server environment | Some books should be created in the database. | Service can handle %80 of the request under 2 seconds. |
| 2 | Run getRooms service every 1 second with 5000 different request | - | Production server environment | Some rooms should be created in the database. | Service can handle %80 of the request under 2 seconds. |
| 3 | Run getCourses service every 1 second with 5000 different request | - | Production server environment | Some courses should be created in the database. | Service can handle %80 of the request under 2 seconds. |
| 4 | Run getRecommendation service every 1 second with 5000 different request | - | Production server environment. | Some books and some reviews for each book should be created in the database. | Service can handle %80 of the request under 10 seconds. |

## Usability Testing

We are planning to use SUS technique to carry out our Usability test. First we are planning to create SUS tests that include the features and ratings we want to measure. We are planning to focus on performance, accessibility and usability as the features that will be tested. After preparing our tests, we plan to make two to three users use and test our application for a week. After their testing ends, we will ask them to fill out the SUS tests on our laptops.

Expectations

We are expecting to score more than 70 as 70 is the threshold for being considered good usability for a mobile application.

# Item pass/fail criteria

We identified the pass / fail criteria different for each test level. However some criterias are taken into account together, such as that the unit test level’s and integration test level’s line coverage are taken in together since they run together.

At the Unit test level:

* All test cases completed successfully.
* At least %60 lines are covered by the tests.

At the Integration test level:

* All test cases completed successfully.
* At least %60 lines are covered by the tests.

At the System test level:

* All cases are completed successfully with a some number of minor defects.

At the Performance test level:

* All cases are completed successfully

At the Usability test level:

* SUS technique’ score should give us between 70-100.

At the Master test plan:

* All lower-level plans completed.

# Schedule

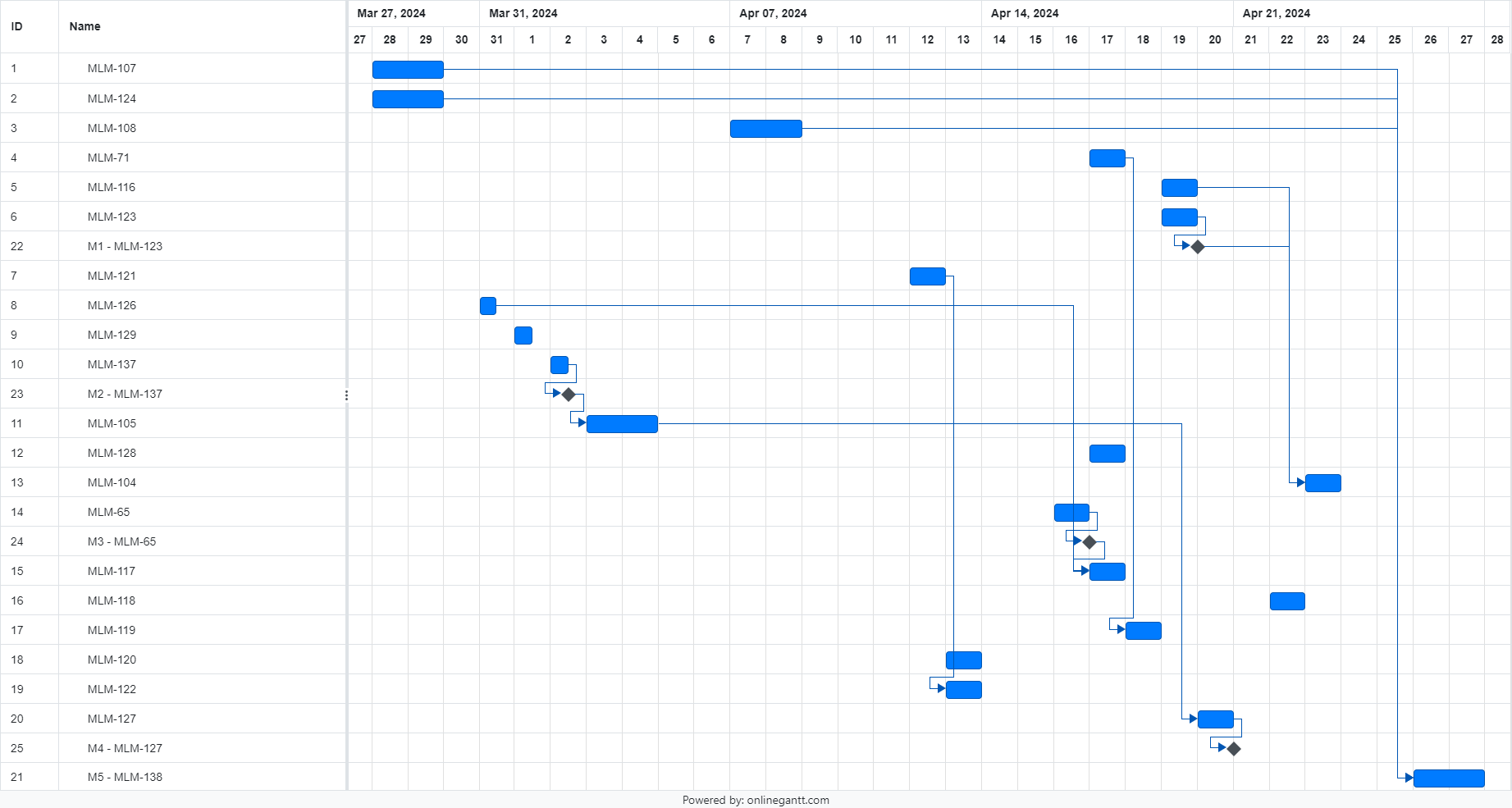
## Milestones and Tasks

*List the test milestones and tasks, along with their estimated durations in terms of weeks, task dependencies, and who is responsible for each task.*

| ID | Milestones | Estimated Duration (Week) |
| --- | --- | --- |
| M1 | MLM-123 / Backend - Book Recommendation System Java Side | 0.05 |
| M2 | MLM-137 / Preparing Pipeline | 0.05 |
| M3 | MLM-65 / Flutter - Copy Card Management Page | 0.05 |
| M4 | MLM-127 / Flutter - Course pages | 0.1 |
| M5 | MLM-138 / Preparing Document | 0.1 |

| Task ID | Task Name | Estimated Duration (Week / Hour) | Dependencies | Assigned | Status |
| --- | --- | --- | --- | --- | --- |
| MLM-107 | Preparing Unit Tests | 0,1 / 16 | N/A | Doğukan | DONE |
| MLM-124 | Preparing Integration Tests | 0,1 / 16 | N/A | Bartu | DONE |
| MLM-108 | Preparing Test Scenarios for System Testing | 0,1 / 16 | N/A | Ataberk | DONE |
| MLM-71 | Backend - Ebook Services | 0,05 / 8 | N/A | Doğukan | IN PROGRESS |
| MLM-116 | Backend - Book Recommendation System Python Side | 0,05 / 8 | N/A | Doğukan | IN PROGRESS |
| MLM-123 | Backend - Book Recommendation System Java Side | 0,05 / 8 | N/A | Bartu | IN PROGRESS |
| MLM-121 | Backend - Favourite List | 0,05 / 8 | N/A | Doğukan | TO DO |
| MLM-126 | Backend - User Identity services | 0,05 / 4 | N/A | Bartu | DONE |
| MLM-129 | Backend - Sending Email to User when book returned to give review | 0,05 / 4 | N/A | Bartu | DONE |
| MLM-137 | Preparing Pipeline | 0,05 / 4 | N/A | Bartu | DONE |
| MLM-105 | Backend - Course Development | 0,1 / 16 | N/A | Bartu | DONE |
| MLM-128 | Backend - Creating indetailed filter services for book | 0,05 / 4 | N/A | Doğukan | TO DO |
| MLM-104 | Research - Machine Learning for Recommendation System | 0,05 / 8 | MLM-123, MLM-116 | Ataberk | IN PROGRESS |
| MLM-65 | Flutter - Copy Card Management Page | 0,05 / 8 | N/A | Eren | TO DO |
| MLM-117 | Flutter - User Profile Page for Librarian | 0,05 / 4 | MLM-126 | Eren | TO DO |
| MLM-118 | Flutter - Book Reviews | 0,05 / 4 | N/A | Ataberk | TO DO |
| MLM-119 | Flutter - EBook Viewer | 0,05 / 8 | MLM-71 | Eren | TO DO |
| MLM-120 | Flutter- Room Confirmation Popup | 0,05 / 6 | N/A | Eren | TO DO |
| MLM-122 | Flutter - Favourite List | 0,05 / 4 | MLM-121 | Ataberk | TO DO |
| MLM-127 | Flutter - Course pages | 0,1 / 16 | MLM-105 | Ataberk | TO DO |
| MLM-138 | Preparing Document | 0,1 / 16 | MLM-107, MLM-108, MLM-124 | Ataberk,  Bartu,  Doğukan | WAITING for FEEDBACK |

## Gantt Chart

**

# References

*N/A*