SUPERVISOR MEETINGS

1 – LOGIC

2 – NONE

3 – COMBINE LIGIC

4 – RESEARCH

GANTT / KANBAN

RESEARCH – STRIPE/BANK STATEMENTS/CREDIT REPORT

Github

Week 1 – Logic

In the initial stage of our project during Week 1, our main focus was figuring out the Logic and how to assign people to group packages like 1k, 3k, or 5k and deciding the order in which they'd get money and pay it back over months. We were aware of these aspects but didn't initially consider eligibility checks. However, during a recent meeting, we realized the importance of incorporating checks to ensure that individuals meet certain criteria before registering on the platform. Although we had a general idea of making the platform accessible to everyone, the meeting highlighted the need for defining minimum requirements and maintaining a sense of realism. This discussion was crucial in refining our approach and ensuring that our platform remains inclusive yet practical for all users.

Week 2 – exams

In Week 2, our team faced the challenge of exams, resulting in the absence of a scheduled meeting. While exams are a valid factor, it's essential to acknowledge that they shouldn't be an excuse for a lack of synchronous communication and holistic project planning. Recognizing the need for improvement in future time management, we've realized the importance of maintaining continuous communication even during busy periods. Despite the absence of a meeting, we strategically split the group into two sub-teams. Three members concentrated on refining the logical aspects we discussed in Week 1, working individually on different components. Simultaneously, the other three focused on crucial research and planning for milestones, specifically targeting the implementation of features like bank statements, credit history report checks, and integration with the Stripe payment gateway.

Week 3 – combine logic / MVP / CONTINE RESEARCH

In Week 3, while our research efforts continued, a significant challenge emerged as the three logic components developed in Week 2 featured different variable names and were implemented in distinct programming languages—two in JavaScript and one in Java. Combining these diverse logics into a unified system became a source of annoyance, necessitating the alignment of variable names and coding practices. Despite this hurdle, the experience fostered a collective effort to streamline our coding practices, emphasizing consistency across the project. In response to this integration challenge, our focus shifted collectively towards discussions about the project's future, with a specific emphasis on defining the Minimum Viable Product (MVP) and outlining the project's trajectory. This unforeseen hiccup prompted valuable reflections on our coding practices and highlighted the importance of a more cohesive approach. Moving forward, we were committed to implementing standardized coding practices to ensure smoother integration and a more streamlined development process.

Week 4 – FEW and MVP

In Week 4, our team sustained its momentum by persisting with research on project milestones while concurrently directing efforts towards the active development of the front-end web. This involved the initiation of the website creation process using HTML and CSS, coupled with the commencement of backend development utilizing a NoSQL database. As part of our research discussions, we delved into significant issues, particularly concerning the credit report feature. A debate arose on whether to enable users to simply upload a PDF document or leverage a library to extract information from actual credit reports, and the practicality of randomly assigning salaries was considered. Additionally, we engaged in a constructive dialogue about the necessity of distinct user layouts—one for regular users and another for administrators with editing capabilities.

Week 5 – FEW + BE + Logic

In Week 5, our primary focus shifted towards integrating all components to establish a rudimentary Minimum Viable Product (MVP) before the upcoming reading week. The goal was to create a basic framework that we could submit as an initial MVP, allowing us to subsequently refine and extend functionalities during the following weeks. To achieve this, we concentrated on linking the front-end with the backend database, resulting in a dynamic website where interactive buttons directed users to different pages. Implementation of basic logic was a key aspect of this phase, ensuring that fundamental functionalities were operational. This concerted effort to bring all elements together marked a crucial step towards the creation of a functional MVP, setting the stage for further enhancements and extensions in the subsequent stages of the project.

Gitub – Unit Testing  
Throughout the development process, we leveraged GitHub as our central repository, meticulously recording progress and maintaining transparency within the team. Our commit history serves as a detailed chronicle of the evolution of our project, demonstrating an iterative approach to code development. Each logic component, from the initial planning in Week 1 to the challenging integration phase in Week 3, is well-documented in the repository, providing insights into the collaborative nature of our work. Unit testing was a key aspect of our development, with a dedicated folder showcasing comprehensive test cases. This not only ensured the functionality of individual components but also contributed to the systematic testing regime outlined in the mark scheme. Our commitment to effective error handling is evident in the code, where we implemented mechanisms to gracefully manage unexpected situations, contributing to the robustness of the system. The GitHub repository also captures milestones, including the linking of front-end and back-end components in Week 5, providing a tangible demonstration of our progress. This approach aligns with the principles of agile development, fostering an environment conducive to incremental changes.

A screenshot of a computer

Description automatically generated

**Unit Testing**

Throughout the development process, our team implemented a robust unit testing strategy, adhering to the principles of test-driven development (TDD)[1]. One pivotal component subject to rigorous testing was the eligibility logic, responsible for determining participants' suitability for different lending packages based on their monthly salaries. We employed JUnit, a widely-used testing framework in Java, and specifically used the JUnit Jupiter API[2] for writing our tests. This decision was based on our research and the proven effectiveness of the Jupiter API in the Java community. For instance, the code snippet below showcases tests for the £1k lending package eligibility, where we defined scenarios with monthly salaries above and below the eligibility threshold:

@Test  
void testEligibility1k() {  
 *assertEquals*("Eligible", calculateEligibility(120, 100));  
 *assertEquals*("Not Eligible", calculateEligibility(80, 100));  
}

[1] - “Test-Driven Development (TDD).” Agile Alliance, 2023, Accessed 16 Nov 2023.

[2] - [“JUnit 5.10.1 API.” JUnit, 2023,](https://junit.org/junit5/docs/current/api/) Accessed 16 Nov 2023.

Similarly, we extended this approach to other lending packages (£3k and £5k), ensuring a thorough examination of the eligibility logic's functionality. This meticulous testing methodology contributed to achieving a high level of confidence in the reliability of our eligibility assessment mechanism. In parallel, our team tackled the challenge of assigning individuals to specific lending packages and randomly distributing them across repayment months.

In the third week of our project, we embarked on a significant transition: consolidating our codebase from Java and JavaScript into a unified JavaScript environment. This decision was driven by the need for consistency and good coding practices. We used the above Java code as a base to transition into the JS code below. These can be viewed fully in our Git Repository.

Our team meticulously implemented a comprehensive unit testing strategy using the Mocha[3] testing framework, a popular choice for JavaScript applications due to its flexibility and ease of use.[4]

[3] - [“Mocha - the fun, simple, flexible JavaScript test framework.” Mocha, 2023,](https://mochajs.org/) Accessed 16 Nov 2023.

[4] - [“Mocha Unit Testing Tutorial: Getting Started - LambdaTest.” LambdaTest, 2023,](https://www.lambdatest.com/learning-hub/mocha-unit-testing)Accessed 16 Nov 2023.

Three core components were subjected to rigorous testing: eligibility logic, group package assignment, and random assignment mechanisms.

Eligibility Logic Testing: We crafted a series of test cases to validate the correctness of the eligibility logic, ensuring that individuals were appropriately categorized based on their monthly salaries. Each test case focused on specific scenarios, such as being eligible or not for the £1k, £3k, and £5k lending packages.[5]

[5] - [“How to Start Unit Testing Your JavaScript Code.” freeCodeCamp, 2023, 8](https://www.freecodecamp.org/news/how-to-start-unit-testing-javascript/). Accessed 16 Nov 2023.

// Eligibility Test Cases

describe('Eligibility', () => {

    it('should be eligible for £1k package if monthly salary is greater than 100', () => {

        const result = calculateEligibility(120, 100);

        assert.strictEqual(result.eligibility1k, true);

    });

Group Package Assignment Testing: The assignment of individuals to different lending packages was a critical functionality that we rigorously tested. We verified that eligible individuals were correctly assigned to the appropriate lending packages and that the rejection mechanism worked as intended.

// Group Package Assignment Test Cases

describe('Group Package Assignment', () => {

    it('should assign individuals to the £5k package if eligible', () => {

        const result = assignGroupPackages(createMockDataMap({ eligibility5k: true }));

        assert.strictEqual(result.packages['£5k\_1'].length, 1);

    });

Random Assignment Testing: The random assignment of unique indexes within each lending package group was a critical aspect to ensure fairness. Our tests validated the randomness and uniqueness of assigned indexes.

// Random Assignment Test Case

describe('Random Assignment', () => {

    it('should assign random and unique indexes to individuals within a group', () => {

        const indexes = getRandomIndexes(5);

        const uniqueIndexes = new Set(indexes);

        assert.strictEqual(uniqueIndexes.size, indexes.length);

    });

});

In summary, our unit testing strategy played a pivotal role in ensuring the robustness and reliability of our lending platform. Each test case was thoughtfully constructed with defined inputs, expected outputs, and a clear rationale. Our testing efforts rigorously validated critical components, including eligibility logic, package assignment, and random distribution. This process not only confirmed the correctness of our system but also facilitated swift identification and resolution of potential flaws.

The strategic integration of error-handling mechanisms enhanced the resilience of our system, fortifying it against potential anomalies and contributing to a seamless user experience. Additionally, our meticulous test documentation fostered a systematic testing regime, guiding the creation of appropriate test cases and offering a transparent justification for our testing methods. This documentation not only adhered to good coding practices but also facilitated critical evaluations of our system.

Lastly, a comprehensive evaluation involved representative stakeholders throughout formative and summative stages, ensuring that our platform not only functioned effectively but also met the needs and expectations of its users. This holistic approach significantly contributed to the overall success of our lending platform.