


LEARNING MANAGEMENT SYSTEM



GROUP 5

Esha

Scrum Master

Jijendran

Product Owner

Dhakshini

Quality Analyst

Yuhan

Risk Manager

Hui Yi

*User Experience
Specialist*

Ee Dhing

Minutes Taker



Project Inception - Version 3.0

Table of Contents

Table of Contents.....	2
Google Drive Link - FIT2101 Group 5.....	3
Agile Iteration II: The Project Plan.....	3
1.1 Project's Vision.....	3
1.2 Team Introduction.....	4
1.3 The Client.....	5
1.4 Stakeholder Analysis.....	5
1.5 Team Members, Contact Methods, and Roles.....	6
1.6 Team Process Model.....	8
1.7 Definition of Done.....	9
1.9 Task Allocation.....	10
1.10 Progress Tracking.....	11
1.11 Backlog Management.....	12
1.12 Time Tracking.....	12
1.13 Git Policy.....	13
2: Analysis of Alternatives.....	14
2.1 Summary.....	14
2.2 Terms of reference.....	15
2.2.1 Application type.....	15
2.2.2 Programming Language.....	16
2.2.3 Project Management Software.....	17
2.2.4 Database Management Tool.....	18
2.3 Body.....	19
2.4 Recommendations.....	23
3: Risk register.....	24
3.1 Risk Monitoring.....	24
3.2 Risk Mitigation.....	24
3.3 Risk Matrix.....	24
3.4 Live Risk Register.....	25
Appendix.....	28
Reference.....	29

Agile Iteration II: The Project Plan

1.1 Project's Vision

According to the structure that we developed during Sprint 1, our LMS is moving towards a complete learning system which alters the way teachers and learners collaborate in structured learning spaces. Better course structure, classroom tools and better user experience are added as part of the better course structure in this second version.

Improved Platform Architecture.

Sprint 2 also extends our LMS above and beyond the mere organization of lessons and introduces a complete learning hierarchy. Courses are now the primary container that stores the numerous lessons and each lesson could have a classroom in which to live teach. This design is based on the way teachers do their work and it provides the teachers with the means of teaching the lessons sequentially.

Extensive Course and Classroom Management.

Teachers have complete control of what they teach using our platform. They are able to create entire courses, prepare elaborate lessons and schedule classrooms that commence and finish at a given date. We included pre-requisite rules whereby students have to complete previous lessons before proceeding to more difficult lessons.

The lesson building screen has large text boxes, allowing teachers to type longer descriptions and objectives. The lesson list is also more readable, which gives the teachers easy time managing the lesson list, and adding content without much difficulty.

Enhanced User Interface and browsing.

Sprint 2 is aimed at the role-specific and user-friendly design. Teachers would be able to access lesson plans and student data quickly because when they log in, they can immediately see their courses. The students access the courses they are taking directly and hence getting what they need is easy.

Our color palette is plain black, blue and white color, which appears professional all over. It is also user- friendly even to individuals with varying technology skills.

Student and Teacher Advanced Interactions.

It is now possible to view all that teachers need in a single place. They can view live lists of students to be able to manage class well, and publish courses to enable students to get them at the appropriate time. pupils possess their page of lessons, on which they can follow their course and access the materials without any difficulties.

Infrastructure Improvements on the technical side.

We have added a common database to make sure that data is used in the same way across devices and users. This prevents the issue of data stagnating in a single location and retaining the experience at any point. We also prepared complete test reports to be able to know that the system is working and complies with all the rules.

Principles of Future Development.

After the expiry of Sprint 2, our LMS ceases to be a lesson manager but an entire educational system that represents all the teaching and learning levels.

In the following sprint, we would be focussing on the finishing features of the software to make it full fledged. This includes but is not limited to hosting, admin features, grading features for students and more.

1.2 Team Introduction

Ctrl + H + D



Team Technical Interests

35395117	evee0001@student.monash.edu	Esha Verayya	Scrum Master
35447001	jije0001@student.monash.edu	Jijendran	Product Owner
35555890	hooi0004@student.monash.edu	Ooi Hui Yi	User Experience Specialist
35097698	dsub0009@student.monash.edu	Dhakshini Subramanian	Quality Analyst
34474285	yche0705@student.monash.edu	Chen Yuhan	Risk Manager
34477861	etan0105@student.monash.edu	Tan Ee Dhing	Minutes Taker

1.3 The Client

Client Name	Mr. Chong Chun Yuan
Affiliation	Monash University
Contact details	Chong.ChunYuan@monash.edu (email)

Info about the context

This project is being developed for the Monash University FIT2101 - Software Engineering Process and Management unit.

1.4 Stakeholder Analysis

This section outlines the key stakeholders of the project, their interests, and their level of influence.

Stakeholders	Interest in this Project	Influence
Monash University	Overall alignment with university standards, security, and professional appearance.	High
F2101 Course Team	Ensuring the system is an effective educational tool and meets all assignment requirements.	High
Students (Users)	Usability, performance, and features of the system.	Medium to High

1.5 Team Members, Contact Methods, and Roles

Esha Verayya - Scrum Master/Integration Lead

- Facilitates sprint planning, ensures backlog is prioritized, aligns team goals with stakeholder expectations.
- Organizes sprint planning to break down features like “Add Class” or “View Timetable.”
- Helps clarify user stories and acceptance criteria, encourages collaboration between developers and product owners.
- Tracks progress via burndown charts, ensures smooth handoffs, and supports iterative releases.

Jijendran - Product Owner/Integration

- Acts as the voice of the customer, ensuring user needs are met.
- Collaborates with the team during sprint planning and reviews.
- Determines feature priorities and release planning.
- Refines user stories and tasks.

Ooi Hui Yi - UI/UX Specialist/Frontend Developer

- Collaborate with the Product Owner and team to understand user stories and acceptance criteria.
- Contribute to system design discussions, ensuring scalability.
- Leads UI/UX design decisions for lessons, courses, and classroom views.

Dhakshini Subramanian - Quality Analyst/Frontend Developer

- Collaborates on front-end implementation and code quality.
- Ensures documentation, user manuals, and agile processes are followed.
- Designs and executes test cases for front-end and back-end features.
- Check that all documentation requirements, design, user manuals are clear, complete, and consistent.

Chen Yuhan - Risk Manager/Backend Developer

- Implements back-end features and assists with database integration.
- Oversees risk management for system integrity and team conflicts.

Tan Ee Dhing - Database Lead/Backend Developer

- The person in charge of this is responsible for tracking each meeting, how long they last and the agenda of each meeting.
- Manages database architecture and integration with back-end logic.

While each team member has been assigned specific roles to streamline our workflow such as Scrum Master, Product Owner, UI/UX Specialist, Risk Manager, and Database Lead, while everyone also acts as a developer, contributing to coding and implementation tasks. During the project we will approach our project with a collaborative mindset. Responsibilities are not rigidly siloed, instead, we will actively support each other across tasks to ensure smooth progress and shared accountability. Whether refining user stories or facilitating discussions, our team emphasizes flexibility, mutual assistance, and active, responsible engagement to complete all aspects of the workload.

Communication

i. Communication Medium

- Our team has collectively chosen Whatsapp for communication, and Google Docs for documentation as required by the project guidelines.
- We have joined a collaborative workspace called jira where we can catch up with to-do,in-progress and backlog tasks.
- Furthermore,our teammates have agreed to be responsive and reply quickest to the messages shared on our common platform *Whatsapp*.
- If any of the team members do not respond within 4-5 hours, other team members would check up on their task progress through voice call.
- Additionally we have decided to meet up every week on a pre discussed date and venue to catch up on our assigned tasks

ii. Communication Timelines

- As discussed among our team members, the acceptable timings for communications are between 9 am to 5 pm, with flexibility for urgent matters. And we will be meeting every Monday 5-6 PM to catch up work on our project.
- Members should respond within a reasonable time at the earliest, ideally earlier than 4 hours. As deadlines approach, we expect team members to be more actively communicating so the expected timeliness of responding would also decrease accordingly.

iii. Communication Code of Conduct

- We understand that it is our duty and responsibility to communicate respectfully and professionally within the team to each other, and to value each member's contribution.
- We hope to address conflicts constructively, and to make sure that everyone in the team has an equal chance to voice their opinions and any disagreements without hesitation.
- It is our duty to also maintain professionalism while interacting with our mentors, TAs and clients by being receptive to feedback.

1.6 Team Process Model

During this semester, our team will follow the Agile Scrum process model, which is suitable for our project as it offers many benefits, such as faster product delivery, enhanced team productivity, and improved adaptability among team members. It enables us to continuously deliver small increments of working software while effectively managing complex projects and responding to constantly changing demands.

Our group used the Scrum model within a 2-week academic sprint to provide the increment of our Learning Management System (LMS). The Product Owner (Jijendran) prioritised the backlog and made acceptance criteria clear. The Scrum Master (Esha Veena) led sprint ceremonies and blockers were fixed. The increment was implemented by the development team (Dhakshini, Esha, Jijendran, Yuhan, Hui Yi, Ee Dhing), and Dhakshini executed tests, and carried out reviews. Developer roles were divided into Frontend (Dhakshini, Hui Yi) and Backend (Ee Dhing, Yuhan), and everyone should contribute to ensuring functional features and quality across the system.

In Sprint Planning, the Product Owner presented high-priority stories based on the client's requirements, along with the related epics. The team estimated each story with Planning Poker, agreed capacity according to availability and made promises to achieve the Sprint goal. Then we broke down the user stories into subtasks covering different aspects (UI design, backend integration, and testing) and assigned owners to each of the tasks.

How we will apply Scrum:

1. Sprints:

Our team will work in sprints, the core component of the Scrum framework, following the procedures of Sprint Planning, Stand-ups Meeting, Sprint Review, and Sprint Retrospective to

ensure the processes are well-managed and transparent. Each sprint will last two weeks, enabling us to deliver incremental working software in short and frequent cycles.

2. **Scrum Team:**

Our team will work as the Scrum team, adopting the three Scrum roles: one Product Owner, one Scrum Master and the Developers. Every team member will collaborate closely, aligning their efforts toward the shared product goal.

3. **Artefacts:**

We will use the main Scrum artefacts to maintain the transparency of work and tracking of process.

- **Product Backlog:** A prioritized list of all the user stories need to be developed based on its business value and risk. It will be maintained by the Product Owner, estimated by the team, and refined throughout the project.
- **Sprint Backlog:** The subset of the product backlog, which contains the user stories selected by the Development Team for the current sprint. And the team should commit to complete these items within the sprint.
- **Burndown Chart:** A visual representation of the estimated remaining work in the sprint. Our team will track the chart frequently and maintain a sustainable pace, avoiding burnout during development.
- **User Stories:** Short and simple descriptions of the working software features, told from the user's perspective. The user stories focus on delivering the value of the project to the user and ensuring every member involved can understand them.

Rule followed (DEEP+ INVEST)

1. Our Product Backlog is based on **DEEP**:

- **Detailed** properly high-priority items (login + CRUD) were detailed fully with acceptance criteria, whereas lower-priority items are less detailed.
- **Estimated** - top stories were estimated in story points at Sprint Planning.
- **Emergent** - backlog changes because new requirements emerge.
- **Prioritised** - The Product Owner prioritises the user stories.

2. **INVEST** is followed in our Sprint Backlog and stories too:

- **Independent** - Stories can be implemented independently.
- **Negotiable** - Requirements explained with PO at Sprint Planning.
- **Valuable** - Every story provides functionality that the user can see.
- **Estimable** - All stories measured in points.
- **Small** - Stories could be done in a single sprint.
- **Testable** - Acceptance criteria outlined definite tests to be completed.

1.7 Definition of Done

- ☐ No errors or bugs appeared in the feature code.
- ☐ Code must be written within the coding standards or guidelines.
- ☐ No unused codes remain in the feature.
- ☐ Feature testing must be done and it must be documented.
- ☐ Performance and load tests must be carried out.
- ☐ The feature must meet the requirements of the client.
- ☐ The feature must be integrated correctly and causes no errors or bugs.
- ☐ All team members tested the codes and passed the tests.
- ☐ Feature is successfully merged into Gitlab with proper commits and correct branches.
- ☐ CI/CD pipeline runs successfully post-merge (tests, builds, deploys).
- ☐ Feature flag or toggle is in place if needed.
- ☐ Project Management (PM) tools like Jira and Clockify are updated with the current status.

1.8 Weekly Schedule

1. Weekly Standup Meetings
Day & Time: Every Monday or Tuesday (5:00pm - 5:10pm)
Purpose: To discuss questions including what we have done before the standup meetings, what are we going to do after the meeting and what problems we faced during the work.
2. Weekly Meetings
Day & Time: Every Monday (5:10pm-5:30pm)
Purpose: To review project progress, address issues, and plan upcoming tasks.
3. Collaborative work times
Day & Time: Every Monday (5:30pm-6:30pm)
Purpose: To work together on project deliverables, share updates, and provide real-time feedback.
4. Individual work times commitment
Hours per week: 3 - 4 hours
Expectation: Each team member will complete assigned tasks independently and meet agreed deadlines.
Reporting: Updates will be provided during weekly meetings and via Whatsapp group chat.
5. Attendance and Communication
 - Members must notify the team at least **1 day in advance** if they cannot attend a scheduled meeting or session.
 - Emergencies should be communicated as soon as possible.

1.9 Task Allocation

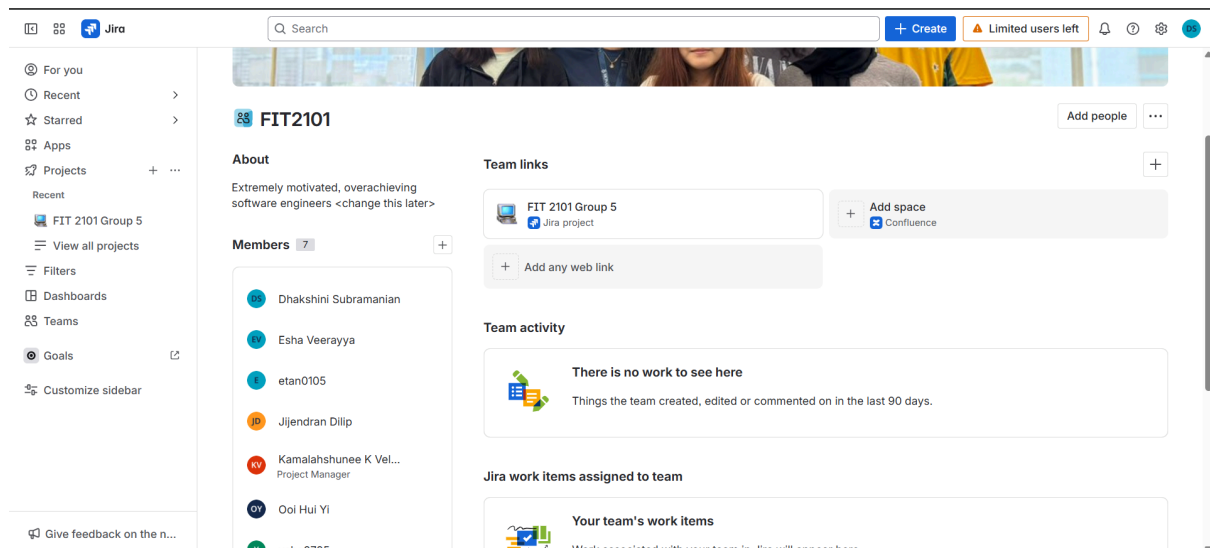
Our team will allocate the tasks collaboratively during the projects:

- At the start of each sprint, we'll break the user stories into smaller, more manageable tasks, which include **frontend development, backend development, and integration activities**.

Every team member can choose the tasks they feel most confident (based on their role), enabling the work to be completed more efficiently.

- For more challenging problems, two members can work together or seek assistance from other group members
- We'll try to keep the workload balanced by continuously monitoring progress through the Sprint Backlog, ensuring no one is overloaded or left idle.
- If the project requirements change during the sprint, the Product Owner will reprioritize the Product Backlog, and our team will adjust tasks accordingly in the next sprint.
- Each member should be responsible for the tasks they commit to, ensuring the quality of their output.

1.10 Progress Tracking



Our team is using Jira as the primary tool for progress tracking. A shared Jira workspace has been set up, which all team members can access. This workspace allows us to record, monitor, and update tasks across their different stages: To Do, In Progress, and Done.

Each task created in Jira is linked to the sprint backlog and assigned to specific members. Progress is visually tracked using Jira's boards and burndown charts, ensuring transparency across the team. During our weekly meetings (Mondays, 5–6 PM), we review the board together to confirm task statuses, identify any blockers, and reassign tasks if necessary.

By using Jira, our team ensures:

[JIRA workspace](#)

- **Accountability:** Every task has a clear assignee and deadline.

- **Visibility:** All members can view the current progress of the project at any time.
- **Collaboration:** Updates to tasks, comments, and dependencies are managed within Jira.
- **Proof of contribution:** The Jira workspace provides a record of who completed which tasks and when, which will be used to demonstrate progress and contributions.

This system ensures our project progresses in line with Scrum practices, allowing us to track work both individually and collectively throughout each sprint.

1.11 Backlog Management

Our team will use Jira and Clockify to store and manage backlogs each sprint. This is to ensure that our backlog consists of prioritisation, smooth task tracking and consistent maintenance of our work.

We will use Jira to manage the tasks and Clockify to track the amount of time each developer spends on their tasks. We can integrate both softwares together to add a timer to each of our tasks. We will create user stories and come up with corresponding tasks to get done with.

Prioritisation

- Product Owner will present the user stories selected for the sprint and agree with the team on their priority levels: high, medium, or low.
- Our team will use the MoSCoW method (Must have, Should have, Could have, Won't have) to guide prioritisation.
- We prioritise the high prioritisation features first so these features will be worked on development first.

Task tracking

- Estimate the time taken for each task and team members will update its status as work progresses.
- Every team member should work according to their role (Frontend, Backend, Integration), while actively contributing across tasks to maintain a balanced workload.
- Each feature must go through the Definition of Done Checklist.
- Backlog refinement meetings will be held every week to review and update the backlog as well as discuss the large issues and ways to solve them.
- Commit messages in GitLab will be used consistently to track code changes, updates, and bug fixes

1.12 Time Tracking

1. Tracking Method

- The team will use: Clockify
- Purpose: To record the time spent on each assigned task for accountability and workload monitoring.

2. Recording Frequency

- Each team member's log time: every time when working on the every sprints

3. Details to Include

- Total time spent

4. Review of Logged Time

- Logged time will be reviewed: weekly meetings
- Purpose: To monitor progress, balance workloads, and adjust schedules if necessary.

5. Accountability

- Missing or inaccurate logs will be addressed during team check-ins to maintain accurate project timelines.

1.13 Git Policy

I. Branch Structure

Our Branching Strategy is based on the Git Flow model which allows us to manage different stages of framework development.

- A. Main Branch: This branch holds code that should always be stable and deployable. Direct commits are forbidden to the main branch and it can only be updated through merges from the release branch.
- B. Develop Branch: This branch represents all the latest completed work into which the feature branches are merged into and created from
- C. Feature branches: Each new feature/user story must be developed in its own branch. It should have a naming convention of "feature/feature-name" and should always branch off from the latest develop branch. This branch is used to isolate new development from the main code till it is ready
- D. Release Branch: When preparing for a new release, this branch is created from the develop branch. This branch is used for final testing and bug fixes and only critical bug fixes are allowed. Once this is done the release branch is then merged to the main branch (and develop to maintain consistency)
- E. Hotfix Branches: These branches are created from the main branch for critical bug fixes that may have slipped through and are then merged back into the main branch (and develop branch to ensure the fix is propagated). The naming convention should be of "hotfix/urgent-fix"

II. Merge Requests

Merge requests are a key part of our reviewing process as they ensure code quality

- A. When a feature is completed, a merge request is created from said feature branch into the develop branch

- B. One of the team members (preferable someone who is more well versed in the area) is assigned as the reviewer who will go through the code and check for quality and style, logical errors, etc
- C. The request can only be merged after it has been approved by the reviewing team member

III. **Commit Messages**

Commit messages should be clear and consistent for the proper understanding of the project's history. A commit message should have the following structure

- A. The type of change made: Whether the change made was a new feature, a bug fix, document changes, test cases, code formatting
- B. The description of the change made: The changes made should be described in clear detail including the work on user stories and tasks, so that other members of the team can follow up on the changes made should the main developer not be available.

IV. **Handling merge conflicts**

Merge conflicts are common in a collaborative development work environment. To prevent this from happening the latest develop branch should be pulled from the remote repository before any changes are made. Should a merge conflict arise

- A. Resolving the conflict manually: The conflicting file is read through by the reviewer who will then decide which version of the code we want to keep or combine both the codes and stage the changes completing the merge process.
- B. The resolved conflicts are then pushed onto the remote repository

2: Analysis of Alternatives

2.1 Summary

We have analyzed three options for both the choice of programming language and platform based on 9 and 8 criteria respectively. The following provides a summary of our analysis:

Application options:

- Web application
 - Most accessible (runs in any browser, no need installation)
 - Cost effective, fast to develop
- Mobile application
 - High performance with device features
 - Expensive and OS dependent
 - Harder to update
- Desktop application
 - Stable and great for offline, heavy processing
 - OS- dependent
 - Slower to develop

In this case, the **best fit** is a web application for our purposes.

Programming language options:

- Python
 - Easy to learn
 - Team has experience
 - Many libraries
 - Hard to integrate frontend and backend for web, less direct web development
- JavaScript + [Node.js](#) + React
 - Strong integrations and widely used, a lot of resources
 - Steep learning curve
 - Team is less experienced
 - Setup is more complex
- HTML + CSS + JS
 - Beginner friendly, simple, fast to implement, works across browsers
 - Limited flexibility, weak backend, good for simpler apps

In this case, the **best fit** is a web platform built with HTML + CSS + JS because it is simple but enough for our needs in this project. Additionally, all members of the team have sufficient knowledge in order to overcome the learning curve in this case.

Project management tool options:

- Trello
 - Very quick setup and easy to use with a minimal learning curve, but lacks flexibility and advanced Agile structures.
- Jira
 - Steeper learning curve, but offers comprehensive Agile support, sprint planning, burndown charts, and strong integration capabilities.
 - The free plan covers up to 10 users with full functionality.

We selected **Jira** as the project management tool because it aligns with our Agile process, supports backlog tracking, reporting, and burndown chart generation, and provides all necessary features under its free plan.

Database management tool options:

- Firebase
 - Easy to use and flexible, but non-relational and harder for complex queries.
- Supabase
 - Relational (PostgreSQL), good for structured data and smooth integration with JavaScript, some learning curve but team has SQL experience.

We selected **Supabase** as the database management tool because it supports relational queries, integrates smoothly with the website applications, and is free to use for our project scope.

2.2 Terms of reference

The purpose of this analysis is to decide which platform and programming language are most suitable to use for this project. This decision requires the consideration of various aspects, including how users will access the platform, whether it can meet customer needs, its cost and platform performance. It is also important to identify any new programming skills that team members may need to master.

Furthermore this analysis also talks about the most preferred Project Management tool as well as database management system we will use throughout this project.

2.2.1 Application type

We have identified the following options to decide the most suitable **application type**:

1. **Web Application** – It runs on a server and is accessed directly through the web browsers, eliminating the need for users to download or install any software (Kinza Yasar, 2023).

2. **Mobile Application** – It runs on the mobile device, and must be downloaded and installed before use (Terrell Hanna & Wigmore, 2023) .
3. **Desktop Application** – It runs directly on the computer’s operating system and generally requires local installation, without relying on a web browser or internet connection to function (Rohn, 2022).

We will evaluate each option against the following criteria:

- **Accessibility** – Ensures users can easily access and interact with the platform across various devices and environments.
- **User Experience (UX)** – Ensures the overall satisfaction of users when interacting with the application.
- **Cost** – Evaluates the developing expenses, such as hosting, tools or any infrastructures.
- **Development Timeline** – Estimates how quickly the development can be completed and whether it can be delivered before the deadline.
- **Compatibility** – Ensures the application can function properly across different operating systems, browsers or devices.
- **Scalability** – Determines whether the platform can handle the growth of workloads, users and volume of data over time (Chiaramonte, 2024).
- **Maintainability** – Assesses the ease of application modification , update or fix when facing new requirements.
- **Performance** – Evaluates the efficiency and effectiveness of the application in running and delivering its intended functions to users
- **Licensing / Legal Compliance** – Ensures the application adheres to relevant licenses, industry standards , and legal regulations.

How important is each factor?

- High Importance – **Accessibility, Technical Expertise, Cost, Development Timeline, Compatibility, User Experience (UX)**
- Medium Importance – **Scalability, Maintainability, Performance**
- Baseline – **Licensing / Legal Compliance**

2.2.2 Programming Language

We have identified the following options to decide the most suitable **programming language**:

1. Python
2. JavaScript + [Node.js](#) + React
3. HTML + CSS + JavaScript

We will evaluate each option against the following criteria: (Kareliya, 2025)

- **Learning curve** - The ability to learn the languages in a short period of time
- **Overall experience with the language in the team** - How many people have encountered and coded in this language before
- **Ease of integrating the backend and frontend** - Whether or not we can connect the UI/UX to the backend functionality
- **Simplicity in relation to our goals in this project** - Making sure that there aren't unnecessary extra features that might add weight and extra complexity during the coding process
- **Available frameworks and resources online** - How widely used the language is and whether or not there are resources online that can be used to learn
- **Cross-platform support** - Whether or not it can work on a variety of browsers and devices

The most important of these factors are:

- Simplicity
- Learning curve
- Ease of integration

2.2.3 Project Management Software

We have identified the following options to decide the most suitable **project management software**: (Atlassian, 2025)

1. **Jira** - This is the industry standard for software projects, and contains functionalities like kanban boards, time tracking and burndown charts automated.
2. **Trello** - Simple and intuitive kanban boards and easy to set up

We will evaluate each option against the following criteria:

- **Ease of Use** – How quickly team members can learn the interface and begin managing tasks.
- **Collaboration Features** – Support for real-time updates, comments, and notifications.
- **Customization & Flexibility** – Options to use the software for our needs in a way that we are not forced to use features that we don't need.
- **Cost** – The breadth of the free tier

The most important of these factors are:

- Cost
- Ease of use

2.2.4 Database Management Tool

We have identified the following options to decide the most suitable **database management tool** for our project:

1. **Supabase** - An open-source relational database platform built on a PostgreSQL, which offers real-time capabilities and easy API integration.
2. **Firebase** - A cloud-based NoSQL database, and can integrate seamlessly with Google's ecosystem (Rafalski, 2024).

We will evaluate each option against the following criteria:

- **Learning curve** - How easy the developer team can learn and use the tool
- **Overall experience** - How familiar the team members are with the tool.
- **Cost**- Whether the tool need extra payment
- **Integration** - How easy to connect the database to the current system.
- **Type of database** - Relational (SQL) or Non-relational (NoSQL) and how it fits the project's data needs.

The most important of these factors are:

- Integration with frontend and backend
- Familiarity / learning curve
- Database type

2.3 Body

Comparison Table — Application Options

Criteria	Web Application	Mobile Application	Desktop Application
Accessibility	Accessible via any modern browser (such as Google Chrome, Safari), no installation needed (Indeed, 2024)	Requires download from app stores, OS(operation system) dependent	Requires local installation,OS dependent (Rohn, 2022).
User Experience (UX)	Responsive and consistent, but experience might be influenced by browser variations(Nicolaou, 2023)	Offers the most integrated and seamless native experience (Nicolaou, 2023)	Provides a stable and reliable experience on the installed device, but lacks cross-device flexibility
Cost	Most cost-effective	Most expensive	Moderate
Development Timeline	Faster due to mature frameworks and easy deployment	Faster due to mature frameworks and easy deployment	Slower due to OS-specific builds and installers
Compatibility	Cross-platform via browsers(Nicolaou, 2023)	Limited to mobile OS (iOS/Android)	Limited to chosen OS (Windows, macOS, Linux)
Scalability	Highly scalable, as the backend and frontend are separate, servers can be scaled easily.	Moderately scalable, since the backend can be scaled, but limited by device hardware	Less scalable, limited to the hardware of each individual computer
Maintainability	Best maintainability, with centralized updates applied instantly to all users (BrowserStack, n.d.)	Moderate maintainability, as updates require app store resubmission and approval (Indeed, 2024).	Lowest maintainability, with manual or installer-based updates

Performance	Good for general use, with the limitation due to internet dependency	High performance for graphics, sensors, and device-native functions	Excellent for heavy processing and offline capabilities
Licensing/Compliance	Easiest, follow standard hosting and data protection requirements	Hardest, must comply with app store policies and mobile regulations	Moderate, follow OS-specific compliance

Comparison Table — Programming language Options

Criteria	Python	JavaScript + Node.js + React	HTML + CSS + JS
Learning curve	Easy to learn because very similar to regular language	Hardest because the learning curve is steep and nobody has that much experience with JavaScript on this team	Moderate because HTML is easy to learn but then the integration with CSS and JS
Overall experience with the language in the team	Easiest, everyone has had experience with this language.	Hardest, because most people are not as familiar with these languages and coding style	Moderate, some people have had some sort of experience with HTML.
Ease of integrating the backend and frontend	Harder, needs extra frameworks like Django for backend and a separate frontend solution	Easiest since Node.js and React are designed to work together	Limited because although it works together with the backend, it can get messy when the project is more complicated.
Simplicity in relation to our goals in this project	Simple if the project needs data-heavy processing	Complex setup but powerful if project needs a dynamic web app	Simple for basic interactions
Available frameworks and resources online	High	Extremely high	High for UI - focussed work but limited when it comes to backend

Cross-platform support	Strong support - can run on any OS can be used for backend, data and scripting	Stronger support - works natively for web apps and backend with Node.js	Cross - platform for web browsers only, not suitable for mobile apps or backend by itself

(Kareliya, 2025)

Comparison table - Project management software options

Criteria	Jira	Trello
Ease of use	Steep learning curve, extreme functionality and capabilities of tracking and integration. Complex systems and processes. Flexibility is based on personalized set up.	Very quick initial set up and easy learning curve. Minimalistic and does not require expertise to set up and use. However, it is less flexible.
Cost	Jira free plan for up to 10 users with full functionalities.	Free plan has unlimited cards & members, some limitations relating to boards and workspaces, and limits on automations.

(Atlassian, 2025)

Comparison table - Database management tool options

Criteria	Supabase	Firebase
Ease of use	Moderate, similar to SQL, easy for those with database experience (Rafalski, 2024)	Easy for beginners, but some concepts may differ from SQL
Overall experience	Some team experience with PostgreSQL	Limited team experiences.

Type of database	Relational (PostgreSQL), which is good for structured data and complex relationships (Prins, 2024).	Non-relational (Firestore / Realtime DB), which is more flexible but can make complex queries harder (Prins, 2024).
Cost	Essentially free for basic usage.	Essentially free for basic usage.
Integration	Backend–frontend integration straightforward.	Integrates tightly with Google Cloud services, but less direct SQL-style querying.

2.4 Recommendations

After evaluating three application options based on the selected criteria, the web application seems the most suitable choice for our project. It scores the highest mark in the most important factors, such as accessibility, compatibility, user experience, cost and development time. Unlike mobile or desktop applications, a web application does not need installation and can be accessed directly through any modern browsers, which also reduces the limitations related to operating systems and devices. This supports both user convenience and our project's limited timeline. Additionally, the centralized update process of the web application improves its maintainability and scalability, ensuring the project can evolve with future requirements. Based on these considerations, our team recommends building a web application for the project.

Furthermore, considering the evaluation factors for the programming languages, such as the learning curve, required technical skills, ease of integration, simplicity of development, availability of frameworks, and the cross-platform support, we will use HTML, CSS, and JavaScript as the main programming languages. Compared with other options, it is more suitable for beginners, makes development more time-efficient, and widely supported through various platforms, ensuring smooth integration between frontend and backend. At the same time, it offers sufficient flexibility to implement the required functionalities which minimises the risks of switching languages during development.

We have decided to use Jira as our project management tool because of the extent of its customizability and its ability to handle our unique workload. The deciding factor between our other alternatives was Jira's ability to track what we need it to, i.e., time tracking and burndown chart automation. And, we selected Supabase as our database management tool. Since it provides a relational PostgreSQL database, which aligns with our project's structured data model (courses, students, and classrooms). Additionally, like Firebase, Supabase is essentially free for basic usage, making it a cost-effective solution.

In summary, after considering both the application type and programming language options, a web application built with HTML, CSS and JavaScript will be the most practical and efficient solution for the current project requirements. Additionally, we have decided on Jira to use as a project management software and integrate Supabase as our database platform.

3: Risk register

3.1 Risk Monitoring

Risk monitoring is important for us to identify how to avoid every risk in our project. We can identify and keep track of the risks to monitor the likelihood, severity, and impact of each risk. This will be executed throughout the project to keep track of continuous risk management.

By doing risk monitoring, we can make sure to prepare for the worst case scenario as well as prepare how to counter the risks when it is likely going to happen. This will ensure our team does not panic when the risks happen because we have already been monitoring and predicting the scenarios.

3.2 Risk Mitigation

Risk Mitigation involves a structured process to proactively handle potential issues that may be encountered in the duration of the project. Ultimately the goal would be to either prevent risks from happening or lessen the negative impact of them should they occur.

This involves a thorough identification of all the possible threats followed by an analysis to determine their likelihood of occurrence and severity of the impact. Based on this analysis, a system of responses are developed for each risk either to reduce its impact or accepting the risk and creating a contingency plan.

This plan is then implemented and monitored throughout the project allowing the team to handle risks, in the case they occur, ensuring that the project stays on track.

3.3 Risk Matrix

During this sprint, we used the following risk level matrix (Figure 2.1 Risk Level Matrix) to classify and evaluate the risks we identified in our project. The risk matrix considers two factors, the impact and the likelihood of the risk happening. By combining these two dimensions, each risk is assigned to a severity level (Low, Medium, High). And this helps us to identify which risks require more attention and stronger mitigation strategies.

3.4 Live Risk Register

This table lists the risks we faced during the development of the current sprint , and is ordered from high severity to low severity.

Risk ID	Date raised	Description	Estimated Impact	Likelihood	Severity	Owner	Status	Monitoring Strategy	Mitigation Plan	Contingency Plan
R1	N/A	Team member unavailable (due to illness or prolonged unresponsive ness)	High	Medium	High	Scrum master (Esha)	Not occurred (monitored)	Check the member's health condition every meeting and also constant check on each other's work progress	Encourage early task starts and cross-train members on each other's tasks	Redistribute unfinished work evenly among members and adjust deadlines of each tasks accordingly
R2	N/A	Delays due to unclear requirements	High	Medium	High	Product Owner (Jijdendran)	Not occurred (monitored)	Communicate regularly with client and review user stories	Double confirm the requirements before Sprint Planning	Re-prioritize the tasks in the next sprint.
R3	N/A	A critical bug discovered late in the project	High	Medium	High	Lead Developer (Jijdendran)	Not occurred (monitored)	Implement a structured testing plan and track and review all logged bugs daily	Follow coding standards, conduct reviews, and adopt continuous testing	Reassign team members from lower-priority tasks to bug resolution
R4	21 Sept 2...	Dependency on another task or team member that has fallen behind	High	Medium	High	Scrum Master (Esha)	Occurred – Solved	Host regular meetings to checkup on each individual's task to make sure they are proceeding	Document dependencies clearly and ensure communication between members	Reallocate effort to unblock critical dependencies and communicate project impact

Risk ID	Date raised	Description	Estimated Impact	Likelihood	Severity	Owner	Status	Monitoring Strategy	Mitigation Plan	Contingency Plan
		schedule						according to schedule		
R5	N/A	Data loss due to technical failure	High	Low	High	Scrum Master (Esha)	Not occurred (monitored)	Monitor the backup frequently	Regularly back up code and documents to cloud repositories	Recover from latest backup and reassign tasks if rework is needed
R6	N/A	Software crashes during development	High	Low	High	Scrum master (Esha)	Not occurred (monitored)	Monitor development environment stability	Use reliable softwares and test software beforehand	Restore code from backups or switch to alternative tools
R7	24 Sept 2...	Version control conflicts in Git	Medium	High	Medium	Lead Developer (Jijendran)	Occurred – Solved	Review git commits, pull merge requests and ensure everyone follows the pre-determined git policy	Enforce Git workflow guidelines and frequent commits	Resolve conflicts collaboratively and use code review sessions
R8	N/A	Team member workload conflicts	Medium	High	Medium	Scrum Master (Esha)	Not occurred (monitored)	Track task completion and hold each other accountable through the weekly scrum meetings	Distribute workload evenly using Jira assignments	Reassign tasks as needed and pair inexperienced members with experienced ones

Risk ID	Date raised	Description	Estimated Impact	Likelihood	Severity	Owner	Status	Monitoring Strategy	Mitigation Plan	Contingency Plan
R9	20 Sept 2...	Technical skill gaps within the team	Medium	Medium	Medium	Scrum Master (Esha)	Open	Monitor the progress of the tasks, identify which team member is struggled	Schedule training sessions for team members to learn the new technologies	Reassign critical tasks to experienced members and extend deadlines for beginner members if required
R10	N/A	Drop in team morale causing low productivity	Medium	Medium	Medium	Team Leader (Esha)	Not occurred (monitored)	Hold regular team checkins and anonymous rating system to gauge team spirit	Ensure healthy work-life balance, celebrating small wins and ensuring clear communications	Organize team-building sessions and address issues in an open discussion
R11	N/A	Additional features added last minute	Medium	Medium	Medium	Product Owner (Jijendran)	Not occurred (monitored)	Make sure there is backlog grooming regularly in every weekly meeting	Track if there is any new features request and use Jira to monitor backlogs	Reallocate prioritizations and prioritize the critical features.
R12	N/A	Hosting or server downtime	Medium	Low	Low	Scrum Master (Esha)	Not occurred (monitored)	Need monitor the normal operation time of server	Choose reliable hosting providers and hold the backup servers	Switch to backup server until the service is restored

Appendix

Figure 2.1 Risk Level Matrix

		Impact		
		Low	Medium	High
Probability	High	Low	Medium	High
	Medium	Low	Medium	Medium
	Low	Low	Low	Low

Reference

Atlassian. (2025, March 28). Trello vs. Jira: which to choose (and how to use them together) -

Work Life by Atlassian. Work Life by Atlassian.

<https://www.atlassian.com/blog/project-management/trello-vs-jira>

Chiaramonte, M. (2024, June 21). *Application Scalability: Ensuring Performance and Reliability*.

VFunction. <https://vfunction.com/blog/application-scalability/>

Kareliya, A. (2025, August 21). *NodeJS vs Python - Which Language is Best for Backend Web*

Development? Radixweb. <https://radixweb.com/blog/nodejs-vs-python>

Kinza Yasar. (2023). *What is web application (web apps) and its benefits*. TechTarget .

<https://www.techtarget.com/searchsoftwarequality/definition/Web-application-Web-app>

Mobile App vs Web App: What's the difference? (n.d.). BrowserStack.

<https://www.browserstack.com/guide/mobile-app-vs-web-app>

Nicolaou, K. (2023, October 9). *Web vs. Mobile Applications: Which One Offers Better User*

Experience? (2023). Brain Box Labs.

<https://brainboxlabs.com/blog/web-vs-mobile-applications-which-one-offers-better-user-experience>

Terrell Hanna, K., & Wigmore, I. (2023, February). *What is mobile app?*. TechTarget .

<https://www.techtarget.com/whatis/definition/mobile-app>

Rafalski, K. (2024). *Supabase vs. Firebase: Which BaaS is Best for Your App?* Netguru.com; Netguru.

<https://www.netguru.com/blog/supabase-vs-firebase>

Rohn, S. (2022, July 28). *What is a desktop application? +challenges, use cases*. Whatfix.

<https://whatfix.com/blog/desktop-application/>

Web app vs mobile app: key features and differences. (2024). Indeed.

<https://uk.indeed.com/career-advice/career-development/web-app-vs-mobile-app>