ASSIGNMENT 1

PROJECT INCEPTION

Unit Code: FIT2101 Software Engineering Process and Management (SEPAM)

Date: 26th August 2025

Team Members:

No	Student Name	Student ID
1	Chen Zixuan	34474293
2	Cheong Wai Yan	35742437
3	Jason Tan Jee Kin	34040544
4	Meow Kai Bin	34296646
5	Low Qing Yu	35599081
6	Lydia Ong Jin Xuan	34089896

TABLE OF CONTENTS

1.0 Deliverable - The Project Plan

1.1 Project Vision

For teaching centres and training institutes that struggle with managing courses, tracking student progress, and keeping learners engaged, Beacon is an LMS that provides a streamlined, easy-to-use platform tailored to the needs of smaller educational organisations. Unlike many traditional learning management systems, our product avoids unnecessary complexity, offering only the essential tools in a clean and intuitive interface that does not require technical expertise to manage.

1.2 Team Profile

Team name: DevNest

We are DevNest, a team of developers dedicated to building innovative software through collaboration and creativity. As developers, we share a passion for crafting solutions that bring value to users and push our skills forward. Our team operates like a nest, a safe and collaborative space where ideas are nurtured, trust is built, and creativity flourishes. Each member brings unique skills and perspectives, and together we foster an environment where innovation can take flight. The name “DevNest” captures this spirit of teamwork, growth, and building something greater together.

1.3 Client Information

Client Name: Ms. Kamalahshunee K Velautham
Client Email: Kamalahshunee.KVelautham@monash.edu
Client Affiliation: Lecturer with over 30 years of experience

Ms Kamalahshunee K. Velautham is the primary client for this project and plays a pivotal role in guiding the development process of this project. With extensive experience in adult education and curriculum design, her team provides invaluable domain expertise, defines project requirements, and offers continuous feedback to ensure alignment with their organisation's educational goals.

Her organisation, operating from a single campus, which is located in EcoCity, Kuala Lumpur, Malaysia, is dedicated to providing learning opportunities for adults aged 18 and above who may not have had the opportunity to attend university. The institution aims to equip learners with practical skills that enhance their career prospects and support personal growth. Currently, the organisation faces operational challenges due to the manual management of resources and the storage of information across multiple platforms, which has proved to be inefficient and time-consuming. These limitations affect course administration, tracking of learner progress, and overall engagement. Recognising these challenges, Ms Kamalahshunee engaged with our team to design and implement a cost-effective Learning Management System (LMS) that addresses these inefficiencies while remaining accessible and user-friendly.

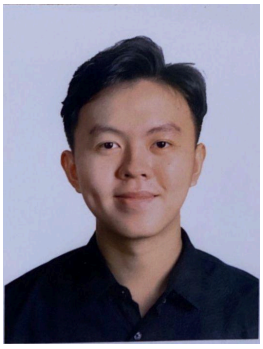
The client actively participates in the project through weekly meetings, providing guidance on requirements, priorities, and user experience considerations. Her involvement ensures that the LMS is aligned with the organisation's vision and meets the practical needs of both instructors and learners.

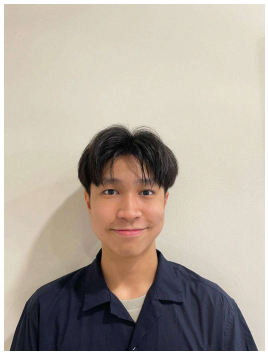
Through this collaboration, the LMS aims to:

- Simplify course and lesson management for instructors
- Centralise student data and learning progress
- Enhance learner engagement and flexibility
- Offer practical suggestions for decision-making
- Support adaptability and long-term growth

Ms Kamalahshunee's leadership and expertise make this project an invaluable opportunity for our team to contribute to meaningful educational outcomes and deliver a solution that will have a lasting impact on adult learners.

1.4 Roles & Responsibilities

Name	Contact	Roles
Meow Kai Bin 	kmeo0001@student.monash.edu	Quality Assurance (QA) Developer: Back End
Low Qing Yu 	qlow0006@student.monash.edu	Scrum Master (SM) Developer: Front End
Cheong Wai Yan 	wche0236@student.monash.edu	UI/UX Designer Developer: Full Stack

Jason Tan Jee Kin 	jtan0417@student.monash.edu	Tech Lead Developer: Back End
Chen Zixuan 	zche0305@student.monash.edu	Tech Lead Developer: Front End
Lydia Ong Jin Xuan 	long0020@student.monash.edu	Product Owner (PO) Developer: Back End

In this project, our team is structured around Agile roles to ensure smooth collaboration and clear accountability, as each member is responsible for the tasks of their specific role. The Agile roles are based on the Scrum framework, consisting of the Product Owner, Scrum Master, and the Development Team. These roles define how we manage requirements, facilitate processes, and deliver value incrementally. In addition to these, we have defined ourselves into each development team's roles that reflect the specific skills and responsibilities of individual members, such as Tech Leader, UI/UX Designer, Front-End Developer, Back-End Developer, Full-Stack Developer, and Quality Assurance. While each member has their own primary focus, all members collaborate closely to achieve the project goals, following Agile principles of teamwork and adaptability.

1.4.1 Agile Roles

Product Owner *Lydia Ong Jin Xuan*

As the Product Owner, Lydia represents the client's perspective and ensures that the product vision is met. She defines and prioritises the product backlog, clearly outlining requirements for the team to implement. Throughout the project, she communicates with team members to clarify expectations and address uncertainties. During sprint reviews, she evaluates the completed work and determines whether it meets client needs, making her the key decision-maker in validating the product.

Scrum Master *Low Qing Yu*

Qing Yu serves as the Scrum Master, facilitating the Agile process within the team. Her responsibilities include organising sprint planning, reviews, retrospectives, and daily stand-ups. She ensures that the team follows Scrum principles and works efficiently within sprints. Beyond managing the process, she also helps remove obstacles that may slow down progress, allowing the team to stay focused on delivering results.

Development Team *(More details below)*

The Development Team is cross-functional and responsible for designing, building, testing, and delivering increments of the product. This involves all members of the development team collectively managing technical implementation, quality assurance, and integration of all components while collaborating closely with the Product Owner and Scrum Master.

1.4.2 Our Development Team

Tech Leader

Jason Tan Jee Kin & Chen Zixuan

Jason and Zixuan serve as the Tech Leaders, who mainly focus on guiding the technical direction of the project. They are responsible for selecting the appropriate technology stack, setting coding standards, and ensuring that the architecture is scalable and maintainable. They also support other developers by reviewing code, solving technical issues, and providing expertise in complex areas of development. Their leadership ensures the technical quality and reliability of the final product.

UI/UX Designer

Cheong Wai Yan

Wai Yan takes on the role of UI/UX Designer, focusing on the usability and design of the system. She creates wireframes, prototypes, and mockups to visualise how the product should look and feel. Her work ensures the interface is user-friendly, accessible, and aligned with the client's needs by collaborating with the Product Owner. She also collaborates closely with developers to ensure that her designs are practical and can be effectively implemented in the final system.

Front-End Developers

Low Qing Yu & Chen Zixuan

Qing Yu and Zixuan serve as Front-End Developers, focusing on the client-facing side of the system. Their responsibilities include implementing user interfaces based on the designs provided by the UI/UX Designer and ensuring that the product is visually appealing, responsive, and intuitive for users. They also work on integrating the front-end with back-end services, making sure that interactions such as role selection, navigation, and real-time updates function seamlessly.

Back-End Developers

Meow Kai Bin, Jason Tan Jee Kin & Lydia Ong Jin Xuan

As Back-End Developers, Kai Bin, Jason, and Lydia are responsible for building and maintaining the server-side logic of the system. Their work focuses on database management, server configuration, and the creation of secure and efficient APIs that allow the front-end to communicate with the back-end seamlessly. They ensure data is stored, processed, and retrieved reliably while also addressing performance and scalability. Their contributions form the backbone of the application, enabling smooth system functionality.

Full-Stack Developer *Cheong Wai Yan*

Wai Yan takes on the role of Full-Stack Developer, contributing to both the front-end and back-end of the project. Her versatility allows her to bridge gaps between the two sides. This ensures that the design vision translates smoothly into functional implementation. She may assist with database design, API development, and front-end coding, while also aligning technical solutions with usability considerations. By covering both domains, she provides flexibility and balance in the team's overall development process.

Quality Assurance (QA) *Meow Kai Bin*

Kai Bin is responsible for Quality Assurance. He aims to ensure the final product meets the required standards of performance, reliability, and usability. His role involves designing and executing test cases, identifying bugs or issues, and working with the team to ensure problems are resolved before deployment. He documents defects, verifies fixes, and ensures that each feature functions as intended. By maintaining quality throughout the development cycle, he helps deliver a stable, user-friendly, and dependable system.

1.5 Process Model

For this project, our team has collectively agreed to adopt the Agile methodology, specifically the Scrum framework, as the guiding process model for managing and executing the Software Development Life Cycle (SDLC). This approach was selected because it promotes adaptability to changing requirements and enhances collaboration among stakeholders by providing and receiving feedback at multiple stages rather than only at the end of the project. By integrating Scrum practices into the SDLC, the team can remain responsive to evolving business needs, enhance collaboration across roles, and maintain a steady focus on delivering value. Ultimately, this approach supports the creation of a product that not only meets technical specifications but also aligns with user expectations and organisational goals.

1.5.1 Scrum

Scrum is an Agile framework designed to manage complex software development projects through an iterative and collaborative approach. By leveraging Scrum, our team structures development into a flexible approach that divides the work into short, time-boxed iterations, known as sprints. This enables continuous feedback, regular reassessment of priorities, and incremental delivery of functional components.

1.5.2 Scrum Artifacts

Product Backlog

The product backlog is a key artifact that captures and expresses the customer's requirements as user stories. User stories provide concise, user-focused descriptions of desired features, ensuring that the development process remains aligned with stakeholder needs. The product backlog is owned and continuously refined by the product owner, who prioritises items based on business value and project goals. It serves as the primary source of work for the development team, guiding sprint planning and ensuring that the most valuable functionality is delivered first.

Sprint Backlog

The sprint backlog is established in sprint planning, when the team commits to specific high-priority items from the product backlog for delivery within the sprint timeframe. It provides a focused set of tasks that guides the team's work throughout the sprint, ensuring clarity of scope and preventing unnecessary tasks. By clearly defining what will be delivered, the sprint backlog promotes accountability, transparency, and alignment towards achieving the sprint goal.

Sprint

In Scrum, a sprint is a short, set timeframe, normally one to four weeks, where the team works on the items selected from the sprint backlog. For this project, the sprint duration has been set to two weeks to better align with the university schedule. Throughout the sprint, the team focuses on completing the planned tasks while holding daily Scrum meetings to track progress and address any challenges that arise. The sprint acts as the core container for all other Scrum events and drives the iterative and incremental development process.

1.5.3 Scrum Events

Sprint Planning

Sprint Planning is the event that initiates each sprint, enabling the team to decide on the deliverables and the approach to completing them. During this meeting, items from the product backlog are moved into the sprint backlog. The development team then breaks down these items into smaller, actionable tasks, estimates the required effort, and considers team capacity and skills. The outcome is a shared sprint goal and a clear plan of work, ensuring that everyone is aligned and committed to delivering value by the end of the sprint.

Daily Scrum (Stand-Ups)

The daily scrum is a short, structured meeting held daily to ensure the team stays synchronised. Each member provides an update on their progress, plans for the next 24 hours, and highlights any impediments. This key event ensures transparency, fosters collaboration, and allows the team to adapt quickly to emerging challenges. By maintaining regular communication, the daily scrum helps the team stay aligned and focused on achieving the sprint goal.

Sprint Review

The sprint review is carried out once a sprint finishes and serves as a presentation of the completed work to stakeholders. The team showcases the minimum viable product (MVP), gathers feedback, and discusses progress toward the overall product goal. This collaborative session helps refine the product backlog, adapt priorities, and ensure that future work continues to meet stakeholder needs and expectations.

Sprint Retrospective

Following the sprint review, the sprint retrospective is held to evaluate the Scrum team's internal processes and collaboration. Unlike the review, which involves external stakeholders, this meeting is dedicated to reflecting on what went well, what could be improved, and how the team can enhance their workflow in future sprints. It is a key event for fostering continuous improvement, strengthening teamwork, and ensuring that the team becomes more effective over time. This is a good time to adapt the way the team works together, not just the product.

1.6 Definition of Done (DoD)

The Definition of Done (DoD) is the team's formal agreement on what it means for a backlog item to be considered complete and ready for release. It exists to prevent misunderstandings, ensure consistent quality, and avoid the risk of "undone work" being carried forward. In Scrum, the goal of each sprint is to deliver a *Minimum Viable Product (MVP),* work that is not only functional but also tested, reviewed, and ready for client feedback. For our team, the DoD provides the standard checklist against which all features and tasks are measured, ensuring that every increment meets the same high level of quality.

A backlog item is not "done" just because it has been written or compiled without error. Code is only considered complete once it has been reviewed, tested, and accepted by both the team and the Product Owner. To make this clear, our DoD includes several layers of requirements. At the development level, all code must adhere to team coding conventions, be pushed through GitLab using merge requests, and pass peer review before being merged into the shared dev branch. Peer review ensures that no feature is integrated without at least one other team member confirming its quality, readability, and maintainability.

Testing and quality assurance are equally central to our DoD. Every new feature must include unit tests, with a target of at least 80% coverage for new functionality, as well as integration testing to ensure that components work together correctly. Manual QA testing will also be conducted to validate that the feature meets its acceptance criteria and behaves as expected in real scenarios. Regression testing will be performed to confirm that existing features are not broken by new changes. Only when all these tests pass can the feature progress toward completion.

Documentation is another requirement for "done". Developers are expected to update README files, add meaningful inline comments where needed, and ensure that Jira tickets are updated with relevant test results, decisions, or review notes. This ensures that knowledge is not lost and that other team members can understand and maintain the feature later.

Finally, deployment readiness forms the last stage of our DoD. A feature must be successfully merged into the dev branch without conflicts, verified in the staging environment, and formally reviewed by the Product Owner. Only after this acceptance will the feature be considered complete and eligible for inclusion in the main branch at the end of the sprint.

In summary, our Definition of Done ensures that a feature is not simply “coded”, but thoroughly tested, reviewed, documented, and ready to ship. By holding every backlog item to this standard, we guarantee consistent quality, reduce technical debt, and maintain transparency in our development process. Over time, the team will continue to refine and strengthen the DoD as our capabilities mature, moving closer to the ideal of delivering fully shippable increments at the end of each sprint.

Area	Completion Criteria
Code Quality	Code passes linting, adheres to team coding standards, and undergoes peer review via GitLab merge requests prior to merging.
Functionality	The functionality properly implements all acceptance criteria and satisfies client needs.
Testing	All automated tests have passed, including unit tests with ≥80% coverage, integration tests, regression tests, and manual QA.
Integration	The feature merges without issue, goes through CI/CD pipelines, and functions well with existing modules in staging.
Documentation	Inline comments, README files, and Jira tickets are all updated with important remarks, test results, and decisions.
UI/UX	The feature complies with agreed-upon design criteria and passes usability assessment without major concerns.
Sign-off	During the Sprint Review, the Product Owner reviews and accepts the feature.

1.7 Team Schedule & Availability

At the start of the project, the team established a structured weekly schedule to balance collaborative work, client engagement, and individual commitments. This schedule ensures that the team remains consistent in communication, aligned in progress, and responsive to both client and tutor feedback.

- **Wednesday Meetings (2:00 PM – 4:00 PM, on campus):**
  Dedicated to internal discussions, these meetings act as a checkpoint to ensure the project is progressing as planned. The primary focus is on preparing for upcoming client sessions, where the team discusses strategies for presenting updates, clarifying requirements, and addressing client feedback. In addition, progress tracking is carried out to ensure that each member is meeting their commitments and contributing consistently towards the sprint goals. The team also uses this time to review new project components, such as UI designs or updated documentation. Other than technical discussions, the Wednesday meetings allow the team to resolve challenges, brainstorm solutions, and strengthen team alignment. This dedicated slot ensures that potential issues are identified early, responsibilities are clearly distributed, and the team remains synchronised before engaging with the client later in the week.

- **Thursday Applied Classes (10:00 AM – 1:00 PM, on campus):**
  Thursday meetings take place during scheduled applied classes and act as both a regular team collaboration session and a formal client meeting. Unlike Wednesday's internal focus, Thursday's structure is more formal, with team members expected to present progress clearly and professionally. This meeting also provides a valuable opportunity to receive direct feedback from both the client and the tutor. In addition to presentations and discussions, Thursday sessions often serve as collaborative working time, where the team refines deliverables, integrates feedback, and resolves issues raised earlier in the week. By combining internal collaboration with external validation, Thursday applied classes reinforce accountability and ensure that the development process remains transparent and stakeholder-focused.

- **Stand-up Meetings (Online, every three days):**
  Once the first sprint begins, the team will conduct short online stand-up meetings, limited to 15 minutes, to provide progress updates, discuss impediments, and coordinate upcoming tasks. The stand-ups are intentionally kept short to minimise disruption while ensuring regular communication. During each session, members are expected to also provide an outline of their goals for the next 24 hours. Although brief, these meetings are critical for maintaining transparency, fostering accountability, and ensuring that the team adapts quickly to emerging challenges. The online format was chosen to balance efficiency with convenience, acknowledging that each team member has additional academic and personal responsibilities outside of this project.

- **Sprint Planning Meetings:**
  To maximise efficiency, sprint planning will be divided across the Wednesday and Thursday meetings. On Wednesdays, the team will draft the sprint backlog, and the draft plan will be presented and finalised on Thursday with input from the tutor, ensuring alignment with project goals and realistic capacity planning. This structured approach ensures that each sprint begins with a clear, shared vision and a practical plan of execution.

- **Sprint Review and Retrospective (End of each sprint):**
  After each sprint, the Scrum Master will facilitate two important events, which are the sprint review and the sprint retrospective. The sprint review involves presenting the completed increment to the client and other stakeholders, demonstrating tangible progress, and gathering feedback. By doing so, the client remains closely involved in shaping the product, and the team's work continues to meet evolving requirements. After which, the sprint retrospective lets the team reflect on processes, overall effectiveness and identify improvements. While these events do not have fixed weekly times, they are scheduled at the end of each sprint and play a vital role in maintaining both accountability to stakeholders and continuous team improvement.

1.8 Task Allocation Strategy

In line with the Scrum framework, task allocation within our team will take place during sprint planning meetings, where the team collectively decides how to break down backlog items into actionable tasks. To ensure fairness, efficiency, and accountability, team members will volunteer for tasks based on their strengths, skill sets, and availability. This approach ensures that the workload is manageable while also providing opportunities for members to broaden their experience by taking on tasks outside of their usual comfort zones, thereby encouraging growth and cross-functional collaboration.

During these meetings, the Scrum Master's role is to ensure that the allocation process remains smooth, transparent, and balanced. Their responsibility is to facilitate an environment where no member is overworked and everyone is on board with the allocation. In some cases, when members are unsure of whether they can take on a particular task or when the discussion reaches a standstill, the developer team's designated Tech Lead may provide guidance. The Tech Lead can suggest suitable matches between tasks and team members by considering technical strengths, prior experience, and the complexity of the work involved. However, in keeping with the principles of Scrum, the final decision always rests with the individual, preserving the element of autonomy and ownership over assigned responsibilities.

Since the first sprint has yet to start, the task allocation for this project inception has been organised in a more structured and predefined way. This initial allocation helps establish a clear baseline, ensuring that all team members understand their roles and responsibilities before the iterative sprint cycles begin. The designated tasks for each team member for the project inception are as follows:

- User Interface (UI) Prototype: All members
- Project Inception (Deliverable 1): Low Qing Yu and Zi Xuan
- Project Inception (Deliverable 2): Jason and Meow Kai Bin
- Project Inception (Deliverable 3): Cheong Wai Yan and Lydia Ong Jin Xuan

1.9 Progress Tracking

As we all know, ensuring that every member stays aligned and connected is essential in Scrum. A lack of coordination can easily lead to inefficiencies, misunderstandings, or duplicated work, all of which slow down productivity. To avoid this, the team has carefully selected a set of tools that will help us collaborate smoothly and stay on the same page throughout the project. The main tools we will be using are Google Drive, Google Meet, GitLab, WhatsApp, and, most importantly, Jira. Each of these plays a unique role in supporting different aspects of our workflow.

Google Drive will be the team's primary platform for documentation and file management. All materials that are related to the project, like the prototype versions of the UI or the project inception document, will be systematically stored in designated folders on Drive. One of the biggest reasons for choosing Google Drive is its accessibility. As long as a member has internet access, files can be easily viewed, shared, or updated from any device, ensuring that no one is left out of the loop.

When it comes to online meetings, the team has decided to use Google Meet as the main platform for virtual stand-ups. Since our daily scrums will be held every three days, it was important to choose a tool that is quick, reliable, and does not require additional setup. Google Meet runs directly in the browser, making it easy for everyone to join meetings without the need to download or install separate software. Another advantage is its integration with Google Drive, which allows meeting links and notes to be automatically shared with the team. This level of integration minimises miscommunication and ensures that important meeting information is never missed.

For coding and version control, GitLab has been chosen as our repository. This tool is crucial for the developer team since it allows multiple people to work on the same project simultaneously without clashing with each other's work. Each member can develop features or fix bugs on their own branch and merge them into the main branch. GitLab's built-in version history also acts as a safety net. If we ever need to revisit older code, the system keeps a clear record of past changes. Beyond version control, GitLab also promotes good collaboration practices by making code reviews and merge requests part of the workflow.

On a more casual note, WhatsApp will serve as the team's primary communication platform outside of formal meetings. While project-related updates will certainly be shared there, such as reminders about meetings or minor progress updates, the group chat will also act as a more relaxed space where members can bond and support each other. The team felt that while meetings tend to be more formal and structured, it's just as important to create a channel for casual interactions. Building this sense of trust is key to fostering a collaborative team spirit and making sure everyone feels comfortable asking questions or reaching out for help when needed.

Finally, the most critical tool for our Scrum process is Jira. Jira will serve as our task and sprint management system, allowing us to fully embrace Agile principles. The platform comes with a Scrum template that we can customise to fit the unique needs of our project. One of the most powerful aspects of Jira is its ability to provide a clear overview of the project at any given moment. This transparency helps the team stay accountable. Moreover, Jira allows us to plan sprints more effectively by ensuring that tasks are distributed evenly and progress is measurable.

In essence, each of these tools has been carefully chosen to serve a specific purpose within our workflow. Google Drive centralises our documentation, Google Meet facilitates online discussions, GitLab manages code collaboration, WhatsApp keeps us connected informally, and Jira ensures our Scrum process runs smoothly. When we utilise all of them together, they form the backbone of our communication, documentation, and development practices, ensuring that the team is equipped to work productively and efficiently from start to finish.

1.10 Backlog Management

Product Backlog

Product backlog is one of the most critical aspects in a Scrum framework. It serves as a single, centralised source that captures all of the client's requirements, priorities, and expectations. In short, the product backlog acts as the blueprint for development. Without it, the team would waste significant time and effort figuring out what to build next, potentially leading to confusion, misalignment, or duplicated work.

As mentioned before, ownership of the product backlog lies with the Product Owner, who carries the responsibility of maintaining, refining, and prioritising it throughout the project. After each client meeting, where requirements and feedback are collected, the Product Owner translates these inputs into user stories. While the Product Owner oversees this task, the responsibility of writing user stories does not have to fall entirely on their shoulders. Other team members can also draft user stories based on certain requirements, but the Product Owner ensures they meet quality standards before being officially included in the backlog.

Nonetheless, the team must follow a clear and user-centred format when writing user stories. That being said, when creating a user story, the writer should prioritise writing the requirements from a user's perspective rather than from a purely technical or developer-orientated lens. It emphasises why a feature is needed and the value it brings, not just what needs to be implemented. It should follow the format of:

As a (target user/audience),
I want to (purpose/goal) so that I can (benefit/value).

This ensures that we are clear on what needs to be done to satisfy the user story. Beyond writing, each user story must also be given story points. It is a way to estimate the complexity, effort, or time required to complete the work. Even so, story points are not decided by one's judgement. The assignment of story points is a collaborative activity involving the entire team, ensuring that estimates reflect a balanced view of effort based on the skills and capacity of the group. This prevents bias, builds shared understanding, and fosters ownership across the team.

Over time, the Product Backlog will continuously evolve. If user stories are unclear, incomplete, or if new client requirements are brought out, the Product Owner revisits and refines the backlog. Refinement is always done with the client's best interests in mind, making sure the backlog reflects what will provide the most value to them rather than what the developers find easiest to implement. Finally, all user stories are organised and stored in Jira, which serves as the team's central tool for scrum process management. Jira allows the team to view, update, and track stories efficiently, keeping the backlog transparent and accessible to everyone.

Sprint Backlog

While the product backlog represents the entire scope of work for the project, the sprint backlog narrows the focus down to a specific set of tasks that the team commits to delivering within the current sprint. The sprint backlog is created and updated during sprint planning, which is a joint session involving the Scrum Master, Product Owner, and Development Team.

During these meetings, the Scrum Master is responsible for facilitating the meeting, preventing any overcommitment from any of the members. The Product Owner presents the most important work items from the product backlog, explains their value, and helps the team craft the sprint goal. At the same time, the Developer Team works with the Product Owner to understand the product backlog items and plan how to complete them. By the end of the planning session, the team collectively agrees on a sprint goal and a set of user stories to move into the sprint backlog. While the Product Owner contributes heavily to setting priorities, the sprint backlog is ultimately owned by the Development Team, since they are the ones responsible for delivering the work.

Likewise, the sprint backlog is also managed using Jira. This provides a transparent, real-time view of what tasks are in progress, what has been completed, and what remains to be done. Having it stored in Jira also makes it easy for all team members to stay aligned, track progress during daily scrums, and adapt quickly if unexpected challenges arise.

1.11 Time Tracking

To keep track of the time spent on project tasks, our team will use Clockify integrated within Jira so that time tracking happens directly in the same environment where we manage our Scrum process. This integration allows team members to log time, ensuring that each user story, task, or bug has an accurate record of the effort required to complete it. Whenever a member begins working on a task, they can simply start the Clockify timer from within Jira, which automatically links the time entry to that specific backlog item. Once they finish, they stop the timer, and the time is logged under that issue without needing to switch between applications. For cases where the timer is not used, team members can also manually add their working hours directly in Jira, ensuring flexibility and accuracy in reporting.

By keeping all tracking tied to Jira, we create a direct connection between backlog items and time spent, making it easier to evaluate workload distribution, identify bottlenecks, and refine our sprint planning process. At the end of each week, team members will review their logged hours, while the Scrum Master will monitor the data through Jira's integrated Clockify reports, which can be exported to PDF or CSV if needed. These reports allow the team to analyse effort across tasks, user stories, or sprints and provide visibility for both internal reflection during sprint retrospectives and for demonstrating accountability to stakeholders. Overall, this approach ensures that our time-tracking is transparent, accurate, and aligned with our Scrum workflow in Jira.

1.12 Git Policy

To maintain collaboration, stability, and accountability during development, our team will follow a structured Git workflow managed in GitLab. The repository will adopt a three-tier branching model. The *main* branch will always contain stable, production-ready code and will be strictly protected against direct commits. The *dev* branch will act as the integration branch, where all approved work is merged during the sprint. All new development will be carried out in feature branches, created from *dev* and named according to their purpose, such as feature/student-signup for new functionality or fix/login-bug for bug fixes. Each team member will maintain their own feature branches linked to the user stories assigned to them in Jira, ensuring clear ownership and traceability of work.

All contributions must be integrated through merge requests (MRs). No feature branch may be merged directly into *dev* or *main* without going through this process. An MR must include a descriptive title, a summary of the changes made, references to the relevant Jira issue or user story, and confirmation that all local tests have passed. Peer review is mandatory: at least one team member must review and approve the MR before merging. In case of disagreements or quality concerns, the Tech Lead or QA will have the authority to provide final approval. This process ensures code quality, encourages collaboration, and prevents errors from entering shared branches.

Commit messages will follow the Conventional Commits format to provide consistency and clarity in project history. The main prefixes used will be:

- feat: for new features (e.g., feat(login): add Google login option)
- fix: for bug fixes (e.g., fix(UI): correct dashboard layout)
- docs: for documentation updates
- test: for new or updated tests
- refactor: for code restructuring without changing functionality

Each commit should represent a small, logical unit of work rather than bundling multiple unrelated changes. This practice makes the history easier to read, simplifies code reviews, and allows changelogs to be generated automatically.

To prevent merge conflicts, developers are expected to pull the latest changes from *dev* regularly, especially before beginning new work or submitting a merge request. Feature branches should remain short-lived and be merged back into *dev* frequently to avoid divergence. If a conflict occurs, the developer responsible for the branch will first attempt to resolve it locally. For complex cases, they will collaborate with other team members; if the conflict cannot be resolved, the issue will be escalated to the Scrum Master to coordinate a solution. Once resolved, the branch must be retested to ensure stability before merging.

Several safeguards will be applied to enforce this workflow. GitLab's branch protection rules will be enabled to prevent direct commits to *main* and *dev*, ensuring all contributions pass through the MR process. Each team member will use their own GitLab account, making contributions traceable and transparent. Automated continuous integration and continuous deployment (CI/CD) pipelines will run on every MR to perform builds, linting, and tests, blocking merges if checks fail. Deployment will follow this structure: code in *dev* will be deployed to a staging environment for testing during the sprint, while only code in *main* will be deployed to production at the end of each sprint.

In summary, our Git policy establishes clear rules for branch structure, merge request handling, commit message conventions, and conflict resolution. By combining disciplined practices with automated safeguards, the team ensures that the codebase remains stable, transparent, and traceable throughout development. This structured approach reduces risks, supports collaboration, and enables the delivery of high-quality increments that align with Scrum principles.

2.0 Deliverable - Analysis of Alternatives

2.1 Terms of Reference

One of the most important factors in this project is the client's requirements. The client has requested that the system provide both card and list views for displaying information, ensuring flexibility in how data can be accessed. In addition, the client did mention that the overall user interface must reflect the theme of Taylor Swift's new album. This requires the design to be interactive, visually appealing, and professional, creating a balance between creativity and usability. The client also requires the system to be accessible on both desktop and mobile devices for the students and instructors to access the course material or their desired needs at the nick of time. Meaning that the solution must be fully responsive and adaptable across platforms.

Another factor taken into account is the time frame for development. Since the project must be completed within a limited period, the technologies selected must enable rapid development and deployment. Choosing frameworks and tools that reduce overhead while offering built-in functionality is essential to meeting deadlines without sacrificing quality.

The knowledge and expertise of the development team also played a key role in determining the technologies to be used. By selecting languages and frameworks that the developers are already familiar with, the project can be initiated quickly, reducing the learning curve and ensuring smooth progress. This decision also improves efficiency and helps mitigate risks associated with unfamiliar technologies.

Several options were considered during the planning stage. For the application platform, web application, mobile application and desktop application were evaluated against several criteria, including cross-platform accessibility, development time and effort, team familiarity and skills, cost, performance and responsiveness, ease of maintenance and updates, scalability, UI/UX flexibility and deployment and distribution. From these, key criteria were priorities to align with the client's requirements, including accessibility across both mobile and desktop devices, overall performance and responsiveness, and the ability to deliver a polished and flexible user interface. Given the project's tight time frame, emphasis was also placed on selecting a platform that supports rapid development and leverages the team's existing expertise.

For backend programming languages, Java, Python, and PHP were examined. For frontend programming languages, HTML, CSS, JavaScript, and PHP were evaluated as well. On the framework side, Django, Tailwind CSS, HTMX, and React were examined. Each option was assessed in terms of suitability, scalability, ease of use, and ability to meet both the client's requirements and the project's time constraints.

Database selection was another crucial factor to a successful LMS that stores personal information for both instructors and students. For database query language, both SQL and NoSQL options were examined to decide which would the team choose to structure the data. In addition to query language, SQL dialects such as MySQL, PostgreSQL, SQLite and MariaDB were evaluated in criteria such as open source, ACID compliance, standards compliance, advanced features, performance, scalability, best use cases and difficulty scale. As security and data privacy were also significant considerations throughout the evaluation process, it is important to ensure the selected database must adhere to industry-standard security protocols. Performance, advanced features, and scalability are reviewed to address the growth of the platform in the future. We also evaluated the difficulty scale and listed some best use cases to help align the choice with the team's expertise and project requirements.

For hosting services, various alternatives were being explored, such as Heroku, Supabase, Render, and Railway, based on criteria such as types of hosting, suitability for the project, ease of use, pricing, deployment complexity, and built-in features. The platforms were carefully evaluated to determine which could provide a balance of flexibility, scalability, and cost-effectiveness while minimising setup and maintenance overhead. Given the project's time constraints, preference was placed on platforms that offered straightforward deployment pipelines, managed infrastructure and integration with version control systems for continuous delivery.

Taking these factors into account, the selection process involved not only comparing technical specifications, but also aligning each choice with the client's expectations, user experience goals, and the development team's capabilities. This comprehensive evaluation of application platforms, programming languages, frameworks, databases, and hosting services ensures that the chosen technology stack will provide a strong foundation for a responsive, secure, and visually engaging LMS that meets both functional requirements and project deadlines.

2.2 Body

2.2.1 Application Platform

Criteria	Web Application	Mobile Application	Desktop Application
Cross-Platform Accessibility	Inherently cross-platform, the application works on any device with a browser	Often requires separate development for iOS and Android unless using cross-platform frameworks (Flutter, React Native)	Platform-specific; requires separate builds for Windows, macOS, Linux
Development Time & Effort	Faster development since one app works across devices	Higher effort due to multiple platforms, app store compliance	Can be complex due to different OS environments; more testing required
Team Familiarity & Skills	Moderate	Moderate	Low
Cost	Lower cost overall; single codebase	Higher cost to build and maintain multiple versions; app store fees	Higher cost due to distribution challenges and support for multiple OS
Performance & Responsiveness	Slightly lower performance as limited by browser and internet	Better performance, quicker loading times	High performance; direct system access
Ease of Maintenance & Updates	Seamless updates via server; no user action needed	Updates require store approval; rollout delays	Users must manually install updates; version control is harder
Scalability	Highly scalable; server scaling handles user load easily	Scales well if built properly; app store distribution supports large audiences	Scaling is harder; must be installed individually on devices
UI/UX Flexibility	Limited by browser capabilities but improving (PWA)	Rich, device-integrated UI/UX (gestures, notifications)	Full control over UI; great for specialised software
Deployment & Distribution	Easy distribution via URLs; no store restrictions	Requires App Store/Google Store Play approval; good discoverability	Manual distribution or enterprise tools needed; less discoverable

2.2.2 Frontend Framework

Criteria	HTMX	Bootstrap	Tailwind CSS	React
Language	HTML	CSS, JavaScript	CSS	JavaScript / TypeScript
Complexity	Very lightweight, minimal learning curve	Easy to start, higher complexity for customizations	Moderate, requires understanding utility-first CSS	High complexity, requires component-based thinking and a build setup
Built-in Features	AJAX, WebSockets, SSE, easy HTML-based interactivity	Prebuilt responsive grid, components (buttons, navbars, modals)	Utility classes for styling, responsive design, and customization via config	Virtual DOM, state management, component lifecycle, JSX
Development Progress	Fast prototyping, small learning effort	Fast for prototypes, slower for deep customization	Fast styling once familiar with the class system	Slow start, but scalable for large apps
Community Support	Growing community, but niche	Very large, long-standing community	Large and growing, strong documentation	Massive community, the largest ecosystem of frontend libraries
Usage	Add interactivity without writing JavaScript. Recommended for simple apps	Building responsive sites quickly with ready components	Styling modern UIs quickly, great for design consistency	Building complex, interactive, large-scale single-page apps
Hosting Services	No hosting needed	No hosting needed	No hosting needed	Needs bundling / deployment
Hosting Cost	None	None	None	Depending on the hosting platform
Difficulty Scale (1-5)	2	2	3	4

2.2.3 Backend Language

Language	Python	Java	PHP
Feature	High-level, easy-to-read, dynamically typed	Strongly typed, object-oriented, platform-independent (JVM)	Designed for web development, embedded in HTML
Team Experience	High	Moderate	None
Syntax	Simple, readable, close to pseudocode	Verbose, strict typing	C-like, less strict, but can be messy
Performance	Slower than Java	High performance	Moderate performance
Use Cases	Data science, AI/ML, web apps	mobile apps (Android), large-scale backend	Web apps, CMS (WordPress, Drupal), e-commerce
Concurrency	Limited by Global Interpreter Lock	Strong concurrency with multithreading and JVM support	Basic concurrency support
Community Support	Very large, strong open source ecosystem	Very large, enterprise backing such as Oracle, OpenJDK	Large for a web developer
Web Framework	Django	Spring Boot	Laravel
Database Connectivity	Excellent support with SQLAlchemy and Django ORM	Strong JDBC ecosystem, enterprise-grade	Strong with MySQL, MariaDB, and PostgreSQL support
Real-TIme Support	Good with libraries	Strong with Java NIO, WebSockets	Supported Ratchet and Swoole, but less mainstream
Versatility	Very versatile	Very versatile	Mostly web-focused
Development Speed	Very fast due to simple syntax	Slower, more boilerplate, and setup needed	Fast for web apps
Difficulty Scale (1-5)	2	4	3

2.2.4 Backend Framework

Criteria	Django	Spring Boot	Laravel
Language	Python	Java	PHP
Complexity	Provides most tools built in, making it easier for teams to start quickly on complex projects	Requires more setup and boilerplate, but offers strong foundations for large and long-term applications	Moderate in difficulty, easier to pick up for PHP developers
Built-in Feature	Ships with an ORM, admin interface, user authentication, and strong security measures by default	Offers dependency injection, robust security modules, and seamless database connectivity	Includes Eloquent ORM, Blade templating engine, authentication, and built-in support for queues and caching
Development Progress	Fast and a wide range of ready-made tools	slow at first due to required setup, but pays off with long-term scalability	Fast; designed for developer productivity with expressive syntax
Community Support	Strong open source community and extensive documentation, and libraries	Extensive enterprise-level support and open-source community	Large PHP community, strong support for web development
Usage	Complex applications, APIs, AI/ML integrations, web platforms	Large-scale enterprise applications, financial systems, and microservices	Web application
Hosting Services	Heroku, PythonAnywhere, and AWS	AWS, Google Cloud, DigitalOcean	Shared hosting, AWS, DigitalOcean, Laravel Forge
Hosting Cost	Free tiers are available, or paid hosting starting at $5/month	Free tiers are available, or paid hosting starting at $5/month	Free tiers are available, or paid hosting starting at $5/month
Difficulty Scale (1-5)	2	4	3

2.2.5 Database Query Language

Criteria	SQL	NoSQL
Data Model	Tables with rows & columns	Documents, key-value pairs, graphs, or wide columns
Schemas	Fixed schema; changes require migrations	Dynamic schema; fields can vary between records
Scalability	Vertical scaling (add more power to one server)	Horizontal scaling (add more servers)
Query Language	SQL (Structured Query Language)	Varies (JSON, XML, YAML, etc.)
Transactions	Strong ACID guarantees (Atomicity, Consistency, Isolation, Durability)	Often BASE model (Basically Available, Soft state, Eventually consistent)
Performance	Good for complex queries and transactions	Great for high-speed reads/writes with large datasets
Flexibility	Rigid but reliable for structured data	Highly flexible; good for rapidly changing data
Use Cases	Financial systems, e-commerce, banking, CRM systems	Real-time analytics, social networks, caching, IoT
Examples	MySQL, PostgreSQL, Oracle, MS SQL server	MongoDB, Cassandra, Firebase, Redis, DynamoDB

2.2.6 SQL Dialects

Criteria	PostgreSQL	MySQL	SQLite	MariaDB
Open Source	Open source (PostgreSQL License, very permissive)	Open source (GPL), enterprise paid editions	Public domain, fully open-source	Open-source under GPLv2 (client libraries LGPL)
ACID Compliance	Full ACID compliance	ACID-complaint with InnoDB engine	ACID-compliant	ACID by default, mirrors MySQL behaviour
Standards Compliance	Highly standard-compliant (SQL: 2016 features)	Less strict, often prioritises speed over strictness	Mostly compliant but minimal feature set	Designed MySQL-compatible, but diverges in newer features
Advanced Features	Support for JSONB, arrays, window functions, and rich indexing options	JSON support, replication, clustering features	Lightweight, embedded, no client-server	Enhanced over MySQL: Galera, plugin engines, and better query optimisation
Performance	Excellent for complex queries and large datasets	Performs very well for read-heavy workloads	Lightweight, great for mobile or small apps	15-25% faster than MySQL in benchmarks, especially with complex queries
Scalability	Supports vertical and horizontal scaling options	Good scalability, wide support in hosting	Limited (single file DB, no clustering)	Multiple storage engines and built-in clustering (Galera) make scaling flexible
Best Use Cases	Complex, enterprise-grade apps; data warehouses	Web apps, CMS platforms, e-commerce	Mobile IoT, desktop apps	Great for existing MySQL ecosystems needing performance improvements or replication
Difficulty Scale (1-5)	3	2	1	2

2.2.7 Hosting Service

Criteria	Heroku	Supabase	Render	Railway
Types of Hosting	PaaS (Platform-as-a-Service)	DBaaS (Database-as-a-Service) + Auth + Storage	PaaS + DBaaS	PaaS + DBaaS
Suited For	Full-stack apps & quick prototypes; great for beginners and demos	Database-first apps, APIs, and projects needing built-in auth & storage	Full-stack web apps, APIs, static sites	Small-to-medium full-stack apps; easy for quick setups
Easiest Approach to Use	Push code via Git → app auto-deploys; add-ons for DB	Easy setup with dashboard & auto-generated APIs	Slightly more setup, great docs, GitHub-first	Simple templates, deploy in minutes, minimal setup
Cost	Free tier with limit to 550-1000 dyno hours per month; Paid starts around 7 dollars per month	Free tier with 500MB DB, Auth and Storage; Paid starts around 25 dollars per month	Free tier with 750 compute hours and free PostgreSQL given; Paid starts around 7 dollars per month	5 dollars per month free credit; usage-based billing with around 50 cents per GB storage
Performance	Good for prototypes, shared resources	Fast global scale network, scalable	Fast boot times, dedicated servers	Solid, lightweight for smaller apps
Deployment	Git-based deployment, pipelines, CLI	One-click project, instant backend setup	GitHub auto-deploy, CLI support	GitHub auto-deploy, CLI, templates
Built-in Features	Add-ons marketplace (logging, caching, etc.)	Auth, storage, real-time DB, functions	Free SSL, cronjobs, PostgreSQL, scaling	AutoSSL, secrets manager, metrics

2.3 Recommendations

To determine the most suitable configuration for this project, several alternatives were considered across programming languages, frameworks, database, hosting, platforms, and development tools. Each option was assessed against the factors identified in the Terms of Reference, namely client requirements, development time, developer familiarity, and cross-platform accessibility.

The team evaluated three main platform options for the proposed system, which are a native desktop application, a mobile-native application, and a web application. A desktop application offers great offline functionality, but would however limit accessibility to specific operating systems and require separate installers and updates. A mobile application is capable of providing an optimised mobile experience, but it would increase development time due to the need for separate builds for iOS and Android platforms. Therefore, a web application was chosen because it offers seamless cross-platform access through any browser without installation, updates can be done without user action, and rapid development.

As the proposed system is a web-based application, HTML will be utilised as the foundational language to define the structure and layout of the interface. To ensure a visually appealing and user-friendly design, CSS will be employed for styling, formatting, and responsive layout control. Furthermore, JavaScript will be incorporated in a minimal capacity, primarily to support basic interactivity and lightweight animations, thereby enhancing the overall user experience without adding unnecessary complexity.

Python was chosen as the backend language because it is widely known by the development team, easy to learn, and integrates directly with Django's built-in ORM database system. This allows faster development without the need to set up external database connectors. HTML and CSS are essential for structuring and styling the user interface, and since all team members are already familiar with them, they naturally fit the project. JavaScript will be used minimally for frontend interactivity, mainly in combination with HTMX.

Django was selected as one of the frameworks because it is a full-featured framework that provides built-in features such as authentication, security, an admin panel, and database integration, which reduce development time and improve reliability. Flask, while lightweight and flexible, would require additional configuration and external libraries to match Django's capabilities, making it less suitable under the project's time constraints. Tailwind CSS was

chosen for styling because it enables rapid development of a modern, professional interface that meets the client's request for a clean and visually appealing theme without requiring extensive custom CSS. HTMX was selected to handle interactivity such as card and list view toggling, providing a dynamic experience without the complexity of writing large amounts of JavaScript. React, although powerful, was not chosen because it introduces unnecessary overhead for this project, given that the client's design requirements are relatively minimal and do not demand a highly complex single-page application.

Both SQL and NoSQL options were considered for data management. Although NoSQL databases provide high scalability and flexibility for unstructured data, they require significant planning for consistency and are less suitable for projects with strong relational data requirements. SQL was chosen because the system will rely on structured, relational data with defined relationships, such as user accounts and course enrolments. Besides, SQL's compatibility with Django's ORM makes it a reliable choice for this project.

We compared MySQL, PostgreSQL, SQLite, and MariaDB for our SQL dialect. SQLite offers simplicity and easy setup but lacks advanced features and scalability for production environments. MySQL is widely used and beginner-friendly, but has more limited standards compliance. MariaDB is an open-source fork of MySQL that offers better performance optimisations, improved storage engines, and additional features while maintaining compatibility. It is a strong alternative for projects requiring scalability and high availability without licensing costs. PostgreSQL was selected because it is open-source, fully ACID-compliant, highly standards-compliant, and integrates seamlessly with Django's ORM, which enables advanced querying and scalability. Its strong community support and flexibility also make it well-suited for both development and production use.

The project considered hosting services such as Heroku, Supabase, Render, and Railway. Heroku provides a beginner-friendly deployment process but has limitations in its free tier and less competitive pricing for scaling. Supabase offers a strong focus on backend-as-a-service features and integrated database hosting but is more specialised and less flexible for traditional web application deployment. Railway is a strong contender with a modern developer experience, but its free tier and scaling options are less robust compared to Render. Render was chosen for its simplicity, automatic deployment pipelines, and ease of integration with Django and PostgreSQL, making it the most reliable and cost-effective choice for this project.

In summary, the evaluation demonstrated that Python, HTML, CSS, and JavaScript as the core languages, combined with Django, Tailwind CSS, and HTMX as frameworks and tools, offer the most suitable configuration for this project. The platform will be a responsive web application to ensure accessibility across desktop and mobile devices. PostgreSQL was chosen as the database for its advanced features, scalability, and seamless integration with Django, ensuring reliable data management. Render was selected as the hosting service for its cost-effectiveness, straightforward deployment, and strong compatibility with the chosen stack. This combination of languages, frameworks, database, hosting and platform offers the best balance between rapid development, professional design and client satisfaction.

Here are the final alternatives our team has chosen:

Application Platform	Web Application
Frontend Programming Language	HTML, CSS, JavaScript
Frontend Framework	Tailwind CSS, HTMX
Backend Programming Language	Python
Backend Framework	Django
Database Language	SQL
Database Management System (DBMS)	PostgreSQL
Hosting Service	Render

3.0 Deliverable - Risk Register

Risk involves uncertainty, or unintended consequences that may occur or not. When a risk is realised, it becomes a problem or issue. Hence, effective risk management is crucial to ensure that this project is completed successfully and on time. By detecting possible issues early on, the team may take proactive steps to reduce disruption and stay on track with the project plan. Each risk identified in this register is tailored to the project's scope and team operations, with particular monitoring, mitigation, and contingency plans in place to properly address them.

Risk Rating Methodology

Risks will be assessed at the beginning of each sprint planning session and monitored continually during the project. The Scrum Master is in charge of maintaining the risk register, and all team members are encouraged to report new risks as they arise. High-severity risks must be escalated to the Product Owner as soon as possible.

Likelihood:

- **1 (Low):** Unlikely to occur (less than 20% likelihood)
- **2 (Medium):** Possible but seldom (20-60% likelihood)
- **3 (High):** Very likely to occur (more than 60% likelihood)

Impact:

- **1 (Low):** Minor inconvenience with little impact on timeline or quality
- **2 (Medium):** Noticeable interruption with 1-2 tasks are delayed or require rework
- **3 (High):** Significant interruption that affects sprint goals, client deliverables, or necessitates extensive rework

Severity = Likelihood $\times$ **Impact**

- **1 - 3**: Low severity, can be monitored but take limited action
- **4 - 6**: Medium severity, active mitigation planning is required
- **7 - 10**: High severity, requires quick attention and strong contingency plan

To ensure accountability and transparency, the risk register will be reviewed and updated on a regular basis, with modifications recorded in the project plan.

ID	Date raised	Risk Description	Likelihood	Impact	Severity	Owner	Monitoring Strategy	Mitigation Plan	Contingency Plan
R1	15/8/2025	The client may request new or altered features after sprint work has already started, disrupting planned tasks and timelines.	2	3	6	Product Owner	Hold weekly sprint review meetings and track change requests	Use agile backlog refinement to re-prioritize tasks	Adjust the sprint scope and move non-critical tasks to the next sprint
R2	15/8/2025	The client may provide unclear or missing details about features, leading to misunderstandings and rework.	1	3	3	Product Owner	Clarify requirements during every client meeting and keep meeting notes	Conduct thorough requirement elicitation sessions in Weeks 2–4	Build a Minimum Viable Product (MVP) and iterate if uncertainty remains
R3	16/8/2025	Team members may be absent unexpectedly, causing delays in their assigned work and affecting overall progress.	2	3	6	Scrum Master	Check attendance and progress updates daily	Cross-train members on key tasks	Reassign workload to other members if someone is unavailable

R4	18/8/2025	The client may be dissatisfied with the UI/UX design, necessitating adjustments that could cause development delays.	2	2	4	UI/UX Designer	Present prototypes to the client during development reviews	Obtain client approval on prototypes before proceeding with full implementation	Redesign only the necessary pages within the project deadline
R5	20/8/2025	Team members may disagree on the design or technical approach, resulting in conflict and inefficient decision-making.	2	2	4	Scrum Master	Observe team engagement during stand-ups and retrospectives	Establish a structured decision-making strategy to resolve conflicts	Escalate unresolved issues to the teaching assistant or supervisor for prompt mediation
R6	22/8/2025	Tasks that are more difficult than anticipated may be deferred due to poor estimation or skill deficiencies.	2	3	6	Scrum Master	Monitor the progress of allocated tasks during scheduled meetings and weekly evaluations	Divide complex tasks into smaller deliverables and seek tutor guidance	Re-scope the sprint by reducing or eliminating non-critical features to maintain delivery timelines

R7	22/8/2025	Concurrent Git updates may result in code-level merge conflicts, which can lead to code overwrites or delays in committing stable changes.	2	2	4	Tech Lead	Conduct regular code reviews and keep an eye out for potential overlapping edits in pull requests	Use a strict branching approach and encourage frequent, smaller commits	If there are significant conflicts, rollback to the latest stable commit and reapply the conflicting changes to be reapply carefully
R8	24/8/2025	Delivered features may contain bugs, flaws, or be incomplete, affecting overall quality.	2	3	6	QA Tester	Perform both automated and manual testing regularly during the sprint	Adopt Test-Driven Development (TDD) procedures for important modules and develop detailed test cases	Set aside time for a bug-fix sprint, prioritising and resolving key issues before release
R9	24/8/2025	Delays could happen while executing non-critical or low-priority sections.	2	1	2	Scrum Master	Track backlog progress in sprint reviews and planning sessions	Define clear task prioritisation criteria, and make non-critical activities optional deliverables	Postpone non-critical tasks to guarantee that high-priority features are completed on time

R10	25/8/2025	Modules created by various team members may not integrate properly, bringing about unforeseen issues and project delays.	2	3	6	Tech Lead	Plan system-level integration tests at the end of each sprint to ensure compatibility	Implement continuous integration (CI) pipelines and establish integration checkpoints across subsystems	Allocate buffer time for debugging, and, if necessary, temporarily disable malfunctioning modules while core functionality is stable
R11	25/8/2025	The system might mishandle sensitive customer data or fail to meet accessibility criteria, raising ethical problems.	1	3	3	Tech Lead	During sprint reviews, ensure that all data handling regulations and accessibility guidelines are followed	Implement secure data management processes and educate the team on ethical principles	If a breach occurs, apply an immediate patch, notify stakeholders, and adjust protocols to avoid recurrence

AI Acknowledgement

We acknowledge the responsible use of OpenAI ChatGPT (GPT-5, August 2025 release) to assist with brainstorming ideas, structuring content, and refining written text for our project inception. All AI-generated outputs were critically reviewed, edited, and integrated by our team to ensure clarity, relevance, and academic integrity. We confirm that we have verified the accuracy and validity of the content and take full responsibility for all conclusions and findings derived from these AI-assisted outputs.

4.0 Reference

Agarwal, R. (2024, June 6). *Relational Databases: PostgreSQ Vs. MariaDB Vs. MySQL Vs. SQLite.* Strapi.
https://strapi.io/blog/relational-databases-postgresql-vs-mariadb-vs-mysql-vs-sqlite

Atlassian, B. (2025, April 24). *What is the Definition of Done?* Atlassian.
https://www.atlassian.com/agile/project-management/definition-of-done

EddyOJB. (2024, January 18). *Django with HTMX: Economical Modern App Development At Last.* Medium.
https://medium.com/@eddyojb/django-with-htmx-economical-modern-app-development-at-last-066b53825e1e

Eesuola, A. (2024, June 30). *SQLite vs PostgreSQL: A Detailed Comparison.* DataCamp.
https://www.datacamp.com/blog/sqlite-vs-postgresql-detailed-comparison?utm_cid=19589720824

Full Stack Developer. (2024, July 12). *Relational vs. NoSQL Database with example queries and ACID/BASE principles.* Medium.
https://medium.com/%40ByteCodeBlogger/relational-vs-nosql-database-with-example-queries-and-acid-base-principles-016b10db1482

HexShift. (2025, June 21). *How to Build a Progressive Django App with Tailwind CSS and HTMX*. Medium.
https://medium.com/@hexshift/how-to-build-a-progressive-django-app-with-tailwind-css-and-htmx-0d14b8330a9f

Janda, P. (2023, August 31). *Mobile app, web app, desktop app: know the difference!* Mobitech.
https://mobitouch.net/blog/mobile-app-web-app-desktop-app-know-the-difference

Laoyan, S. (2025, January 23). *The Importance of Sprint Planning in Agile Methodologies.* Asana.
https://asana.com/resources/sprint-planning-meeting

Layton, M, C., & Ostermiller, S, J. (2021, March 15). *Steps to Successfully Become an Agile Organization.* Dummies.
https://www.dummies.com/article/business-careers-money/business/project-management/steps-to-successfully-become-an-agile-organization-275120/

MacNeil, C. (2025, February 6). *How to Run An Effective Sprint Retrospective Meeting.* Asana.
https://asana.com/resources/sprint-retrospective

Microsoft. (2022, July 4). *Relational vs.NoSQL data -.NET.* Microsoft Learn. https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data

Nguyen, T. (2025, May 1). *Heroku vs Render vs Vercel vs Fly.io vs Railway: Meet Blossom, an Alternative.* BoltOps. https://blog.boltops.com/2025/05/01/heroku-vs-render-vs-vercel-vs-fly-io-vs-railway-meet-blossom-an-alternative/

PostgreSQL vs MariaDB - Overview and Insights. (n.d.). pgbench.com. https://pgbench.com/comparisons/postgres-vs-mariadb/

Railway vs Supabase. (2025, August 8). GetDeploying. https://getdeploying.com/railway-vs-supabase

render vs. heroku vs. vercel vs. railway vs. fly.io vs. aws. (n.d.). Ritza Articles. https://ritza.co/articles/gen-articles/render-vs-heroku-vs-vercel-vs-railway-vs-fly-io-vs-aws/

Wijer, J. (2023, June 28). *Choosing the Right Technology Stack for Your Web or Mobile Development Project.* Moqod. https://moqod.com/blog/the-right-technology-stack-for-your-project

Wilson, A. (2022, May 26). *Supabase vs Heroku Postgres.* Supabase. https://supabase.com/alternatives/supabase-vs-heroku-postgres

Yigal, A. (2018, November 8). *Sqlite vs. MySQL vs. PostgreSQL: A Comparison of Relational Databases.* Logz.io. https://logz.io/blog/relational-database-comparison/

5.0 Appendix

Google Drive Link:
https://drive.google.com/drive/u/1/folders/1-JfOmveriXM7HptvgoVSQkKz9s7iVF82

Jira Link:
https://academic-teaching-place.atlassian.net/jira/software/projects/FIT2101/boards/2

GitLab Repository:
https://git.infotech.monash.edu/fit2101/fit2101-s2-2025/group-repo/MA_Thursday10am_Team1