The University of Melbourne
School of Computing and Information Systems
SWEN90007 Software Design and Architecture
Semester 2, 2025

Project Overview

In this subject, we will explore various design principles and architectural patterns applicable to enterprise applications. In this project, you will have the opportunity to gain practical experience in the implementation of these patterns by developing a large-scale enterprise application.

The project aims to give you the experience to:

- Produce an architectural design for an enterprise system.
- Choose suitable design patterns to be applied to the different architectural layers.
- Implement an application for your design.
- Develop employability skills: teamwork, good communication, collaboration, problem-solving, critical thinking, adaptability, initiative and leadership.

Application Domain

Your task this semester is to develop a ride-sharing application that enables riders to request rides and drivers to accept and complete them.

Functionality

1. **User Registration**
   Users can create an account by providing their name, email, password and role (Rider or Driver). All fields are mandatory, and registered emails are expected to be unique.

2. **Authentication and Authorisation**
   Users can login and logout, with the appropriate dashboard/functionality shown based on the user's role (Rider or Driver).

3. **Driver Availability Schedule**
   Drivers can (optionally) configure a recurring weekly schedule (e.g., Mondays 9am-5pm) during which they are available to take on rides. If configured, a driver can only see and accept ride requests during their availability window. ~~If configured, a driver can see and accept rides at any time.~~ If NOT configured, a driver can see and accept rides at any time.

4. **View Rides (Driver)**
   A Driver sees a list of REQUESTED (unassigned) rides if they are within their availability window, or if they do not have a configured window. Otherwise, a user-friendly message indicating they are currently outside of their availability window is shown.

   For each ride, the following information must be accessible to the user:
   - Pick up location
   - Destination
   - Fare estimate
   - Requested time

5. **Accept Ride (Driver)**
   A Driver can select and accept a ride request if it is inside their availability window (or if they do not have a configured window). When a driver accepts a ride, the application must assign the driver to the ride and update the ride status to ACCEPTED.

6. **Begin Ride (Driver)**
   A Driver updates the status of a ride they have accepted to ENROUTE when the ride begins.

7. **Complete Ride (Driver)**
   A Driver updates the status of a ride to COMPLETED when the ride finishes.

   Specifically, on ride completion, all the following actions must take place:
   - Set the status of the ride to COMPLETED
   - Deduct the ride's fare from the rider's wallet
   - Credit the fare to the driver's wallet
   - Create a payment record that includes the amount, the ride reference, and the time when the payment took place.

8. **Request Ride (Rider)**
   A Rider can request a ride by specifying the pickup location and the destination. For simplicity, you can assume that an address consists only of a 4-digit postcode, as this is the only data needed to estimate a ride's fare.

   Once the pickup and destination locations are known, the application must calculate a ride fare based on a fixed-zone model and display a confirmation screen displaying this fare to the user. The user should then be able confirm or cancel the ride request. The request should only be successful if the Rider has sufficient funds in their wallet.

   Fare Calculation

   The fare is determined by the postcodes of the pickup and destination locations according to the following rules, which must be checked in order:
   - Airport Fare: If either the pickup or destination postcode is for Melbourne Airport (3045), the fare is a fixed $60.00. This rule overrides all other zone rules.
   - Interstate Fare (Zone 3): If either the pickup or destination is outside Victoria (approximated by postcodes not in the 3xxx), the fare is a fixed $500.00.
   - Regional Fare (Zone 2): If the trip is not an Airport or Interstate trip, and either the pickup or destination is in Regional Victoria (any Victorian postcode not in the Metro Melbourne range), the fare is a fixed $220.00.
   - Metro Fare (Zone 1): If none of the above rules apply, the trip is considered a Metro Melbourne trip. The fare is a fixed $40.00.

   For the purpose of this project, "Metro Melbourne" is defined by the postcode range 3000-3299 inclusive. All other Victorian postcodes (e.g., 3300-3999) are considered "Regional".

9. **Cancel Ride (Rider)**
   A Rider can cancel a ride after requesting it, but before it is ENROUTE. The status of the ride should be updated to CANCELLED, after which drivers should not be able to accept it.

10. **View and Update Wallet (Driver and Rider)**
    Riders can view and credit their wallet balance. Drivers can only view their wallet balance, but they cannot update it.

11. **Ride History (Rider)**
    Riders can view the list of rides they have requested. For each ride, the following information must be available to the user:

- Pickup
- Destination
- Fare paid (optional)
- Timestamp of payment (optional)
- Driver's name (optional)
- Status

Note that the fields marked as optional are displayed depending on the ride's status.

12. Ride History (Driver)

Drivers can view a list of rides they have accepted. For each ride, the following information must be available to the user:

- Pickup
- Destination
- Fare received (optional)
- Timestamp of payment (optional)
- Rider's name
- Status

Note that the fields marked as optional are displayed depending on the ride's status. Also, note that the fare received by the Driver is the same as the fare paid by the Rider.

Ride Lifecycle

Based on the requirements defined above, a ride can be in one of the following states:

- REQUESTED
- ACCEPTED
- ENROUTE
- COMPLETED
- CANCELLED

In addition, state transitions must only take place via valid paths:

- REQUESTED -> ACCEPTED
- ACCEPTED -> ENROUTE
- ENROUTE -> COMPLETED
- REQUESTED -> CANCELLED

Concurrency Issues

1. Concurrent cancel and accept

A Rider cancels a ride while one or more Drivers attempt to accept it. This could result in a CANCELLED ride being ACCEPTED.

2. Concurrent wallet top-up and payment deduction

A Rider tops up their wallet while payment is being deducted for ride completion. This might lead to an inconsistent balance in the Rider's wallet.

Programming Language and Architecture

The system you design and develop must be web-based.

The **back-end** (i.e., server-side components) must be implemented in **Java (no exceptions)**. To ensure you gain hands-on experience with architectural patterns, the use of high-level application frameworks is **not allowed**. You cannot use any framework that provides dependency injection, inversion of control (IoC), or built-in Object-Relational Mapping (ORM). Whenever in doubt, get in touch with the teaching team on Ed discussion board for additional information. The goal is for you to implement key architectural and design patterns (e.g. Controller, Data Mapper, Unit of Work, ...) from scratch, not to use a framework where the patterns are already built in.

We recommend the use of Servlets to expose back-end services. This is the approach that is supported by the teaching team and is the approach used in the sample template projects provided to you as learning resources. If you choose to, you are allowed to expose these services via a REST API instead, however, please note that this approach is not officially covered in the subject content and will not be directly supported by the teaching team. Use of REST APIs is at your own discretion and risk.

For the **front-end** (i.e., UI/client-side components) of your application, you are free to use any technology/framework/library of your choice. Lectures covering material relevant to the front-end may include JSP and/or React examples. To help you get started, we will provide 2 template projects: one based on JSP (which integrates with a Servlet back-end) and one based on React. Please note that the React template project will be the **only form of support provided for JavaScript frameworks**, as they are outside the primary scope of this subject. You are free to use any other framework, but integrating it with the Java back-end is your responsibility. **We do not recommend you make use of different front-end frameworks, but you are allowed to. Again, we will not provide assistance in these cases.**

All **design diagrams** (e.g., domain model, class diagram, sequence diagrams) **must be in UML**.

Plagiarism

All submitted work must represent the genuine and original contribution of the student or team. The use of Generative AI (GenAI) tools is permitted **only if appropriately acknowledged**, and students must clearly document **what was used, how it was used, and why**. Unacknowledged or excessive reliance on such tools will be treated as a breach of academic integrity.

To assure learning and maintain the integrity of assessment, **all students will be required to individually demonstrate and explain** their contributions, design decisions, and code during scheduled project demos. This may include responding to targeted questions related to specific patterns, implementation choices, and rationale. Students unable to confidently explain their work may be subject to further investigation or academic review.

Example GenAI Use Disclosure (GitHub Wiki)

Some sections of our project documentation and code comments were drafted with the assistance of ChatGPT (model version). Specifically, we used GenAI tools to help:

- *Reword explanations of the XYZ pattern in our Github Wiki.*
- *Generate a starting point for validating email input in our Rider registration form.*

- *All code and architectural decisions were critically reviewed, modified, and implemented by the team. No part of the project was copied without understanding or adaptation.*

All code and documentation submitted as part of this project must be your own team's work; it must be based on your own knowledge and skills. All students must abide by the University's academic integrity policy. Further resources are available here: https://academicintegrity.unimelb.edu.au/.

Collaboration between teams is not allowed; copying from the work of another team (whether in this year's offering of the subject or from previous years) constitutes academic misconduct and will be treated as such. Similarities will be determined by Turnitin and Moss (Measure of Software Similarity – source code).

Teams

This assignment will be carried out in teams of 4, no exceptions. Team members must be enrolled in (and attend) the same workshop. This is because the workshops have been structured to support your teamwork, e.g., by allowing time for the team to work on the project and consult queries with the tutor.

This is a semester-long project that will require constant interaction and contribution from all team members; we strongly encourage you to create your own teams and make a conscious choice of whom you will be working with. You are free to team up with friends or peers you are already acquainted with. **Students without a team will be randomly allocated into teams by Eduardo and Maria.**

A survey to submit your team details is available here: Team Registration

Use of Code Repositories and Collaborative Tools

For this project, we will adopt GitHub Organisation. **Your team and repository will be created by the teaching team** and will be available here: https://github.com/SWEN90007-2025-sem2 (**you'll be able to access this space once you receive our invitation in Week 2 – do not create your own repositories**). Please read further instructions on how to gain access to the subject's GitHub organisation here: https://canvas.lms.unimelb.edu.au/courses/215895/assignments/582841

Your repository must be kept private.

Once your team and repository are created, we want you to make use of the **GitHub Wiki** to document your project throughout the semester. All major design decisions, architectural choices, key implementation notes, and team workflows should be captured and regularly updated in the Wiki. This forms the **official project documentation**, which will be reviewed as part of your assessment. At each submission point, your team must export the current Wiki content as a single PDF, add it to your repository, and create a **GitHub release tag** that includes both the documentation and committed code. **There will be no submissions via Canvas**; your tagged release will serve as the formal submission. You will receive support on using GitHub Wiki and release tags during lectures and tutorials.

We also strongly encourage you to explore the **Projects** tab available on your GitHub team space for internal project management.

This repository must be used for the duration of your project to manage documentation, source code, and related configuration items. Make sure you structure your Wiki and repository properly.

Example of repository structure:

```
├── docs/       # Docs (you can create subfolders to organize your project reports and minutes)
├── │── part1/       # Part1 documentation (use this folder for Part 1A and Part 1B)
├── │── part2/       # Part2 documentation
├── │── part3/       # Part3 documentation
├── │──data-samples/     # Data (inputs) necessary to simulate/demonstrate your project
├── src/         # Source code
└── README.md
```

A README file must explain your GitHub repository: what is included, a link to your deployed app, changelog, details about team members and so on.

At the end of each deliverable, generate a RELEASE TAG from your repository (master branch).
The tag must have the following format: SWEN90007_2025_Part<number>_<team name>.

Keep in mind that your **GitHub repository will also be used to assess individual contributions**. This means that it is very important that you commit and push your work frequently. It is also good practice to do this, as it protects your work from accidental crashes or deletions!

Team Responsibilities, Demos and Assessment of Individual Contribution

All team members are expected to contribute equally across all aspects of the project, from early design discussions and documentation to implementing front-end and back-end features. It is the team's responsibility to manage and monitor task distribution fairly throughout the semester. **Teaching staff will not be involved in dividing work among members.**

To support equitable collaboration, students will complete a group review using the FeedbackFruits tool, which will be made available on Canvas shortly before each submission deadline. Through this tool, each student will evaluate their own contribution and the contributions of their team members. **Extensions will not be granted for group reviews.** After the group review deadline, students will be able to view the feedback received and provide responses or clarifications directly in the tool (that's why we cannot provide extensions here). **Important**: Students who fail to complete the group review will miss the opportunity to contribute to our evidence-based assessment process and may also affect their own marks, as they will not be able to assess or justify their own contributions.

In cases where contributions differ significantly, and following a review by the teaching team, **individual marks may be adjusted**: marks will be scaled up for students who over-contribute, and down for those who under-contribute.

In addition, project **demonstrations will be used to assess individual understanding**. During demos, each student will be asked targeted questions related to the system's design, implementation, and architecture. These will be assessed individually and aligned, where possible, to each student's contributions. To support this, students are strongly encouraged to maintain clear records of work using the GitHub Wiki and the Projects tab (task management board). These tools should document who worked on what, and when. If such documentation is missing or unclear, staff will allocate topics and questions to students at their own discretion. These decisions will not be negotiable.

If a key architectural or design pattern is missing in the deliverable, the entire team may be penalised (as the delivery is incomplete). However, if a student is unable to explain or justify a pattern, design

decision, or feature that was implemented and documented, only that student will receive a reduced mark for the demonstration component of the deliverable. Maintaining transparency and accountability throughout the project is essential for fair and accurate assessment.

Teamwork during Workshops

Workshops will be mostly dedicated to project teamwork. You are encouraged to think of this time as your team's weekly meeting (or at least one of them). You can use the time to plan activities for the week ahead, work collaboratively on a task, discuss any issues that need to be resolved, etc. Most importantly, your team should use this time to seek clarification and guidance from the tutor.

You must take minutes and document during these workshop team meetings in your GitHub Wiki. The minutes should contain:

- Workshop date/week.
- Team members that attended.
- A few bullet points summarising the team's activities during the workshop meeting.

These minutes must be exported from GitHub Wiki and included in your releases (docs/meeting-minutes/) and can be used as part of the individual contribution process (if disputes arise).

Part 1A [8 marks]

Task

Based on the provided application domain and use cases, you will design your application's domain model and present it in a written report by means of a domain model diagram and an accompanying description.

Deliverables

1. A report exported from GitHub Wiki, in pdf format, that includes the following:
   - Your team's name, team members' names, student ids, Unimelb usernames, GitHub usernames and emails.
   - A domain model diagram.
   - A description of your domain model.
   - A GitHub release tag: you must create and push a release tag to your GitHub repository for this deliverable in the following format: SWEN90007_2025_Part1A_<team name>[1]. The tag must be created and pushed before the submission deadline, as this will be used to assess your deliverable (no exceptions).
   - This report must be available in your GitHub (docs/part1).

Part 1B [12 marks]

Task

In this part you will implement and deploy a partial use case. The aim is for your team to decide on a technology stack, define the overarching architectural design of the application, gain practical

[1]Git Basics – Tagging: https://git-scm.com/book/en/v2/Git-Basics-Tagging

experience and insight into what it takes to implement a use case, and get over the hurdle of deploying a distributed cloud-based application.

Functionality:

- Schedule Availability – Driver (3)
  In this part, you are only required to implement the functionality that allows a Driver to configure a recurring weekly schedule. There is no need to differentiate between roles at this stage, or apply any other business rules (e.g., what rides are visible depending on a Driver's availability window) related to this use case.

  From a UI perspective, we expect a screen where the user selects the availability window, a button to save/confirm the window, and user-friendly feedback to the user indicating either success or an appropriate error message.

  The availability window must be persisted in the database. Your implementation must also ensure user inputs are correct.

Design patterns and principles:

1. Layered Architecture
   Your application must be structured with a clear separation of concerns across the following layers: presentation, service, domain model, and data source.
   Please note that you are not required to implement the data mapper pattern at this stage. You can have a single class representing your data source layer where the database is accessed from.

2. Domain Model Pattern
   You must implement any entities in your domain model that are relevant to the given use case.

Deliverables

1. The use case coded and deployed. The source code of your application must be committed to your GitHub repository. The application must be deployed in Render.
2. You must create a tag in your GitHub repository for this deliverable in the following format: SWEN90007_2025_Part1B_<team name>. The tag must be created and pushed before the submission deadline as this will be used to assess your deliverable (no exceptions). After creating the release tag, you deploy this tag (not your main repository branch).
3. README.md including the link to your deployed app.

Demonstration

Your team will be required to attend a demo session with a tutor. ALL members must attend this session. The intention of this session is to assess the following:

- Ability to deploy and integrate a front-end, back-end, and database.
- Functionality (your deployed use case).
- Your overall design and architecture, including implementation, understanding, and ability to argue for the choices you have made.

Although the demonstration is conducted as a team, each student will be asked individual, targeted questions based on their involvement. To support this process, teams are expected to maintain clear records of individual contributions using the GitHub Wiki and task boards (GitHub Projects tab). This information may be used to inform which questions are directed to each student. We do not expect a strict one-to-one mapping between students and specific patterns as all students are expected to engage with multiple aspects of the project to demonstrate a broad understanding. In the absence of contribution records, or in cases where workload is poorly balanced, staff will allocate questions at their discretion.

Part 2 [40 marks]

Task

This part requires you to implement a subset of the requirements outlined in the Application Domain. It also requires that you design, architect and implement your application based some of the patterns and principles taught in the subject.

Functionality:

- User registration (1)
- Authentication and Authorisation (2)
- Schedule availability - Driver (3)
  You must complete the functionality implemented in Part 1B to ensure configuring a schedule is optional, and that drivers can only see and accept rides during their availability windows (if configured).
- View rides – Driver (4)
- Accept ride - Driver (5)
- Request ride – Rider (8)

Design patterns and principles:

1. Layered Architecture
   Your application must be structured with a clear separation of concerns across the following layers: presentation, service, domain model, and data source.

2. Domain Model Pattern
   Your core business logic must use the Domain Model pattern by defining rich domain objects that encapsulate behaviour, not just data.

3. Identity Field
   You must use the Identity Field pattern to maintain unique database identifiers in your domain objects. The choice of key should be a well-thought-out and deliberate design choice.

4. Foreign Key Mapping
   You must use Foreign Key Mapping where appropriate, based on the design of your domain and relational models.

5. Inheritance Pattern
   You must implement one of the inheritance patterns covered in the subject. You must choose an appropriate pattern based on your domain and relational models. This choice must be a deliberate and well-thought decision.

6. Data Mapper
   The data source layer must use Data Mappers.

7. Authentication and Authorisation
   These features must be implemented based on the patterns learnt in the subject. You are strongly encouraged to use an existing security framework for this purpose.

At this stage, you will assume that your application is only accessed by one user at a time (i.e., there is no need to handle concurrency yet).

Deliverables

1. The application coded and deployed. The source code of your application and updated documents must be committed to your GitHub repository. The application must be deployed in Render.
2. You must create a tag in your GitHub repository for this deliverable in the following format: SWEN90007_2025_Part2_<team name>. The tag must be created and pushed before the submission deadline, as this will be used to assess your deliverables (no exceptions). After creating the release tag, you deploy this tag (not your main repository branch).
3. A report exported from GitHub Wiki in PDF format with the following items:
   - The class diagram of your application.
   - For each use of Foreign Key Mapping, a partial class diagram accompanied by a partial relational diagram depicting the implementation of the pattern.
   - A partial class diagram accompanied by a partial relational diagram depicting the implementation of the inheritance pattern.
   - A contextualised sequence diagram that illustrates the implementation and the use of data mappers. This can be done for a use case of your choice.
   - This report must be available in your GitHub (e.g., docs/part2).
4. README.md updated to include the link to your Render-deployed app.
5. A prepopulated database.
   - Your deployed app must include a populated database with a range of realistic data samples necessary for the teaching team to test your application. You can document the data available in your deployed app, add it to docs/data-sample, and highlight the existence of the document in your README.md.
   - You must also provide explicit instructions on how to use any existing data in your system. For example, the email addresses and passwords of existing users, along with their corresponding roles. We do not expect to have to create any test data from scratch when assessing your app. Ensure that you test your deliverable before submission and that you have realistic/appropriate data in the deployed system. Meaningless data such as abc, 123, blah blah, ASDFG, and so on may compromise the assessment of your project (and your final marks).

Demonstration

Your team will be required to attend a demo session with a tutor. ALL members must attend this session. The intention of this session is to assess the following:

- Functionality (your deployed app).
- Your implementation of the patterns.

- Your understanding of the patterns. This includes your ability to explain your implementation, the role of the patterns, argue for any specific choices you have made, and contrast pattern/design choices against other choices.
- Your overall design and architecture, including implementation, understanding, and ability to argue for the choices you have made.

Important notes:

- **The demo is not only about showcasing a working app. It is also intended to evaluate your understanding of the patterns and principles taught in the subject**, and how you applied that knowledge to design, architect, and implement your application to meet the specified requirements.
- **For the UML diagrams, it is your responsibility to decide the level of detail required to appropriately convey the design and implementation of your application**. Use your judgment and knowledge in software engineering to make this decision; another software engineer, new to your team, should be able to implement/understand your design based on the information contained in the diagrams.
- **Although the demonstration is conducted as a team, each student will be asked individual, targeted questions based on their involvement**. To support this process, teams are expected to maintain clear records of individual contributions using the GitHub Wiki and task boards (GitHub Projects tab). This information may be used to inform which questions are directed to each student. We do not expect a strict one-to-one mapping between students and specific patterns as all students are expected to engage with multiple aspects of the project to demonstrate a broad understanding. In the absence of contribution records, or in cases where workload is poorly balanced, staff will allocate questions at their discretion.

Part 3 [40 marks]

Task

In this part, you will finalise the design and implementation of the application's functionality. You will also incorporate concurrency management to address issues that arise when multiple users access the application simultaneously.

Functionality:

- Begin Ride – Driver (6)
- Complete Ride – Driver (7)
- Cancel Ride – Rider (9)
- View/Update Wallet (10)
- View Rides – Rider (11)
- View Rides – Driver (12)

Design patterns and principles:

1. Layered Architecture
   Your application must be structured with a clear separation of concerns across the following layers: presentation, service, domain model, and data source.

2. Unit of Work

A Unit of Work must be used to coordinate and manage updates made by business transactions, where appropriate.

3. Concurrency Mechanisms
   You must design and implement concurrency management strategies for the concurrency issues 1 and 2 (as outlined in the application domain). For each issue, you can choose to address concurrency by relying on any of the mechanisms learnt in the subject, including database-level constructs (e.g., SELECT for UPDATE), pessimistic offline locks, optimistic offline locks, or a combination of them. The key is to select the mechanism that best addresses the identified problem, considering both the application requirements and the trade-off between liveness and consistency.

Deliverables

1. The application coded and deployed. The source code of your application must be committed to your GitHub repository. The application must be deployed in Render.
2. You must create a tag in your Git repository for this deliverable in the following format: SWEN90007_2025_Part3_<team name>. The tag must be created before the submission deadline, as this will be used to assess your deliverables (no exceptions). After creating the release tag, you deploy this tag (not your main repository branch).
3. A report exported from GitHub Wiki in PDF format with the following items:
   - A contextualised sequence diagram that illustrates the implementation and the use of the Unit of Work pattern. If needed, this can be done for a use case of your choice.
   - For each of the concurrency issues, a contextualised sequence diagram illustrating how concurrency is managed through the chosen concurrency mechanism.
   - For each concurrency issue, a testing strategy and outcome(s). The testing strategy must be robust in terms of the scenarios/cases covered and the level of concurrency (e.g., it should not rely solely on concurrent application use via the UI). How to present the outcome of a test case is up to you, you can rely on text, screenshots, tables, etc.
   - This report must be available in your GitHub (e.g., docs/part3).
   - This report must be available in your GitHub (docs/part3).
4. README.md updated to include the link to your Render-deployed app.
5. A prepopulated database (as for Part 2).

Demonstration

The expectations for the demo of Part 3 are the same as those for Part 2. ALL members must attend this session.

Please note that you should be prepared to describe each concurrency issue and how concurrency occurs, provide a rationale for your choice of concurrency mechanism, contrast your choice against alternative mechanisms, and explain how you have implemented your chosen mechanism.

You must also be prepared to carry out your testing strategy for each concurrency issue. For example, you may be asked to execute your JMeter (or similar, depending on your choice of tool) script during the demo session and discuss the outcome with the tutor. The outcome will also influence the mark associated with the implementation/functionality of the mechanism/use case.

Submissions

There will be no submissions via Canvas.

For the source code and reports, we will assess your (GitHub) RELEASE TAGS for marking purposes. Make sure you generate the tags, push them to your GitHub repository, and deploy them before the assignment deadline.

The following are the project deliverables (submissions) throughout the semester, and the marks associated with the different submissions (adding up to a total of 100 marks).

Deliverable	Marks	Due Week	Due Date
Part 1A	8	3	Friday 15 Aug, 11:59pm
Part 1B	12	4	Friday 22 Aug, 11:59pm
Demonstration Part 1B		During Week 5	
Part 2	40	9	Tuesday 23 Sep, 11:59pm
Demonstration Part 2		During Week 9	
Part 3	40	12	Tuesday 21 Oct, 11:59pm
Demonstration Part 3		During Week 12	

Feedback will be provided for each stage within two weeks of submission.

Extensions

Extensions will only be granted if students are able to **provide appropriate supporting documentation**. If an extension is needed, you must contact the lecturer as soon as possible, **before the submission date**. It will not be possible to apply for an extension after the submission date.

For further details, please see:
https://ask.unimelb.edu.au/app/answers/detail/a_id/5667/~/applying-for-an-extension

Late Submissions

Unless an extension is in place, a 10% penalty will be applied to every day (or part thereof) you delay your submission (RELEASE TAG, and/or deployment).

Assessment Criteria

The goal of this project is for you to apply and demonstrate a clear understanding of the design principles and architectural patterns covered in the subject.

Your project will be assessed based on the following high-level criteria:

- Complete implementation of the functionality specified in the application domain.
- Clear, complete, and correct presentation of the application's design and architecture using UML.
- Correct implementation of the patterns.
- Design quality and rationale. This includes the appropriate selection of patterns (if applicable), sound reasoning behind their selection, the ability to contrast them against alternative patterns, and accurate explanations of the design principles and patterns used.

Note that proper use of UML notation is expected. Inaccuracies in your UML diagrams will affect your mark. Also, your implementation must match the design presented in the documents. Any discrepancies between code and documentation will impact your mark.

Please be aware that simply producing an application that provides the required functionality is not enough to pass the assessment.

Assessment breakdown

Part 1A

Item	***Criteria***	***Marks***
Domain model diagram and description	• Modelling of the problem at hand • Use of UML notation • Appropriate level of detail and abstraction • Accurate description of domain model diagram	8
	Total	**8**

Part 1B

Item	***Criteria***	***Marks***
Full stack deployment	• Deployment of an integrated front-end, back-end, and database	5
Use case implementation	• Coded and deployed use case	3
Design and architecture	• Use of layering with clear separation of concerns • Adequate implementation of domain model entities	4
	Total	**12**

Part 2

Item	***Criteria***	***Marks***
Class diagram	• Use of UML notation • Appropriate level of detail and abstraction • Coherence with implementation	3
Patterns (domain model, identity field, foreign key mapping, inheritance, data mapper, authentication, authorisation)	• Implementation and use • Understanding and knowledge • Rationale for choice/use of pattern • Ability to contrast against alternatives (where applicable)	20
Layering, design, and architecture	• Use of layering with clear separation of concerns • High-quality design and software practices (e.g., low coupling, high cohesion, readable code, modularity, low duplication, consistent naming, etc.)	6
Diagrams (foreign key mapping, inheritance, data mapper)	• Adequate contextualisation (i.e., not generic diagrams, but specific to your implementation) • Correct use of UML notation • Appropriate level of detail and abstraction • Coherence with implementation	3
Functionality	• Working use cases as defined in application domain	8

Total	***40***

Part 3

Item	*Criteria*	*Marks*
Layering, design, and architecture	• Use of layering with clear separation of concerns • High-quality design and software practices (e.g., low coupling, high cohesion, readable code, modularity, low duplication, consistent naming, etc.)	4
Unit of Work	• Implementation and use • Understanding and knowledge	5
Concurrency Issue 1	• Implementation • Understanding and knowledge • Rationale for choice/use of mechanism • Ability to contrast against alternatives • Testing strategy and outcome	9
Concurrency Issue 2	• Implementation • Understanding and knowledge • Rationale for choice/use of mechanism • Ability to contrast against alternatives • Testing strategy and outcome	9
Diagrams (UoW, concurrency issue 1, concurrency issue 2)	• Adequate contextualisation • Correct use of UML notation • Appropriate level of detail and abstraction • Coherence with implementation	3
Functionality	• Working use cases as defined in application domain	10
	Total	***40***