**Sample WAD Project Report**

Cover Page:

Module Name

Tutor

Module Leader

Student Name

Student ID

Date

Word count

Table of contents

Table of Figures

**Guidelines and Requirements**

a report describing how you developed the code, including details of how your code works, any problems encountered, and how you solved them. This should be around 250-500 words per task; simpler tasks will require less, while more complex tasks will require more. (40%)

**Scenario**

Your task is to develop **EventEase**, an **event booking platform** that allows users to search for events in different locations, view event details, and book tickets for available events. The system will manage event listings, available seats, and reservations.

You must accompany your work with a report of around 1000-2000 words which should be written before you start coding.

This should include:

- a written analysis of the requirements of each task and what techniques, from the module’s material, are needed to code the task. This should then lead into a technical discussion of how you would code the task, making use of the material learnt in class. Accompany this discussion with code extracts. This must be done

before you code the corresponding task.

- a description of problems encountered and explanation of how you solved them. This should be done after you have coded the application.

For each task, the write-up will be between 250 and 500 words, the exact length depending on the complexity of the task.

If you are using React for the front end, you do not also have to discuss a non React implementation.

***Important!***

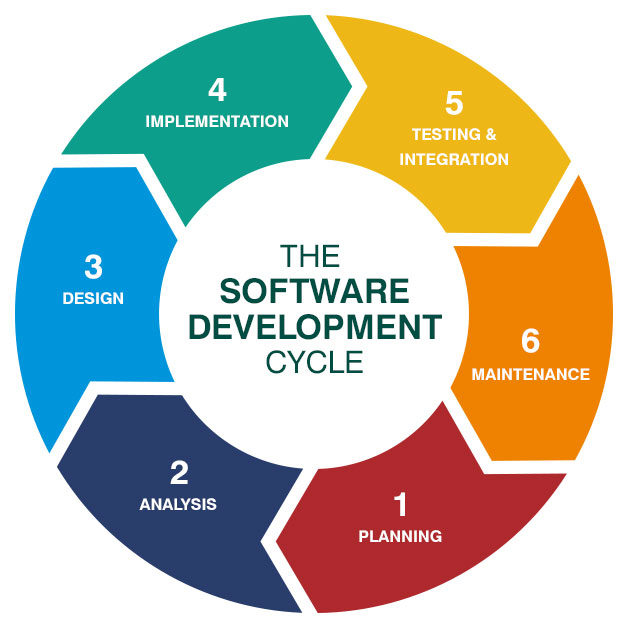
The report must not read like a specification of your code. It must clearly show your thought processes when analysing the requirements and writing the report; if it does not, you may receive a fail grade for the report component of the assessment.

You do not need to use formal analysis and design artefacts (use cases, sequence diagrams or class diagrams, for example).

Introduction

Short description of the project, requirements and main tasks and expected deliverables

Methodology (SDLC Processes)



**SDLC**

The Software Development Life Cycle (SDLC) is a well-structured process that guides software development projects from start to finish. It provides a clear framework for planning, building, and maintaining software, ensuring that development is systematic and meets quality standards.

Defining the specific SDLC stages ensures that development is organized and executed effectively, resulting in high-quality software that meets user requirements. Following a structured approach, development teams can reduce risks, optimize resources, and produce software that aligns with business goals– all within a reasonable timeframe.

**The 6 phases of the software development life cycle**

The SDLC process typically consists of several key phases, each contributing to the successful development of software. The main SDLC phases include planning, implementation, testing, and deployment, but that’s not all.

Each phase plays a crucial role in effectively designing the software, meeting user needs, and ensuring timely delivery.

Phase 1: Planning

The planning phase is the foundation of any successful software development project. Project goals, objectives, and requirements are gathered and documented during this phase. Project requirements can be based on customer feedback or market research evaluating existing product options. Stakeholders work together to define the project scope, establish timelines, and allocate resources. Planning establishes the project's direction, ensuring that all participants have a clear understanding of what needs to be done and how to achieve it.

Phase 2: Feasibility Analysis

Once planning is complete, the feasibility analysis phase begins. During this phase, the project team evaluates whether the project is technically and financially viable. This includes assessing the technical requirements, estimating costs, and performing a risk analysis. Risk assessment is essential to identifying potential challenges and determining if the project is worth pursuing.

Phase 3: System design

The system design phase includes creating the software's architecture and design. Based on the requirements gathered during planning, the team creates a blueprint outlining how the software will function. This includes high-level architecture and detailed design specifications, including user interface design to ensure the software is user-friendly and an assessment of requirements for compatibility with existing products.

Phase 4: Implementation

The implementation phase, also known as the development phase, transforms the design into a functional application. It is here that the actual coding takes place. Developers write the code based on the design specifications, following best practices and coding standards to ensure the result is efficient, secure, and maintainable.

Phase 5: Testing and Integration/Deployment

Testing: The testing phase is critical because it generates essential performance and usability feedback while revealing defects and quirks. Various [types of software testing](https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing) can be used, including [automated testing](https://www.atlassian.com/continuous-delivery/software-testing/automated-testing), unit testing, integration testing, and system testing. The goal is to identify and fix bugs, ensuring the software operates as intended before being deployed to users.

Deployment: Once internal [software testing](https://www.atlassian.com/continuous-delivery/software-testing) is complete, the solution can be deployed to end users. This typically includes a beta-testing phase or pilot launch, limited to a select group of real-world users. Depending on the project's needs, deployment can be done on-premise or in the cloud. The deployment strategy determines how easily users can access and use the software.

Phase 6: Maintenance and Documentation

The last phase of the SDLC is maintenance. Even after the software is deployed, ongoing support is necessary to address issues, apply updates, and add new features. Continuous maintenance ensures that the software remains functional and relevant over time.

**Database Design and Integration**

DB Description, tables and relationships

ER Diagrams

**Project Tasks Implementation**

**Part A – Develop a very simple REST API**

Tasks 1:

Description

Code Snippet

A computer screen with text

AI-generated content may be incorrect.

*Figure 1: Code snippet of the project main elements and imported libraries*

(Figure Reference and Caption )

Implementation Screenshot (Figure Reference and Caption )

Testing Screenshot (Figure Reference and Caption )

Reflections (Challenges, Limitations and Testing)

Tasks 2:

Description

Code Snippet

Implementation Screenshot

Testing Screenshot

Reflections (Challenges, Limitations and Testing)

**Part B – Develop a simple AJAX-based JavaScript front-end**

Tasks 5:

Description

Code Snippet

Implementation Screenshot

Testing Screenshot

Reflections (Challenges, Limitations and Testing)

**Part C – Adding simple error-checking**

**Part D – Adding a map**

**Part E – logins and sessions**

**Part F – Implementing Additional Features**

**Part G – Implementation Using React**

**Part H – Middleware, DAOs, controllers and routers**

Reflections and description of the approach to adapt a modular approach in developing the various components of the project

**Project Special Instructions (if any)**

**Security Measures**

**Conclusions**

**References**

**Appendix**