UMGC Capstone Project Proposal Management System (CaPPMS)

Project Plan

Version 3.0

Prepared By:

Bereket Tamrat

Ephrem Kefle

Kathryn Stewart

Marc Bueno

Tarun Lava

Yonas Mekete

Project Plan Approvals

Name	Signature	Date
Professor Dr. Mir Assadullah		
Project Manager Kathryn Stewart		

Revision History

Date	Version	Description
09/08/2020	1.0	Initial Project Plan Draft
09/29/2020	2.0	Section 3.2: Updated Project Software Specification Table Section 4.1 tables: Removed client approvals for all deliverables except SRS Section 8: Updated entire section
11/3/2020	3.0	Section 9: Added Software Test Plan from separate document into this document.

Table of Contents

List of Figures

List of Tables

1. Introduction

This Project Plan outlines the details necessary to successfully develop the UMGC Capstone Project Proposal Management System (CaPPMS) project. This document outlines the project intent, expected deliverables, constraints, and assumptions regarding the project development. It also outlines how the project will be managed include team organization, roles and responsibilities, risk and change management.

1.1. Statement of Need

A platform is needed where clients and customers can submit capstone project proposals for future University of Maryland Global Campus (UMGC) Software Engineering (SWEN) 670 students. Those projects would then go through an approval process involving UMGC professors and stakeholders who would review and determine which project could be completed by the students.

The UMGC Capstone Project Proposal Management System (CaPPMS) is a web-based application which allows customer, clients, and former students to submit detailed proposals of projects to be designed and implemented by UMGC SWEN 670 students as well as track the stages during the approval process.

1.2. Project Vision

To create a repository of projects for development which provide cost-effective software solutions that benefit the community, students, and the institution.

1.3. Customer

The customer for this project is UMGC Software Engineering Capstone professors. The sponsor is Dr. Michael Brown, the Program Chair of the Software Engineering Programs.

1.4. Definitions, Acronyms and Abbreviations

Below are the terms and abbreviations used in this document.

Table 1 Acronyms

Acronym	Definition
CaPPMS	Capstone Project Proposal Management System
CCB	Change Control Board
ERD	Entity Relationship Diagram
EVM	Earned Value Management
IMS	Integrated Master Schedule
PM	Project Manager
PMI	Project Management Institute
RACI	Responsible, Accountable, Consulted, or Informed
SDLC	Software Development Life Cycle
SRS	Software Requirement Specification
STP	Software Test Plan
SWEN	Software Engineering
UMGC	University of Maryland Global Campus
WBS	Work Breakdown Structure

2. Project Scope

2.1. Scope Statement

The purpose of this project is to develop a web application that serves as a channel for external entities to submit project proposals to UMGC faculty. The scope of this paper is to provide detailed guidelines for planning, organizing, and managing project schedules, resources, and risks to successfully meet the project objectives. Furthermore, the paper describes how the team communicates and manages any changes in the project and lists the project deliverables at each milestone.

2.2. Project Deliverables

The UMGC Capstone Project Proposal Management System should include the following core functions:

- Submit – fill out the initiation form and submit project proposals to subject matter experts and stakeholders for review and approval.
- Collect Feedback – internal stakeholders will review and comment on proposals.
- Store Projects – a repository for the project proposals.
- Monitor – view the status of project proposals and track actions.
- Change – update proposals with new information or update the status.
- Approval – confirm proposal is acceptable for the class and export the details to share.

2.3. Project Approvals/Acceptance Criteria

The approval process will be managed by the project manager (PM). Internally all documents and code will be reviewed by the entire team.

The following documents will need to be submitted for approval to the clients:

- Project Plan
- Software Requirement Specification (SRS)
- Technical Design Document
- Software Test Plan
- Test Report

Additional artifacts will be submitted to the instructor for grading in accordance with the previous section.

2.4. Project Constraints

The project should be completed by November 3, 2020, and a working web-based application along with all the required documents should be delivered within the scheduled timeline.

2.5. Project Assumptions

The user interface will be web based. The proposal submitter is not required to create a login. There is only a single role that will support both usage and administration. The defined role is required to login to perform read, write, and delete functions.

2.6. Out of Scope

A number of items were reviewed and considered out of scope for this development effort.

- The webpage will not report the status of submitted projects to external users.
- A separate administration role will not be implemented.
- The system will not provide a way for professors to display specific project proposals to students. This will be managed through export.
- The system will not provide a way for professors to break down proposals into multiple phases or sub-projects. This will be a manual process.
- The system will not provide a way for professors to filter or search through project proposals by title, sponsor emails, categories, potential value or benefits. Other search options are included.
- Exported files will not be re-imported.
- Files will not be exported in XML or JSON for integration with other systems.
- The system will not provide a separate view for students to access the system.
- Students will not be able to vote for specific projects.
- Frequently Asked Questions will not be modifiable in the GUI. This must be changed in the database.
- The system will not log messages or audit user activities.

3. Project Approach

Software Development Life Cycle (SDLC) is a process followed by software industries to design, implement and test software products. SDLC process is followed strictly to ensure quality products are delivered.

The Agile-Waterfall hybrid model will be utilized for this project. The team discovers that it is the perfect solution for our project. The team will also find ideas for incorporating various combinations of the hybrid model depending on the project needs.

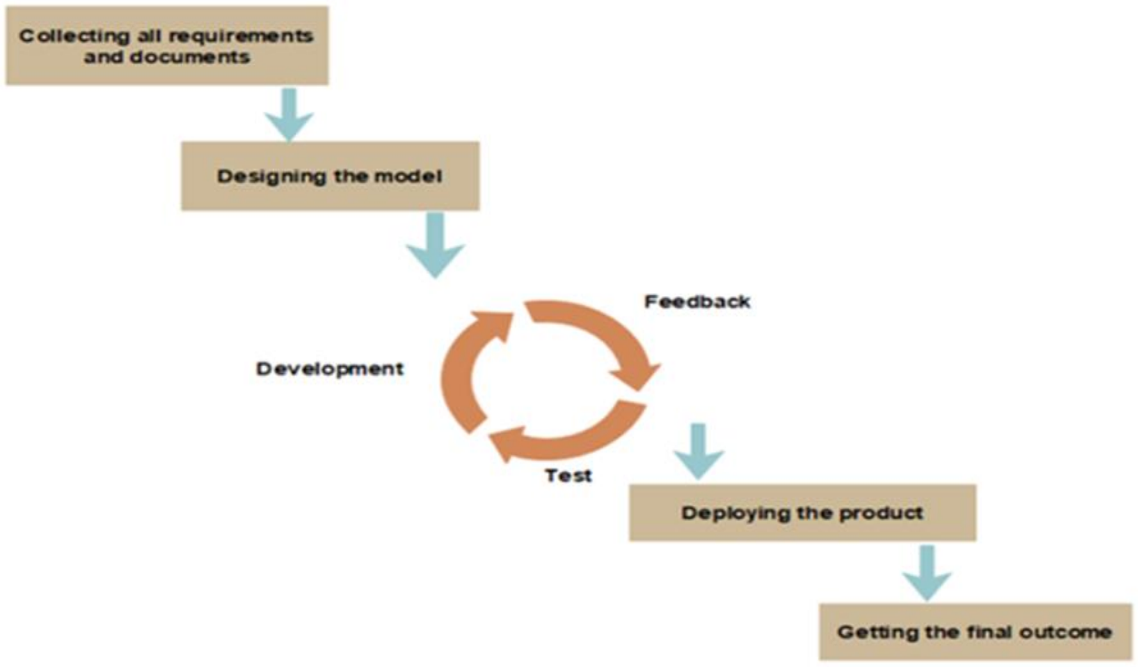

Collecting all requirements and documents
Designing the model
Feedback
Development
Test
Deploying the product
Getting the final outcome

Figure 1 Hybrid Agile and Waterfall Methodology

3.1. Test Plan

The test plans will be delivered with each milestone starting from the milestone 2 delivery. These test plans will be based upon the requirements outlined in the SRS. The test plan will include a requirements matrix that identifies where each requirement will be tested.

3.1.1. Test Approach

The tests will include unit test, system testing, and integration testing. Testing will include sunny day scenarios unless an exception is needed to verify a specific requirement. Test cases will be written to cover multiple requirements in a single test case. They will be grouped together based on related functionality.

3.1.2. Defect Management

Any defects identified during testing will be recorded in GitHub. These issues will include the following information:

- Title
- Description
- Steps to Reproduce
- Any related artifacts

The issues will be reviewed by the project manager, given a priority and status, and assigned to a developer for resolution by a specific date. These priorities and status will be defined per the tables below.

Table 2 Defect Priority Definitions

Priority	Definition
Low	Not urgent or trivial. Minimal business impact.
Medium	Time sensitive. Non-critical functionality loss or the software is still usable with a workaround.
High	Time critical. Substantial loss of service or data loss.

Table 3 Defect Status Definitions

Status	Definition
Open	Defect has been created and is awaiting a developer to correct.
Closed	Defect has been resolved with completed work and testing. State must be approved by the PM.
Work in Progress	Defect is being investigated and corrected by a developer.
Test in Progress	Defect resolution complete and testing is in progress to confirm.

3.2. Software Specification

Capstone Project Proposal Management System will be using the following solution software.

Table 4 Project Software Specifications

Function	Tool
Operating System	Window 10 and above for development, Linux for deployment
Database Server	PostgresSQL
Language and Framework	Java (Eclipse) / HTML / CSS / JavaScript
Webserver	Springboot
Repository	GitHub
Development Hosting	Azure
Production Hosting	Azure

4. Project Schedule and Deliverables

4.1. Deliverables and Milestones

The deliverables are defined by the syllabus (Assadullah, 2020) There are four milestones associated with this project. Each milestone defines the deliverables and is outlined below. In addition to these milestones, there are at least two client meetings that must be held after submissions for milestones 1 and 4.

Table 5 First Milestone

Deliverable	Due Date	Client Approval	Professor Approval
Project Plan	9/8/2020	No	Yes
Software Requirement Specification	9/8/2020	Yes	Yes

Table 6 Second Milestone

Deliverable	Due Date	Client Approval	Professor Approval
Project Plan Updates	9/29/2020	No	Yes
Software Requirement Specification Updates	9/29/2020	Yes	Yes
Technical Design Document	9/29/2020	No	Yes
Software Test Plan	9/29/2020	No	Yes

Table 7 Third Milestone

Deliverable	Due Date	Client Approval	Professor Approval
Project Plan Updates	10/20/2020	No	Yes
Software Requirement Specification Updates	10/20/2020	No	Yes
Technical Design Document Updates	10/20/2020	No	Yes
Programmer Guide	10/20/2020	No	Yes
Deployment and Operations Guides	10/20/2020	No	Yes
Software	10/20/2020	No	Yes

Table 8 Forth Milestone

Deliverable	Due Date	Client Approval	Professor Approval
Project Plan Updates	11/3/2020	No	Yes
Software Requirement Specification Updates	11/3/2020	Yes	Yes
Technical Design Document Updates	11/3/2020	No	Yes
Programmer Guide Updates	11/3/2020	No	Yes
Deployment and Operations Guides Updates	11/3/2020	No	Yes
Finished Software	11/3/2020	No	Yes
User Guide	11/3/2020	No	Yes

Deliverable	Due Date	Client Approval	Professor Approval
Test Report	11/3/2020	No	Yes
Video demonstrations	11/3/2020	No	Yes

4.2. Work Breakdown Structure

The initial work breakdown structure is outlined in the table below. The WBS will be maintained within the IMS. The data will be included in the baseline and will be monitored for modifications. The tables are separated into the tasks associated with each milestone, the weekly earned value management (EVM) reporting, and the presentations required after each milestone.

Table 9 Work Breakdown Structure – Milestone 1

WBS	Task Name
1	CaPPMS
1.1	Project Start
1.2	Inception/Analysis - Milestone 1
1.2.1	Project Plan
1.2.1.1	Team Charter
1.2.1.2	Project Vision
1.2.1.3	Define Scope
1.2	Inception/Analysis - Milestone 1
1.2.1	Project Plan
1.2.1.1	Team Charter
1.2.1.2	Project Vision
1.2.1.3	Define Scope
1.2.1.4	Define Resources
1.2.1.5	WBS/Schedule
1.2.1.6	Risks/Assumption
1.2.1.7	Define Logistics/DevOps
1.2.1.8	Define Approval Process

WBS	Task Name
1.2.1.9	Review Project Plan with Team
1.2.1.10	Review Project Plan with Mentor
1.2.1.11	Obtain Signatures
1.2.2	Software Requirement Specification
1.2.2.1	Gather Requirements
1.2.2.2	Write Introduction/Description
1.2.2.3	Use Case Definitions
1.2.2.4	Use Case Diagrams
1.2.2.5	Review SRS with Team
1.2.2.6	Review SRS with Client
1.2.2.7	Obtain Signatures
1.2.3	Milestone 1 Due
1.2.4	Presentation Preparation

Table 10 Work Breakdown Structure – Milestone 2

WBS	Task Name
1.3	Design/Test - Milestone 2
1.3.3	Technical Design Document
1.3.3.1	Human Interface Design
1.3.3.2	Requirements Matrix
1.3.3.3	ER Diagram
1.3.3.4	Data Dictionary
1.3.3.5	Architectural Design
1.3.3.6	Component Design
1.3.3.7	Class Methods
1.3.3.8	Review design with client
1.3.3.9	Review design with team
1.3.3.10	Obtain Signatures

WBS	Task Name
1.3.4	Software Test Plan
1.3.4.1	Unit Test Plan
1.3.4.2	Integration Test Plan
1.3.4.5	Review Test Plan with team
1.3.4.6	Review Test Plan with client
1.3.4.7	Obtain Signatures
1.3.5	Milestone 2 Due
1.3.6	Presentation Preparation
1.3.8	Software Development
1.3.8.1	Create Prototype
1.3.8.2	Unit testing for prototype
1.3.8.3	Review prototype with team
1.3.8.4	Review prototype with client
1.3.12	Document Updates
1.3.12.1	Project Plan updates
1.3.12.2	SRS updates
1.3.12.3	Review updates with team
1.3.12.4	Review updates with client
1.3.12.5	Obtain Signatures

Table 11 Work Breakdown Structure – Milestone 3

WBS	Task Name
1.4	Code/Guides - Milestone 3
1.4.2	Deployment and Operations Guide
1.4.2.1	Deployment Guide
1.4.2.1.1	Software
1.4.2.1.2	DevOps Integration
1.4.2.1.4	User Creation/Updates

WBS	Task Name
1.4.2.1.5	Database
1.4.2.2	User Guide
1.4.2.2.1	Database
1.4.2.2.2	Front End
1.4.2.3	Review Deployment & Ops Guide with team
1.4.2.4	Review Deployment & Ops Guide with client
1.4.3	Software Development
1.4.3.1	Build Database
1.4.3.2	Develop Code - Back End
1.4.3.3	Develop Code - Front End
1.4.3.4	Unit Debugging/Updates
1.4.3.5	Unit Test Creation/Execution
1.4.3.7	Integration Testing
1.4.3.8	Integration Debugging/Updates
1.4.3.9	Review Code with client
1.4.3.10	Review Code with Team
1.4.4	Milestone 3 Due
1.4.5	Presentation Preparation
1.4.9	Programmer Guide
1.4.9.1	Write Programmer Guide
1.4.9.2	Review Program Guide with team
1.4.9.3	Review program guide with client
1.4.20	Document Updates
1.4.20.1	Project Plan updates
1.4.20.2	SRS updates
1.4.20.3	Review updates with team
1.4.20.4	Review updates with client
1.4.20.5	Obtain Signatures
1.4.20.6	Software Test Plan Updates

WBS	Task Name
1.4.20.7	Design Document Updates
1.4.20	Document Updates
1.4.20.1	Project Plan updates
1.4.20.2	SRS updates
1.4.20.3	Review updates with team
1.4.20.4	Review updates with client
1.4.20.5	Obtain Signatures
1.4.20.6	Software Test Plan Updates
1.4.20.7	Design Document Updates

Table 12 Work Breakdown Structure – Milestone 4

WBS	Task Name
1.5.1	Final Software
1.5.1.1	Review Code with client
1.5.1.3	Review Code with Team
1.5.1.4	Unit Debugging/Updates
1.5.1.7	Database Updates
1.5.1.8	Front End Code Updates
1.5.1.9	Back End Code Updates
1.5.1.10	Unit Test Updates/Execution
1.5.4	Videos
1.5.4.1	Download Software
1.5.4.2	Configure Software
1.5.4.3	Install Software
1.5.4.4	Software Feature Walkthrough
1.5.5	Milestone 4 Due
1.5.6	Test Report
1.5.6.1	Test Report

WBS	Task Name
1.5.6.3	Unit Testing Results
1.5.6.4	Integration Test Execution
1.5.6.8	Obtain Signatures
1.5.9	User Guide
1.5.9.1	User Guide Updates
1.5.9.2	Review guide with team
1.5.9.3	Review guide with client
1.5.10	Presentation Preparation
1.5.24	Document Updates
1.5.24.1	SRS updates
1.5.24.2	Review updates with team
1.5.24.3	Review updates with client
1.5.24.4	Obtain Signatures
1.5.24.5	Software Test Plan Updates
1.5.24.6	Design Document Updates
1.5.24.7	Programmer Guide Updates
1.5.24.8	Deployment & Ops Guide Updates
1.5.24.9	SRS updates
1.5.24.1	Review updates with team

Table 13 Work Breakdown Structure – EVM

WBS	Task Name
1.6	EVM
1.6.1	Create and Populate EVM Tracker Spreadsheet
1.6.2	Week 2
1.6.3	Week 3
1.6.4	Week 4
1.6.5	Week 5
1.6.6	Week 6

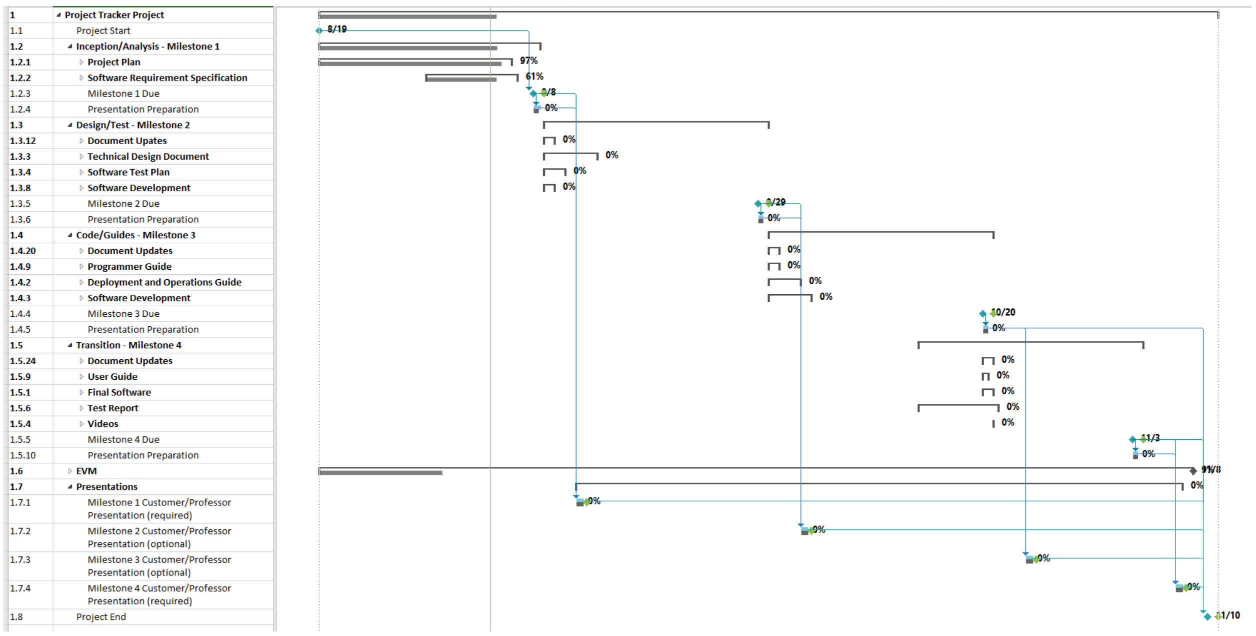
WBS	Task Name
1.6.7	Week 7
1.6.8	Week 8
1.6.9	Week 9
1.6.10	Week 10
1.6.11	Week 11
1.6.12	Week 12

Table 14 Work Breakdown Structure – Milestone Presentations

WBS	Task Name
1.7	Presentations
1.7.1	Milestone 1 Customer/Professor Presentation (required)
1.7.2	Milestone 2 Customer/Professor Presentation (optional)
1.7.3	Milestone 3 Customer/Professor Presentation (optional)
1.7.4	Milestone 4 Customer/Professor Presentation (required)
1.8	Project End

4.3. Project Schedule

The Gantt chart below shows the tasks that are associated with each milestone.



1	Project Tracker Project
1.1	Project Start
1.2	Inception/Analysis - Milestone 1
1.2.1	Project Plan
1.2.2	Software Requirement Specification
1.2.3	Milestone 1 Due
1.2.4	Presentation Preparation
1.3	Design/Test - Milestone 2
1.3.12	Document Upates
1.3.3	Technical Design Document
1.3.4	Software Test Plan
1.3.8	Software Development
1.3.5	Milestone 2 Due
1.3.6	Presentation Preparation
1.4	Code/Guides - Milestone 3
1.4.20	Document Updates
1.4.9	Programmer Guide
1.4.2	Deployment and Operations Guide
1.4.3	Software Development
1.4.4	Milestone 3 Due
1.4.5	Presentation Preparation
1.5	Transition - Milestone 4
1.5.24	Document Updates
1.5.9	User Guide
1.5.1	Final Software
1.5.6	Test Report
1.5.4	Videos
1.5.5	Milestone 4 Due
1.5.10	Presentation Preparation
1.6	EVM
1.7	Presentations
1.7.1	Milestone 1 Customer/Professor Presentation (required)
1.7.2	Milestone 2 Customer/Professor Presentation (optional)
1.7.3	Milestone 3 Customer/Professor Presentation (optional)
1.7.4	Milestone 4 Customer/Professor Presentation (required)
1.8	Project End

Figure 2 Project Schedule

5. Project Resource Management Plan

The resources needed for CaPPMS involve the student's work. To quantify and track these resources, different roles have been created with associated costs. These costs were retrieved from Indeed.com. The resources were added to the integrated master schedule (IMS). A breakdown of the roles and costs per hour is provided.

Table 15 Resource Costs per Hour

Role	Cost
Project Manager	$62.43
Analyst	$47.88
Developer	$47.29
Tester	$47.29

5.1. Project Team Organization

The project team has the following team members:

Bereket Tamrat, bereket.m.tamrat@gmail.com, 678-830-5591

Ephrem Kefle, efysha@gmail.com, 240-549-1015

Kathryn Stewart, kathryns222@gmail.com, 571-286-7937

Marc Bueno, mcbueno2k@hotmail.com, 416-209-1515

Tarun Lava, tejalava@gmail.com, 240-549-0389

Yonas Mekete, ymekete@yahoo.com, 301-801-7957

Additional support will be needed from the DevOps team. The DevOps team details are not included.

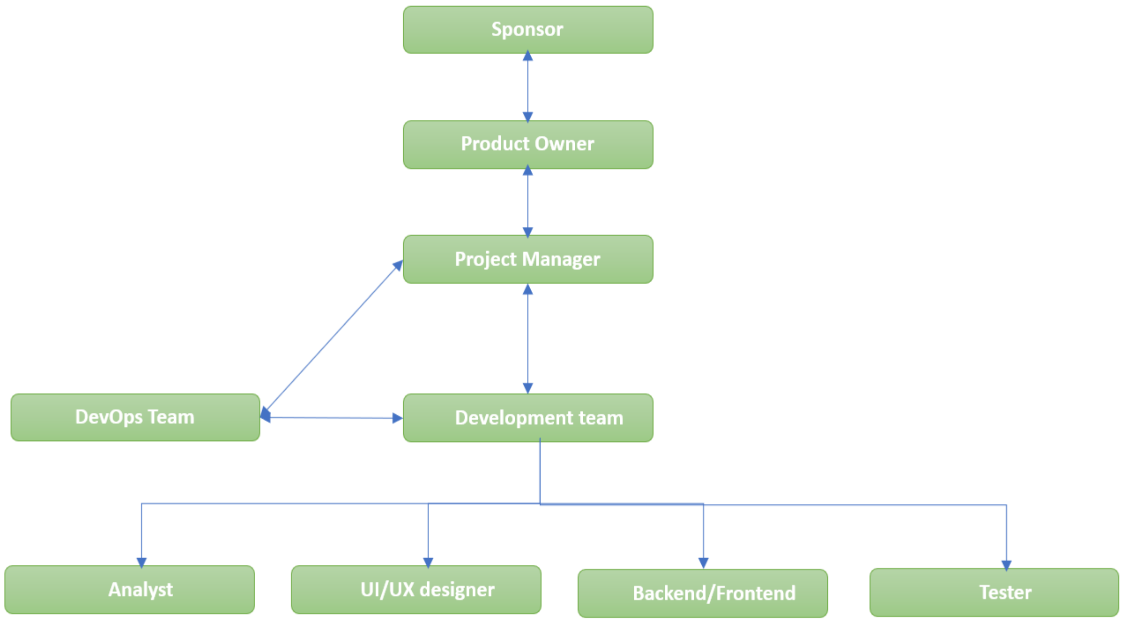


Figure 3 Project team organization chart

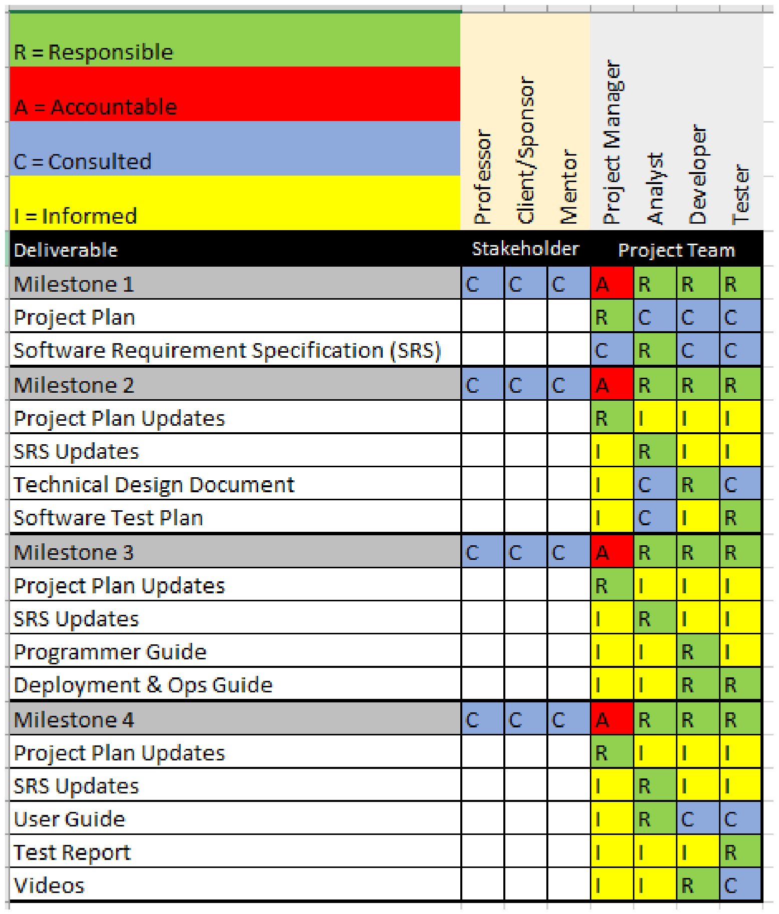
5.2. Project Team Roles & Responsibilities

Table 16 Roles and Responsibilities

Role	Description	Name
Project Manager	Responsible for keeping the project on track, the required artifacts are being generated, coordination between multiple parties, and interface to the clients	Kathryn Stewart Tarun Lava
Analyst	Responsible for requirements gathering, writing specifications, and other supporting documentation. Writing and diagraming use cases.	Yonas Mekete Marc Bueno Tarun Lava Ephrem Kefle
Developer	Responsible for generating code compliant to the requirements and unit testing associated with the code. Create and document the design artifacts compliant to the requirements.	Marc Bueno Ephrem Kefle Bereket Tamrat
Tester	Responsible for generating test plans, assisting with unit test generation, writing integration tests, and raising bugs.	Yonas Mekete Kathryn Stewart

5.3. RACI Matrix

The roles have been broken down further to indicate those Responsible, Accountable, Consulted, or Informed (RACI) for each document generated for this project.



R = Responsible
A = Accountable
C = Consulted
I = Informed

Deliverable	Stakeholder			Project Team			
	Professor	Client/Sponsor	Mentor	Project Manager	Analyst	Developer	Tester
Milestone 1	C	C	C	A	R	R	R
Project Plan				R	C	C	C
Software Requirement Specification (SRS)				C	R	C	C
Milestone 2	C	C	C	A	R	R	R
Project Plan Updates				R	I	I	I
SRS Updates				I	R	I	I
Technical Design Document				I	C	R	C
Software Test Plan				I	C	I	R
Milestone 3	C	C	C	A	R	R	R
Project Plan Updates				R	I	I	I
SRS Updates				I	R	I	I
Programmer Guide				I	I	R	I
Deployment & Ops Guide				I	I	R	R
Milestone 4	C	C	C	A	R	R	R
Project Plan Updates				R	I	I	I
SRS Updates				I	R	I	I
User Guide				I	R	C	C
Test Report				I	I	I	R
Videos				I	I	R	C

Figure 4 RACI Matrix

5.4. Status Reporting

During the twice a week meeting with the team status will be noted in the integrated master schedule. This updating is required so that on Sunday the earned value management report can be generated and submitted to the professor.

6. Risk Management Plan

6.1. Risk Overview

A project risk is defined by the Project Management Institute (PMI) as an uncertain event or condition that, if it occurs, may have a positive or negative effect on a project's objectives. Risk management is an ongoing process that is conducted throughout the life of the project. The process begins with identifying, assessing, and developing response plans for significant risks. It continues with regular risk monitoring, ongoing identification of new risks, and timely implementation of mitigation plans.

Risk management processes address internal risks (those under the control of the project team, such as quality of deliverables, cost, schedule, or technical risks) as well as external risks (those outside the control of the project team such as catastrophes, legislations, etc.).

6.2. Risk Management Process

Project risks are solicited during project meetings, interviews, and workgroups. The project team is using a straightforward method that includes the following process steps:

- **Identify risks**: Create a list of known project risks.
- **Analyze risks**: Determine the consequence of risks listed and prioritize them.
- **Plan**: Develop a response strategy and assign responsibility to manage the risks.
- **Track**: Review and re-examine risks at key project milestones.
- **Control**: Implement the defined response strategies as required.
- **Communicate**: When the project situation changes, the project team will discuss and review the risks in the scheduled meetings on an ongoing basis throughout the life of the project.

6.3. Risk Identification, Analysis, and Response

The risk identification process involves determining which risks might affect the project and documenting their characteristics. Once project risks and opportunities have been identified, an analysis will be performed to determine relative priorities and to develop a prioritized risk list for planning the appropriate level of response to the risks.

A qualitative analysis will be performed on each risk. After an initial prioritization, a decision will be made by the project team on whether the risk warrants more detailed analysis using quantitative techniques to further assess the probability and potential impact of the risk event on the project objectives. A probability value is determined using the likelihood of occurrence (unlikely, likely, and very likely), based on analysis by the project team.

An impact value is determined using the guidelines below. The table below provides an overview of the Risk Impact Values.

Table 17 Risk Impact Values

Impact	Dimensions to Consider			
	Cost	Schedule	Scope	Quality
Low	Little (<10%) to no impact on project cost	Little to no impact on the project schedule	Minor clarification to the existing scope	Project quality is not in danger
Medium	Impact on project cost is between 10% and 20%	Schedule impact is possible	Scope change is noticeable, but not considered significant	Impact on project quality is possible
High	Impact on project cost is greater than 20%	Schedule and deliverable due dates will be impacted	Scope change is considered significant	Impact on project quality is very likely

Based on the above risk probability and impact values, the project team has identified and analyzed the following risks. The team will follow the mitigation plan to reduce the likelihood or impact of the risk.

The risks will be analyzed based on the following matrix and will have an overall risk value assigned. Any medium or high risks will have mitigations plans.

Table 18 Risk Matrix

		Consequence		
		1 Low	2 Medium	3 High
Likelihood	1 Low	1 Low	2 Low	3 Medium
	2 Medium	2 Low	3 Medium	4 High
	3 High	3 Medium	6 High	9 High

Table 19 Risk Evaluation

ID	Risk	Likelihood	Consequence	Risk Score
1	Project scope creep	Low	High	Medium
2	Project schedule not clearly defined	Low	Medium	Low
3	Incomplete requirements resulting in need of re-coding some sections	Low	High	Medium
4	Requirements Inflation	Low	High	Medium
5	Lack of team commitment	Low	Medium	Low
6	Lack of client acceptance; the client asks for minor rework	Medium	High	High

Table 20 Risk Mitigation Plans

ID	Risk	Mitigation Plan	Impact if Realized
1	Project scope creep	Engage stakeholder early and confirm the project scope Ensure the schedule and resource plan is focused on completion of the defined deliverables	Additional scope will increase project costs. Schedule may increase in duration to implement changes.
3	Incomplete requirements resulting in need of re-coding some sections	Engage stakeholder early and often to confirm the project scope	Rework will increase the project costs.
4	Requirement Inflation	Ensure that features not originally requested will not be added Discuss with the client if they want to keep changes that add value and not supersede the initially envisioned features	Additional scope will increase project costs. Schedule may increase in duration to implement changes.
6	Lack of client acceptance; the client asks for minor rework	Make sure client’s requirements are implemented Provide an update and demo to the client at each development phase	Additional work will increase project costs. Schedule may increase in duration to implement changes.

7. Change Management Plan

Change management is an important part of a successful project. A change management plan helps manage the change process if any, and ensures control in schedule, scope, communication, and resources. This change management plan will minimize the impact a change can have on the project, resource, and other important stakeholders.

The Change Management process establishes an orderly and effective procedure for tracking the submission, coordination, review, evaluation, categorization, and approval for release of all changes to the project's baselines.

For changes that are identified, a change request must be submitted to the team via email. This email must contain the following details:

- Title
- Description
- Submitter
- Phone
- Email

These details will be entered into the GitHub repository by the program manager in the issues section where a date and number will be assigned. The issue raised will include the tag 'requirements' and be assigned to an analyst for review.

The priority definitions will be defined as high, medium, or low per the definitions below.

Table 21 Change Priority Definitions

Priority	Definition
High	This requirement change must be implemented in order for the project to be considered successful.
Medium	This requirement change should be implemented in order to function as expected by the user.
Low	This requirement change is nice to have and will be included as time and resources allow.

Once the analyst has completed the review, the change control board (CCB) will meet to discuss, review, and confirm acceptance, partial acceptance, or rejection of each change. The board will include the following roles.

Table 22 Change Control Board Roles

Role	Name	Contact	Definition
Client	Dr. Michael Brown	240-684-2439 michael.brown@umgc.edu	End user of the project
Project Manager	Kathryn Stewart	571-286-7937 kathryns222@gmail.com	Responsible for understanding and monitoring implications to scope and costs of requested changes. Will update the schedule and costs to reflect the changed scope if accepted.
Analyst	Yonas Mekete	301-801-7957 ymekete@yahoo.com	Responsible for understanding and monitoring implications to other requirements. Will write new use cases and/or test cases to reflect the changed scope
Developer	Ephrem Kefle	240-549-1015 efysha@gmail.com	Responsible for implications to the development process. Technical analysis to understand the changed scope to the rest of the project and determine feasibility.

8. Communication Management Plan

8.1. Introduction

This plan outlines how communication will occur within the scope of the CaPPMS project. This plan defines the roles and how these roles will interact. A communications matrix is included which maps the requirements.

8.2. Approach

The project manager will be the main driver in ensuring effective communications on the project with the stakeholders and team.

A Microsoft Teams channel has been setup for the teams' exclusive notes. It includes a chat function, a meeting function, and includes a repository where documents can be stored and reviewed by the team. This shall be the primary means of communication. Alternate means of communication include emails or text messages using the Team Member's provided details in section 5.1.

The project manager will set up regular Teams meetings twice a week, Wednesday and Saturday. Meeting minutes will be taken and stored on Teams. Actions will be recorded and assigned in the Microsoft To-Do applications setup for the team. The regular meetings will include the following objectives:

- Review the status of the project
- Plan for the next week and distribute workload
- Discuss changes in project tasks and milestone deliverables

At each milestone, the project manager will schedule a meeting with the mentor to review the completed work and get feedback. The PM will also set a meeting with the stakeholder for document review and signatures one week prior to the milestone delivery.

The PM will reach out to DevOps for code review and standard document requirements such as programmer's guide as needed.

8.3. Key Stakeholders

The key stakeholders for the CaPPMS project are outlined below.

Table 23 Key Stakeholders

Role	Name	Contact
Client	Dr. Michael Brown	240-684-2439 michael.brown@umgc.edu
Professor	Dr. Mir Assadullah	Mir.Assadullah@faculty.umgc.edu
Project Management Mentor	Roy Gordon	uspsrgordon@aol.com
DevOps Mentor	Johnny Lockhart	jlockhart2@student.umgc.edu
Technology Mentor	Jaime Tello	tellojaime@hotmail.com
Project Manager	Kathryn Stewart	571-286-7937 kathryns222@gmail.com
Co-Project Manager	Tarun Lava	240-549-0389 tejalava@gmail.com
Project Team	See section 5.1	See section 5.1

8.4. Communication Matrix

Table 24 Communication Matrix

Type	Objective	Medium	Frequency	Audience	Owner	Deliverable
Kickoff Meeting	Introduce team and project. Review objectives & approach	Teams	Once	Project Team	Project Manager	Agenda Meeting minutes
Initial Requirement Meeting	Introduce client & discuss requirements and scope of project	Teams	Once	Client Project Team PM Mentor	Project Manager	Questions for client Meeting minutes
Project Team Meetings	Review status with the team	Teams	Twice weekly	Project Team	Project Manager	Schedule Meeting minutes
Technical Design Meetings	Discuss technical design solutions	Teams	As needed	Project Team	Project Team	Meeting minutes
DevOps Support Meetings	Discuss DevOps topics to facilitate development	Teams	As needed	Project Team DevOps Mentor	Project Team	Meeting minutes
EVM Reports	Share project status	Email	Weekly	Client Professor PM Mentor	Project Manager	Report

Type	Objective	Medium	Frequency	Audience	Owner	Deliverable
Milestone Deliverables	Files needed to complete the milestone delivery	Email SWEN 670 Assignment Submission	Once per milestone	Client Professor PM Mentor	Project Manager	Milestone Deliverables
Milestone Meetings	Review the status of the project, present deliverables due in recent milestone	Teams	Once per milestone	Professor PM Mentor DevOps Mentor Project Team	Project Manager	Presentation Meeting minutes

9. Test & Quality Assurance Plan

9.1. Introduction

The purpose of this Software Test Plan (STP) is to outline the scope, approach, and resources needed to support testing activities for the UMGC Capstone Project Proposal Management System (CaPPMS). This document will define the times, features, and expectations to ensure the system will meet the requirements and expectations.

9.2. Background

The UMGC Capstone Project Proposal Management System (CaPPMS) is a web-based application which allows customer, clients, and former students to submit detailed proposals of projects for consideration to be designed and implemented by UMGC SWEN 670 students as well as track the stages during the approval process.

9.3. Test Plan

CaPPMS requirements are verified by assigning them to individual test cases and executing a corresponding test procedure that demonstrates the requirement. The procedure consists of a hardcopy procedure that outlines the steps to be executed by the test engineer. The procedure includes descriptions of the test, prerequisites, necessary input data, expected output data, and a list of requirements that are being satisfied.

All copies of the test plan, test execution records, and test reports will be stored in the project's Microsoft Teams environment.

9.3.1. Schedule

The test plan and functional test cases will be written as part of milestone 2 delivery due on September 29, 2020. The test cases will be executed and updated as needed as part of the milestone 3 delivery due on October 20, 2020.

The final test execution will be executed on November 1, 2020. The final test plan, run for record results and the corresponding test report summarizing the results will be part of the milestone 4 delivery due on November 3, 2020.

Table 25 Test Schedule

Date	Task	Comment
9/24/2020	Test plan and functional test cases delivered to the customer for review and comments.	
9/29/2020	Test plan and functional test cases delivered for milestone 2.	Milestone 2 Delivery
10/10/2020	Smoke test execution.	
10/13/2020	Functional test execution.	
10/15/2020	Test plan updates and interim test report delivered to the customer for review and comments.	
10/20/2020	Test plan updates and interim test report delivered for milestone 3.	Milestone 3 Delivery
10/28/2020	Smoke test execution.	
11/1/2020	Final test execution and run for record. Approval from customer needed for successful testing.	
11/3/2020	Test plan updates and final test report delivered for milestone 4.	Milestone 4 Delivery

9.3.2. Responsibilities

To maintain separation of duties between software developer and team members performing testing, developer and test roles will be separated to ensure that a person other than the developer verifies the success of the code to the requirements. The CaPPMS team has identified two team members responsible for the following:

- Generating the test plans
- Providing inputs to the unit test designs (if needed)

- Writing the functional test cases
- Raising and validating all issues with the software
- Conducting the run for record tests

Due to the teams limited size, team members will perform multiple test functions. Two team members will execute the functional test cases and a third team member will perform quality assurance review.

The customer Dr. Brown will have the final decision on the Pass/Fail status for individual test cases and the overall success of the project.

Table 26 Roles and Responsibilities

Role	Description	Name
Test Conductor	Responsible for writing and executing the functional use cases. Completes and signs the Execution Report for every test cases.	Kathryn Stewart Yonas Mekete
Project Manager	Responsible for ensuring the correct persons are invited to the test event. Determines the Entry/ Exit/ Suspension/ Resumption of testing criteria has been met. Compiles the test report.	Kathryn Stewart
Quality Assurance	Responsible for witnessing and ensuring the functional test cases are executed successfully and confirming the Pass/Fail status with the customer. Signs the Execution Report for each test. Determines the Entry/ Exit/ Suspension/ Resumption of testing criteria has been met.	Tarun Lava

Role	Description	Name
Customer	Responsible for witnessing and confirming Pass/Fail status for each test and the overall test campaign. Signs the Execution Report for each test.	Dr. Michael Brown

9.3.3. Test Execution

Testing will be executed using the functional test cases documented in Section 3. As part of each execution a number of items will be recorded:

- Date
- Test conductor
- Software version
- Test environment
- Pass-fail status.

See Section 4 for the “as run” reports to be completed for each test. These “as run” files will be included in the final test report.

9.3.4. Scope

This document defines the processes that will be used for unit testing as part of the software development process and how the functional testing of the CaPPMS will be executed.

In-Scope

The testing required for the CaPPMS system will consist of the following:

- Unit testing
- Smoke testing
- Functional testing
- Regression testing

All of the functional requirements from the CaPPMS Software Requirement Specification and captured in the CaPPMS Requirements Matrix will be tested.

Out-Of-Scope

Out of scope testing will not be performed during the development of this project:

- Performance testing

9.3.5. Test Case Development

Test cases will be developed based on individual requirements. These requirements are placed into a requirements matrix. Each requirement will link to the test case where the requirement will be verified. All cases will be reviewed by the development team prior to execution. The test cases will also be reviewed and approved by the customer.

9.3.6. Test Approach

CaPPMS testing includes the following items to ensure that all test scenarios are covered and that the system meets all customer requirements.

- Unit Testing
- Smoke Testing
- Functional Testing
- Regression Testing

Unit testing

Automated testing written and executed by the software developers. These tests are written to identify issues and ensure the code under development is working as expected.

Smoke testing

A short test that consists of a subset of the functional test procedures. These tests will check major functionality to ensure the release is sufficient for complete testing. Smoke testing can begin when the code is successfully unit tested.

Functional testing

Complete testing to evaluate compliance with the functional requirements. Functional testing can begin when the code is successfully unit tested and the smoke test passes.

Regression testing

Specific tests to ensure no additional errors were introduced after the new code is committed or updated to support enhancements or bug fixes.

9.3.7. Test Environment

The CaPPMS system testing will always use the most recent version of code from the master branch stored in Github. The version number shall be noted in the test execution reports.

The following tools will be used to test the system:

- Junit5
- Selenium IDE 3.17.0 (Firefox / Chrome Extension)
- Microsoft Word, as part of Office 365

The CaPPMS system will be tested on the following hardware environments:

- Windows 10 Home OS
- Mac OS 10.15 Catalina

The CaPPMS system will be tested on the following web browsers:

- Firefox 81.0
- Chrome 85.0.4183.121

9.3.8. Test Execution

The test case execution will be managed by the Test Conductor. At the start of each test, the top of the Execution Report form in Section 9.5 will be completed. The paper procedure will be available for the Test Conductor to read the test steps, mark the step as passed or failed, and note any redlines or deviations from the written tests. The Test Conductor will execute the test step by step until the end.

All deviations will be recorded officially in the Execution Report and agreed to by all parties before continuing. These deviations are recorded in GitHub and assigned to a developer for corrections in accordance with the Project Plan.

Once the test is finished, the remainder of the Execution Report form in Section 9.5 be completed and saved for inclusion in the test report. The test report will be compiled by the project manager.

In the event that any retesting is necessary, the code and test cases must first be corrected. After the necessary changes are made the test will be repeated as if it were the initial run-through. The data from both test executions will be noted in the test report.

9.3.9. Pass/Fail Criteria

Test status is recorded at the end of the document on the test status form. The test will be considered as Pass if it satisfies all the requirements. Failures that are sufficiently demonstrated to be the result of the procedure, test environment, or operator error shall be considered 'Pass with Exception', provided there is an alternate means of verifying the requirement under test. Other failures result in a test failure that must be reviewed and corrected prior to delivery.

After the pass or fail criteria is agreed at test completion, the requirements matrix will be updated to reflect the test status.

9.3.10. Issue Tracking

All issues that may be found while the test is being conducted will be recorded at the time and any data that is necessary to recreate and resolving the issue will be collected. The issues will be recorded within the GitHub repository and assigned to a developer for correction per the Defect Management section of the CaPPMS Project Plan. Each issue will be assigned a priority and will be corrected beginning with the highest priority items.

Table 27 Defect Priority Definitions

Severity	Priority	Definition
Severity 3	Low	Not urgent or trivial. Minimal business impact.
Severity 2	Medium	Time-sensitive. Non-critical functionality loss or the software is still usable with a workaround.
Severity 1	High	Time-critical. Substantial loss of service or data loss.

9.3.11. Post Test Deliverables

At the end of all functional test campaign, interim testing for milestone 3, and the final testing for milestone 4, a test report will be populated which summarizes the details of the test.

An updated requirements matrix will also be included as part of this package. This report will contain the following items:

- Summary of the test event
- Scope
    - Test cases planned
    - Test cases not tested
- Metrics
    - Number of test cases planned
    - Number of test cases passed/failed
    - Number of defects, their status, and priority
- Test environment and tools
- Conclusion
- Appendix with the test scripts
- Appendix with copies of the test execution reports

9.3.12. Test Entry and Exit Criteria

The test can be started when the following criteria have been met:

- The test cases have been written, reviewed, and approved
- All required documentation and resources are available and both functional and non-functional requirements are understood by the test team
- The hardware and software are set up
- Smoke testing has been successfully completed on the master branch

The test can be exited when the following criteria have been met:

- All planned test cases have been executed
- Issues have been fully documented
- A plan is in place to address all High or Medium priority issues

9.3.13. Test Suspension and Resumption Criteria

The test may be suspended and resumed at a later time in the following conditions:

- A High priority issue occurs

- Requirement changes are requested
- Necessary resources are not available

The test can be resumed when the reason for the test suspension have been resolved.

9.3.14. Testing Risks

Testing risks are managed as outlined in the CaPPMS Project Plan. The risks outlined here are related to testing specifically.

A number of areas need to be evaluated for testing risks. These areas include schedule, technical, and operational risks. See the matrix of testing risks identified in the table below.

Table 28 Test Risk Evaluation

ID	Risk	Likelihood	Consequence	Risk Score
1	Project scope creep or changing requirements during development	High	High	High
2	Late completion of deliverables such as code or document approvals	Medium	High	High
3	Incorrect or insufficient resources	Low	Medium	Low
4	Insufficient time to train staff on test tools	Low	Medium	Low
5	Tight timeline for completion	Medium	Medium	Medium
6	Violation of specification or excessive issues identified	Medium	Medium	Medium

Table 29 Test Risk Mitigation Plans

ID	Risk	Mitigation Plan	Impact if Realized
1	Project scope creep or changing requirements during development	Engage stakeholders early and confirm the project scope. Ensure the schedule and resource plan is focused on the completion of the defined deliverables.	Additional scope will increase project costs. Schedule may increase in duration to implement changes.
2	Late completion of deliverables such as code or test plan approvals	Follow up at every team meeting to ensure that the code is being developed in accordance with the schedule. Ensure test documentation is provided with enough time for review and approvals.	The testing will be delayed or incomplete impacting cost, quality, and schedule.
5	Tight timeline for project completion	Follow up at every team meeting to ensure that the code is being developed in accordance with the schedule. Perform interim testing to identify bugs as the code is being developed.	The testing will be delayed and may be incomplete at the deadline.
6	Violation of specification or excessive issues identified	Perform interim testing to identify bugs as the code is being developed. Review and correct these issues prior to the official test.	The test event will fail and need to be repeated.

9.4. Functional Test Cases

The functional tests that will be written in selenium and executed in an automated fashion. The tests are summarized in the table below.

Table 30 Functional Test Summary

Test ID	Test Name	Part of Smoke Test?
Test-1.1	Navigate to the Homepage	Yes
Test-1.2	Navigate to the About page	Yes
Test-1.3	Navigate to the FAQ page	No
Test-2.1	Show Form Submit Process	No
Test-2.2	Show Form Error Messages	No
Test-2.3	Show Form Cancel Process	Yes
Test-2.4	Login	Yes
Test-2.5	Search and Filter	No
Test-2.6	Show Professor Form Error Messages	No
Test-2.7	Show Professor Cancel Button	No
Test-2.8	Edit Detail	No

9.4.1. Test-1.1 View Homepage

Test Objective: The user can view the homepage.

Preconditions: The user has access to the internet or other location hosting the website.

Table 31 Test-1.1View Homepage

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	Open browser	Browser opens		
2.	Type URL into the browser address bar and press Enter	The browser opens to the CaPPMS home page		REQ-1.1

9.4.2. Test-1.2 Submit Proposal Form

Test Objective: The user can submit a proposal for the SWEN 670 Capstone course.

Preconditions: The user has access to the internet or other location hosting the website.

Table 32 Test-1.2 Submit Proposal Form

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	Open browser	Browser opens		
2.	Type URL into the browser address bar and press Enter	The browser opens to the CaPPMS home page		
3.	Enter the following data into the form: • Name • Phone Number And press cancel	The data is cleared from the form		REQ-1.4
4.	Enter the following data into the form: • Name • Phone Number • Title of proposed project • Description of proposed project • Attach files • Link to user’s website • Non-Functional requirements And press submit	The user is alerted that the required field cannot be blank		REQ-1.5
5.	Enter in the email address into the form and press submit	The proposal is submitted and the user is notified of success		REQ-1.3

9.4.3. Test-1.3 Learn About Capstone Projects

Test Objective: The user can view data and frequently asked questions about the SWEN 670 Capstone program and desired projects.

Preconditions: The user has access to the internet or other location hosting the website. There are frequently asked questions in the database.

Table 33 Test-1.3 Learn About Capstone Projects

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	Open browser	Browser opens		
2.	Type URL into the browser address bar and press Enter	The browser opens to the CaPPMS home page and the introduction details are visible		REQ-1.2
3.	Press the FAQ hyperlink on the top of the page	The FAQ page opens and displays the current list of frequently asked questions		REQ-1.6

9.4.4. Test-2.1 Create New Account

Test Objective: The professor or other accounted users can add additional user accounts.

Preconditions: The professor or other accounted users must be able to access the application with a working internet connection and the professor must have an account in the system.

Table 34 Test-2.1 Create New Account

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	Open browser	Browser opens		
2.	Type URL into the browser address bar and press Enter	The browser opens to the CaPPMS home page and the introduction details are visible		
3.	The professor selects “Admin Login”.	The browser displays a login page.		REQ-1.8
4.	The professor enters log in details and presses submit.	The browser displays the landing page with a button for “Create New Account.”		
5.	The professor selects on the “Create New Account” button.	The browser displays a form to enter the details of a new account.		
6.	The professor enters the details of the new account and presses submit.	The browser displays a popup that says the account was successfully created.		REQ-1.9
7.	The professor enters the details of an account that already exists in the database and presses submit.	The browser displays a popup that says the account already exists in the database.		REQ-1.10

9.4.5. Test-2.2 Forgot Password

Test Objective: The professor or other accounted users can request a link to reset their password if they forgot.

Preconditions: The professor or other accounted users must be able to access the application with a working internet connection, must have an account in the system and access to the email address that's associated with that account.

Table 35 Test-2.2 Forgot Password

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	Open browser	Browser opens		
2.	Type URL into the browser address bar and press Enter	The browser opens to the CaPPMS home page and the introduction and about us details are visible		
3.	The professor selects "Admin Login".	The browser displays a login page with a "Forgot Password" link.		
4.	The professor selects the forget password link.	The browser displays a page that prompts the user to enter the email.		
5.	The professor enters the email associated with their account.	The system validates the email and sends an email to the registered email with a link to reset the password.		
6.	The professor navigates to the email and clicks on the link in the email.	The browser displays a page with two passwords fields to enter and submit the new password.		

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
7.	The professor enters in the new password details and submits.	The professor successfully logs in with the new password.		REQ-1.11

9.4.6. Test-2.3 View All Proposals

Test Objective: The professor can view all the submitted proposals on the

Preconditions: The professor must have an account and there must be proposals populated in the grid.

Table 36 Test-2.3 View All Proposals

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	Open browser	Browser opens		
2.	Type URL into the browser address bar and press Enter	The browser opens to the CaPPMS home page and the introduction and about us details are visible		
3.	The professor selects “Admin Login”.	The browser displays a login page.		
4.	The professor enters log in details and presses submit.	The browser displays the landing page with the top corner displaying the professor’s name		
5.	The professor selects the proposals tab.	The browsers display a grid with all the proposals with proposal name, status, and date		REQ-1.7

9.4.7. Test-2.4 View Proposal Details

Test Objective: The professor can view the details of a selected proposal.

Preconditions: The professor must be logged in and there must be proposals populated in the grid.

Table 37 Test-2.4 View Proposal Details

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	The professor navigates to the “View Proposals Page”	The browsers display a grid with all the proposals with proposal name, status, and date		
2.	The professor presses “View” on a proposal row.	The browser displays the “View Proposal Details” page with the details of the proposal populated and non-editable.		REQ-1.12 REQ-1.15

9.4.8. Test-2.5 Edit Proposal Details

Test Objective: The professor can edit the selected proposal.

Preconditions: The professor must be logged in and there must be proposals populated in the grid.

Table 38 Test-2.5 Edit Proposal Details

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	The professor navigates to the “View Proposals Page”	The browsers display a grid with all the proposals with proposal name, status, and date		

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
2.	The professor presses “Edit” on a proposal row.	The browser displays the “Edit Proposal Details” page with the details of the proposal populated and editable.		REQ-1.19
3.	The professor updates the proposal details fields and presses save.	The browser displays a notification that says the proposal was saved successfully.		
4.	The professor selects the Rejected status and does not enter anything in the “Reason for Rejection” field and presses save.	The browser displays a notification that the “Reason for Rejection” field is mandatory when the status is Rejected.		REQ-1.22
5.	The professor adds to the private comments field and presses save.	The browser displays a notification that says the proposal was saved successfully.		REQ-1.16
6.	The professor removes the sponsor’s name text and presses save.	The browser displays a notification that the field is mandatory.		REQ-1.17
7.	The professor makes edits to the field and presses cancel.	The browser displays a popup that says the edits were not saved and redirects the user to the “View All Proposals” page.		REQ-1.18

9.4.9. Test-2.6 Delete Proposal

Test Objective: The professor can delete a proposal from the table.

Preconditions: The professor must be logged in and there must be proposals populated in the grid.

Table 39 Test-2.6 Delete Proposal

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	The professor navigates to the “View Proposals Page”	The browsers display a grid with all the proposals with proposal name, status, and date		
2.	The professor presses “Delete” on a proposal row.	The browser displays a confirmation box for deleting the selecting proposal.		
3.	The professor selects “Yes.”	The browser displays a notification that says the proposal was deleted successfully and the grid refreshes to show the updated table.		REQ-1.20

9.4.10. Test-2.7 Export Proposal

Test Objective: The professor can export the selected proposal.

Preconditions: The professor must be logged in and there must be proposals populated in the grid. At least one proposal must have comments in the database for use with this test case.

Table 40 Test-2.7 Export Proposal

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	The professor navigates to the "View Proposals Page"	The browsers display a grid with all the proposals with proposal name, status, and date		
2.	The professor presses "View" on a proposal row.	The browser displays the "View Proposal Details" page with the details of the proposal populated and non-editable.		
3.	The professor presses the "Export Proposal" button on the bottom of the screen.	The browser downloads the exported file.		REQ-1.23
4.	The professor reviews the exported file.	There are no professor comments visible on the exported file		REQ-1.21

9.4.11. Test-2.8 Filter/Search Proposal

Test Objective: The professor can search or filter the proposals on the grid by status.

Preconditions: The professor must be logged in and there must be proposals populated in the grid.

Table 41 Test-2.8 Filter/Search Proposal

Step Number	Procedure Step	Success Criteria	Pass/Fail	Requirement
1.	The professor navigates to the “View Proposals Page”	The browser displays a grid with all the proposals with proposal name, status, and date		
2.	The professor selects a status on the grid.	The grid is updated based on the selection.		REQ-1.13 REQ-1.14

9.5. Backend Test Cases

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as expected. The CaPPMS was designed and developed with two main areas to unit testing, namely the front-end (developed in Angular) and the back-end testing consisting of a RESTApi available through the various controllers (Unit-tested individually).

Each backend unit was tested separately before integrating into front-end modules to test the interfaces between front-end and backend in the CaPPMS. The two key criteria for unit test to be complete are when the code coverage is 100% and all possible inputs are tested. Code coverage refers to covering test cases to ensure the entire internal code logic is tested. Data coverage refers to the testing of wide range of data that is accepted by the program, usually done by boundary testing where data ranging from accepted limits of boundary lowest to highest is passed to the program as inputs for testing.

The back-end tests have followed the format below, in which each Test Case has a title, description, status, priority, method being tested, also mention clearly the initial configuration, software configuration, steps to recreate and expected behavior. Please note that the FileUpload controller was created but not implemented within this version of the CaPPMS. The unit tests are included for completeness.

Table 42 Backend Test Summary

Test ID	Test Name
1.	Usr_Type Test Case 1
2.	Usr_Type Test Case 2
3.	Usr_Type Test Case 3
4.	Usr_Type Test Case 4
5.	Status Test Case 1
6.	Status Test Case 2
7.	Status Test Case 3
8.	Status Test Case 4
9.	FAQ Test Case 1
10.	FAQ Test Case 2
11.	FAQ Test Case 3
12.	FAQ Test Case 4
13.	Users Test Case 1
14.	Users Test Case 2
15.	Users Test Case 3
16.	Users Test Case 4
17.	Users Test Case 5
18.	Project Test Case 1
19.	Project Test Case 2
20.	Project Test Case 3
21.	Project Test Case 4
22.	Project Test Case 5
23.	FileUpload Test Case 1
24.	FileUpload Test Case 2
25.	FileUpload Test Case 3
26.	FileUpload Test Case 4
27.	FileUpload Test Case 5

9.6. Back end scripts

9.6.1. Usr_Type Test Case 1

Test case name: GetAllUserType

Method being tested:

@GetMapping("/usr_type")
List<Usr_Type> getAllUsr_Type()

Short Description: This method returns a list of Usr_Types

Input Data to constructor and/or method you are testing: None

Expected Results: Http 200 OK - List with all user types stored in the database.

9.6.2. Usr_Type Test Case 2

Test case name: CreateUserType

Method being tested:

@PostMapping("/usr_type")

public Usr_Type createUsr_Type(@RequestBody Usr_Type usr_type)
Short Description: This method creates a new User_Type and returns the same as the method return value.

Input Data to constructor and/or method you are testing: id (auto_increment), utype_descr

Expected Results: Http 200 OK + newly created user type

9.6.3. Usr_Type Test Case 3

Test case name: GetUserTypeById

Method being tested:
@GetMapping("/usr_type/{id}")

public ResponseEntity<Usr_Type> getUsr_TypeById(@PathVariable Long id)
Short Description: This method returns a Usr_Types Object matching the id

Input Data to constructor and/or method you are testing: id

Expected Results: HTTP 200 OK + usr_type object

9.6.4. Usr_Type Test Case 4

Test case name: UpdateUserType

Method being tested:

@PutMapping("/usr_type/{id}")

public ResponseEntity<Usr_Type> updateUsr_Type(@PathVariable Long id, @RequestBody Usr_Type usr_typeDetails)

Short Description: This method updates the Usr_Type referred to by “id”. Input data include utype_descr_

Expected Results: http code 200 OK. ResponseEntity<Usr_Type>

9.6.5. Status Test Case 1

Test case name: GetAllStatus

Method being tested:

@GetMapping("/status")

public List<Status> getAllStatus()

Short Description: This method returns a list of Status

Input Data to constructor and/or method you are testing: None

Expected Results: Http 200 OK - List with all Status stored in the database.

9.6.6. Status Test Case 2

Test case name: CreateStatus

Method being tested:

@PostMapping("/status")

public Status createStatus(@RequestBody Status status)

Short Description: This method creates a new Status and returns the same as the method return value.

Input Data to constructor and/or method you are testing: id (auto_increment), status_descr

Expected Results: Http 200 OK + newly created status

9.6.7. Status Test Case 3

Test case name: GetStatusById

Method being tested:

@GetMapping("/status/{id}")

public ResponseEntity<Status> getStatusById(@PathVariable Long id)

Short Description: This method returns a Status Object matching the id

Input Data to constructor and/or method you are testing: id

Expected Results: HTTP 200 OK + Status object

9.6.8. Status Test Case 4

Test case name: UpdateStatus

Method being tested:

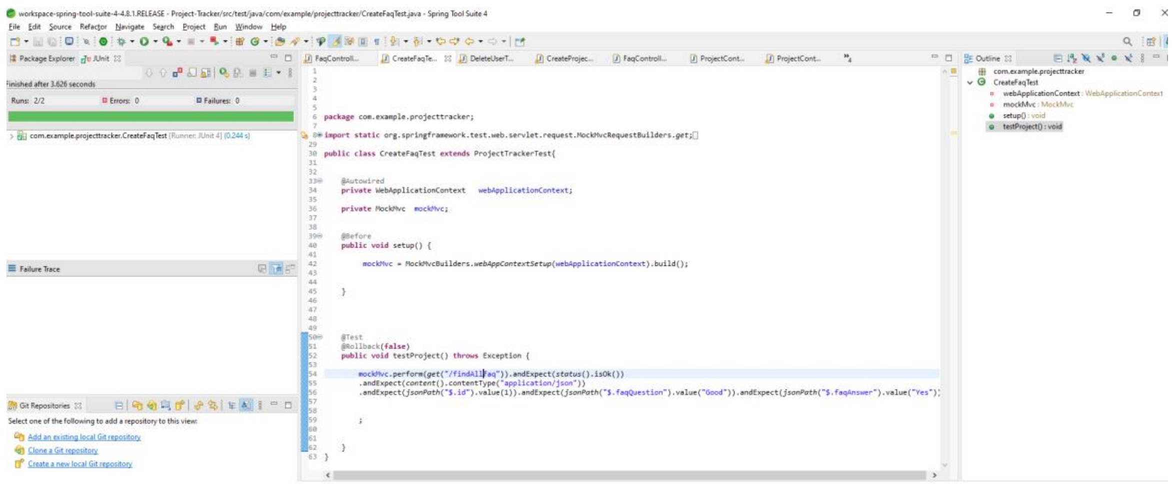
@PutMapping("/status/{id}")

public ResponseEntity<Status> updateStatus(@PathVariable Long id, @RequestBody Status statusDetails)

Short Description: This method updates the Status referred to by “id”. Input data include status_descr

Expected Results: http code 200 OK. ResponseEntity<Status>

9.6.9. FAQ Test Case 1



workspace-spring-tool-suite-4-4.8.1.RELEASE - Project-Tracker/src/test/java/com/example/projecttracker/CreateFaqTest.java - Spring Tool Suite 4

Finished after 3.626 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.example.projecttracker.CreateFaqTest [Runner: JUnit 4] (0.244 s)

```java
package com.example.projecttracker;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

public class CreateFaqTest extends ProjectTrackerTest{


    @Autowired
    private WebApplicationContext   webApplicationContext;

    private MockMvc  mockMvc;


    @Before
    public void setup() {

         mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();


    }




    @Test
    @Rollback(false)
    public void testProject() throws Exception {

        mockMvc.perform(get("/findAllFaq")).andExpect(status().isOk())
        .andExpect(content().contentType("application/json"))
        .andExpect(jsonPath("$.id").value(1)).andExpect(jsonPath("$.faqQuestion").value("Good")).andExpect(jsonPath("$.faqAnswer").value("Yes"));


        ;



    }
}
```

Figure 5 FAQ Test Case 1

Test case name: GetAllFAQ

Method being tested:

@GetMapping("/faq")

    public List<FAQ> getAllFAQ()

Short Description: This method returns a list of FAQ

Input Data to constructor and/or method you are testing: None

Expected Results: Http 200 OK - List with all FAQ stored in the database.

9.6.10. FAQ Test Case 2

Test case name: CreateFAQ

Method being tested:

@PostMapping("/faq")

    public FAQ createStatus(@RequestBody FAQ faq)

Short Description: This method creates a new FAQ and returns the same as the method return value.

Input Data to constructor and/or method you are testing: id (auto_increment), faq_question, faq_answer

Expected Results: Http 200 OK + newly created FAQ

9.6.11. FAQ Test Case 3

Test case name: GetFAQById

Method being tested:
@GetMapping("/faq/{id}")

public ResponseEntity<FAQ> getFAQById(@PathVariable Long id)
Short Description: This method returns a FAQ Object matching the id

Input Data to constructor and/or method you are testing: id

Expected Results: HTTP 200 OK + FAQ object

9.6.12. FAQ Test Case 4

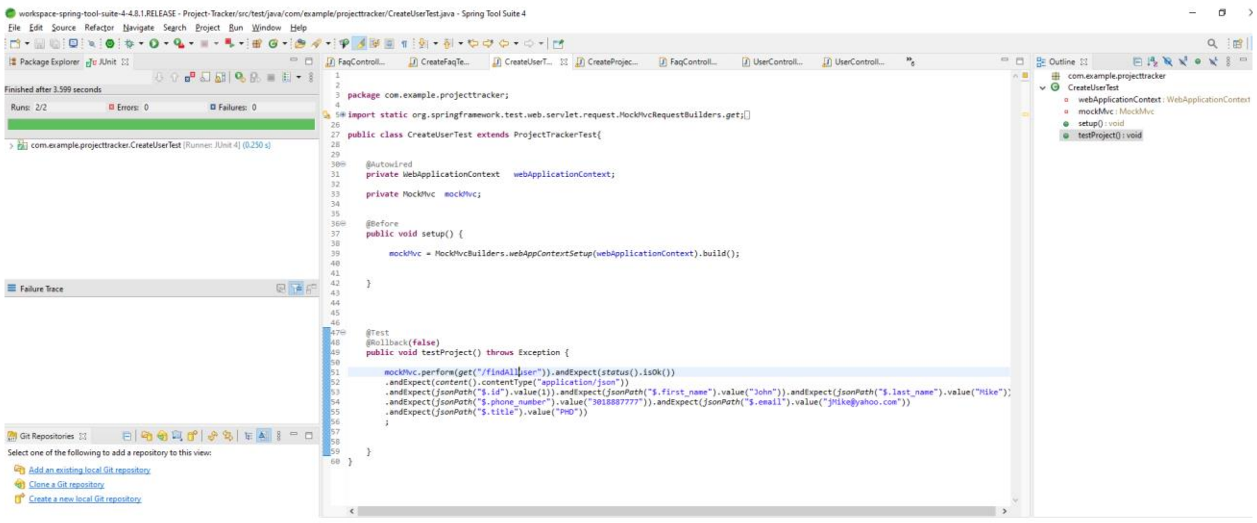
Test case name: UpdateFAQ

Method being tested:
@PutMapping("/faq/{id}")

public ResponseEntity<FAQ> updateStatus(@PathVariable Long id, @RequestBody FAQ faqDetails)
Short Description: This method updates the FAQ referred to by “id”. Input data include faq_question_faq_answer

Expected Results: http code 200 OK. ResponseEntity<FAQ>

9.6.13. Users Test Case 1



Finished after 3.599 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.example.projecttracker.CreateUserTest [Runner: JUnit 4] (0.250 s)

```java
package com.example.projecttracker;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

public class CreateUserTest extends ProjectTrackerTest{


    @Autowired
    private WebApplicationContext   webApplicationContext;

    private MockMvc  mockMvc;


    @Before
    public void setup() {

        mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();


    }




    @Test
    @Rollback(false)
    public void testProject() throws Exception {

        mockMvc.perform(get("/findAllUser")).andExpect(status().isOk())
        .andExpect(content().contentType("application/json"))
        .andExpect(jsonPath("$.id").value(1)).andExpect(jsonPath("$.first_name").value("John")).andExpect(jsonPath("$.last_name").value("Mike"));
        .andExpect(jsonPath("$.phone_number").value("3018887777")).andExpect(jsonPath("$.email").value("jMike@yahoo.com"))
        .andExpect(jsonPath("$.title").value("PHD"))
        ;


    }
}
```

Figure 6 User Test Case 1

Test case name: GetAllUsers

Method being tested:

@GetMapping("/user")

public List<User> getAllUser(){

Short Description: This method returns a list of all Users

Input Data to constructor and/or method you are testing: None

Expected Results: Http 200 OK - List with all user stored in the database.

9.6.14. Users Test Case 2

Test case name: CreateUser

Method being tested:

@PostMapping("/user")

public User createUser(@RequestBody User user)

Short Description: This method creates a new User and returns the same as the method return value.

Input Data to constructor and/or method you are testing: id (auto_increment), first_name, last_name, phone_number, email, title, usr_type, website, empl_id, account

Expected Results: Http 200 OK + newly created user

9.6.15. Users Test Case 3

Test case name: GetUserById

Method being tested:
@GetMapping("/user/{id}")

public ResponseEntity<User> getUserById(@PathVariable Long id)
Short Description: This method returns a User Object matching the id

Input Data to constructor and/or method you are testing: id

Expected Results: HTTP 200 OK + user object

9.6.16. Users Test Case 4

Test case name: UpdateUserById

Method being tested:
@PutMapping("/user/{id}")

public ResponseEntity<User> updateUser(@PathVariable Long id, @RequestBody User userDetails)
Short Description: This method updates the User referred to by "id". Input data include first_name, last_name, phone_number, email, title, usr_type, website, empl_id, account

Expected Results: http code 200 OK. ResponseEntity<User >

9.6.17. Users Test Case 5

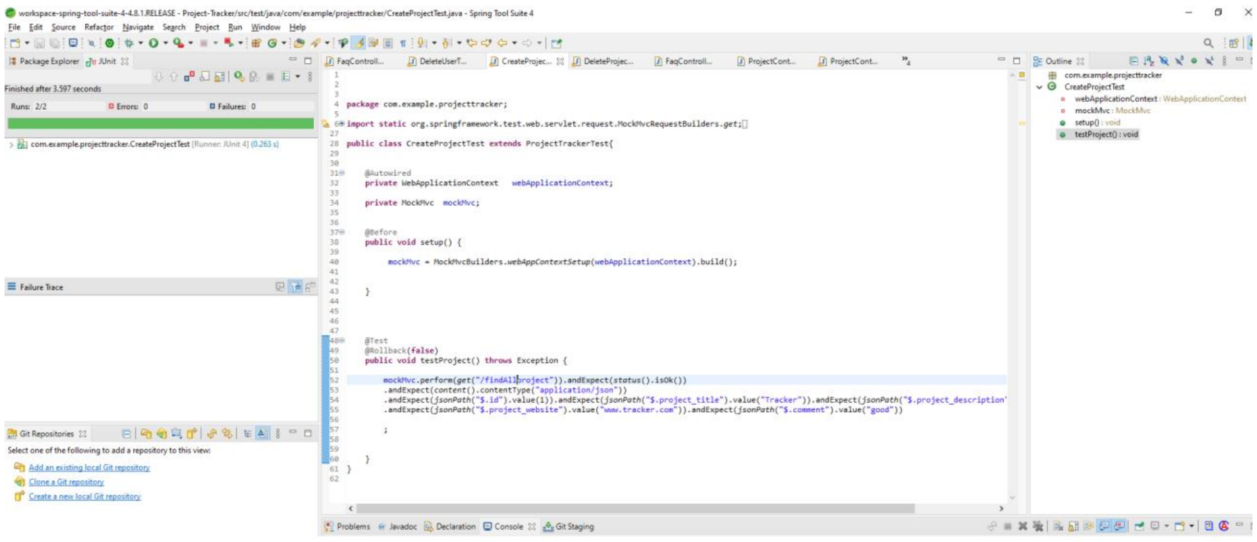
Test case name: DeleteUserById

Method being tested:
@DeleteMapping("/user/{id}")

public ResponseEntity <Map<String, Boolean>> deleteUser(@PathVariable Long id)
Short Description: This method deletes the User referred to by "id".

Expected Results: http code 200 OK. ResponseEntity<User >

9.6.18. Project Test Case 1



workspace-spring-tool-suite-4-4.8.1.RELEASE - Project-Tracker/src/test/java/com/example/projecttracker/CreateProjectTest.java - Spring Tool Suite 4

Finished after 3.597 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.example.projecttracker.CreateProjectTest [Runner: JUnit 4] (0.263 s)

```
package com.example.projecttracker;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

public class CreateProjectTest extends ProjectTrackerTest{


    @Autowired
    private WebApplicationContext   webApplicationContext;

    private MockMvc  mockMvc;


    @Before
    public void setup() {

         mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();


    }




    @Test
    @Rollback(false)
    public void testProject() throws Exception {

        mockMvc.perform(get("/findAllproject")).andExpect(status().isOk())
        .andExpect(content().contentType("application/json"))
        .andExpect(jsonPath("$.id").value(1)).andExpect(jsonPath("$.project_title").value("Tracker")).andExpect(jsonPath("$.project_description'
        .andExpect(jsonPath("$.project_website").value("www.tracker.com")).andExpect(jsonPath("$.comment").value("good"))

        ;


    }
}
```

Outline
com.example.projecttracker
CreateProjectTest
webApplicationContext : WebApplicationContext
mockMvc : MockMvc
setup() : void
testProject() : void
Failure Trace
Git Repositories
Select one of the following to add a repository to this view:
Add an existing local Git repository
Clone a Git repository
Create a new local Git repository
Problems
Javadoc
Declaration
Console
Git Staging

Figure 7 Project Test Case 1

Test case name: GetAllProject

Method being tested:

@GetMapping("/project")

public List<Project> getAllProject()

Short Description: This method returns a list of Projects

Input Data to constructor and/or method you are testing: None

Expected Results: Http 200 OK - List with all projects stored in the database.

9.6.19. Project Test Case 2

Test case name: CreateProject

Method being tested:

@PostMapping("/project")

public Project createProject(@RequestBody Project project)
Short Description: This method creates a new Project and returns the same as the method return value.

Input Data to constructor and/or method you are testing: id (auto_increment), project_description, project_title, project_website, project_status, users, attachments

Expected Results: Http 200 OK + newly created project

9.6.20. Project Test Case 3

Test case name: GetProjectById

Method being tested:
@GetMapping("/project/{id}")

public ResponseEntity<Project> getProjectById(@PathVariable Long id)
Short Description: This method returns a Project Object matching the id

Input Data to constructor and/or method you are testing: id

Expected Results: HTTP 200 OK + project object

9.6.21. 1.2.21 Project Test Case 4

Test case name: UpdateProject

Method being tested:

@PutMapping("/project/{id}")
public ResponseEntity<Project> updateProject(@PathVariable Long id, @RequestBody Project projectDetails)
Short Description: This method updates the Project referred to by “id”. Input data include project_description, project_title, project_website, project_status, users, attachments

Expected Results: http code 200 OK. ResponseEntity<Project>

9.6.22. Project Test Case 5

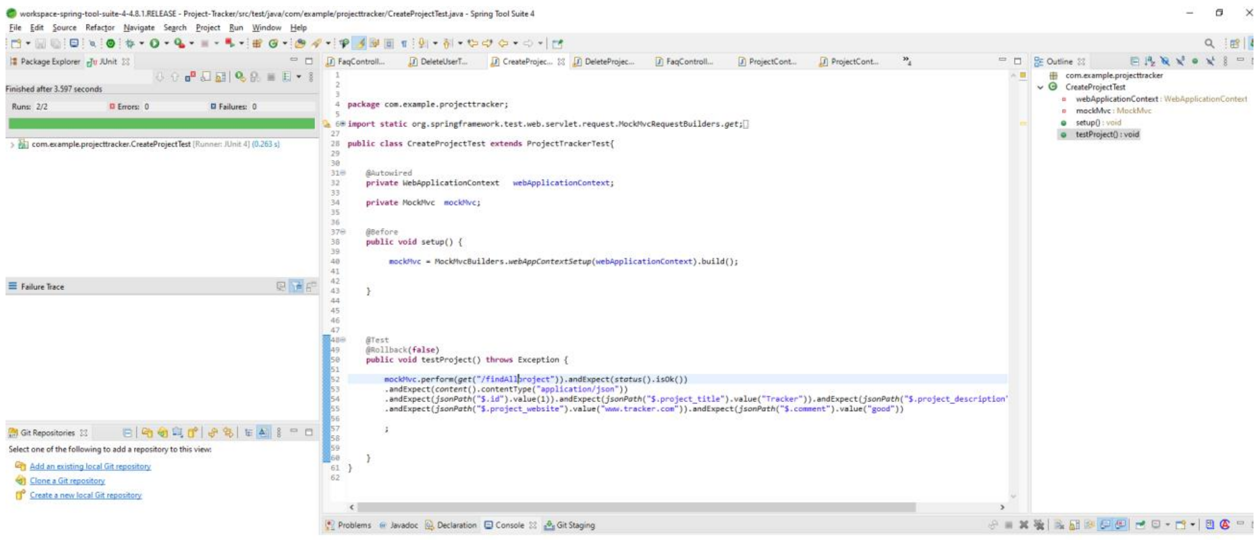
Test case name: DeleteProjectById

Method being tested:
@DeleteMapping("/project/{id}")

public ResponseEntity <Map<String, Boolean>> deleteProject(@PathVariable Long id)
Short Description: This method updates the Project referred to by “id”.

Expected Results: http code 200 OK. ResponseEntity<Project>

9.6.23. FileUpload Test Case 1


workspace-spring-tool-suite-4-4.8.1.RELEASE - Project-Tracker/src/test/java/com/example/projecttracker/CreateProjectTest.java - Spring Tool Suite 4
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
JUnit
FaqControll...
DeleteUserT...
CreateProjec...
DeleteProjec...
FaqControll...
ProjectCont...
ProjectCont...
Outline
Failure Trace
Git Repositories
Problems
Javadoc
Declaration
Console
Git Staging

```
Finished after 3.597 seconds
Runs: 2/2    Errors: 0    Failures: 0
com.example.projecttracker.CreateProjectTest [Runner: JUnit 4] (0.263 s)
```

```
package com.example.projecttracker;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
public class CreateProjectTest extends ProjectTrackerTest{


    @Autowired
    private WebApplicationContext   webApplicationContext;

    private MockMvc  mockMvc;


    @Before
    public void setup() {

        mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();


    }




    @Test
    @Rollback(false)
    public void testProject() throws Exception {

        mockMvc.perform(get("/findAllproject")).andExpect(status().isOk())
        .andExpect(content().contentType("application/json"))
        .andExpect(jsonPath("$.id").value(1)).andExpect(jsonPath("$.project_title").value("Tracker")).andExpect(jsonPath("$.project_description'
        .andExpect(jsonPath("$.project_website").value("www.tracker.com")).andExpect(jsonPath("$.comment").value("good"))

        ;


    }
}
```

```
com.example.projecttracker
CreateProjectTest
    webApplicationContext : WebApplicationContext
    mockMvc : MockMvc
    setup() : void
    testProject() : void
```

```
Select one of the following to add a repository to this view:
Add an existing local Git repository
Clone a Git repository
Create a new local Git repository
```

Figure 8 FileUpload Test Case 1

Test case name: GetAllProject

Method being tested:

@GetMapping("/project")

public List<Project> getAllProject()

Short Description: This method returns a list of Projects

Input Data to constructor and/or method you are testing: None

Expected Results: Http 200 OK - List with all projects stored in the database.

9.6.24. FileUpload Test Case 2

Test case name: CreateProject

Method being tested:

@PostMapping("/project")

public Project createProject(@RequestBody Project project)

Short Description: This method creates a new Project and returns the same as the method return value.

Input Data to constructor and/or method you are testing: id (auto_increment), project_description, project_title, project_website, project_status, users, attachments

Expected Results: Http 200 OK + newly created project

9.6.25. FileUpload Test Case 3

Test case name: GetProjectById

Method being tested:

@GetMapping("/project/{id}")

public ResponseEntity<Project> getProjectById(@PathVariable Long id)

Short Description: This method returns a Project Object matching the id

Input Data to constructor and/or method you are testing: id

Expected Results: HTTP 200 OK + project object

9.6.26. FileUpload Test Case 4

Test case name: UpdateProject

Method being tested:

public ResponseEntity<Project> updateProject(@PathVariable Long id, @RequestBody Project projectDetails)

Short Description: This method updates the Project referred to by "id". Input data include project_description, project_title, project_website, project_status, users, attachments

Expected Results: http code 200 OK. ResponseEntity<Project>

9.6.27. FileUpload Test Case 5

Test case name: DeleteProjectById

Method being tested:
@DeleteMapping("/project/{id}")

public ResponseEntity <Map<String, Boolean>> deleteProject(@PathVariable Long id)
Short Description: This method updates the Project referred to by “id”.

Expected Results: http code 200 OK. ResponseEntity<Project>

9.7. Test Execution Report

Table 43 Execution Report

Test Name:	
Test Start Date & Time:	
Test End Date & Time:	
Test Conductor:	
Software Versions:	
Test Environment:	
Quality Assurance:	
Other Witnesses:	
Deviations:	
Issues Opened:	
Action Items Opened:	
Status:	☐ ***Pass*** ☐ ***Pass w/exceptions*** ☐ ***Fail***
Test Conductor Signature/ Date:	
QA Signature/ Date:	
Customer Signature/ Date:	

10. References

Assadullah, M. (2020). SWEN 670 9040 Software Engineering Project (2208). Retrieved from https://learn.umgc.edu/d2l/le/content/515759/viewContent/18965508/View

Project Management Institute. (2013) A guide to the project management body of knowledge (PMBOK Guide) - Fifth edition. Newtown Square, PA: Author.