Project Plan

DineMaster Pro

Canim Restaurant

Date	**:**	**20.09.2024**
Version	**:**	**3.0**
State	**:**	**In Progress**
Author	**:**	**Samuil Kozarov**

Version history

Version	Date	Author(s)	Changes	State
1.0	03.09.2024	Samuil Kozarov	Project Plan created	Finished
2.0	11.09.2024	Samuil Kozarov	Project Plan revised after feedback session with Mr.Rabeling	Finished
3.0	18.09.2024	Samuil Kozarov	Project Plan revised after feedback session with Mr.Coenen	Finished

Distribution

Version	Date	Receivers
3.0	20.09.2024	Teachers

Contents

1. Project assignment

1.1 Context

Canim is a beloved local restaurant known for its exceptional cuisine and warm ambiance. Over the years, Canim has grown in popularity, attracting a steady stream of loyal customers. However, with its growing popularity, especially during peak dinner hours, the restaurant faces significant challenges. Currently, Canim only offers dine-in service, which has proven to be insufficient as demand continues to rise. What is more, this limits customer convenience, particularly for those who prefer to enjoy their meals at home or cannot visit the restaurant in person.

1.2 Goal of the project

The goal of this project is to help Canim restaurant face the rising interest of the customers for their place and enable customers to conveniently order and track their meals in order to enhance their dining experience and satisfaction. In order to achieve this, a software system will be implemented which will streamline order processing, offer real-time tracking, allow manager/s to easily manage the online menu and give the opportunity for the kitchen staff to easily visualize orders. A role-based access and advanced search and filter options will also be introduced, positioning Canim to capture a broader market, improve operational efficiency, and boost revenue through expanded service offerings.

1.3 Scope and preconditions

Inside scope:	**Outside scope:**
1 Software product	1 Training for staff
2 Database	2 Hardware
3 Demo presentation including: - Description of what features it has - How to use the staff interface	3 User manual
4 Tests	
5 Source Code	

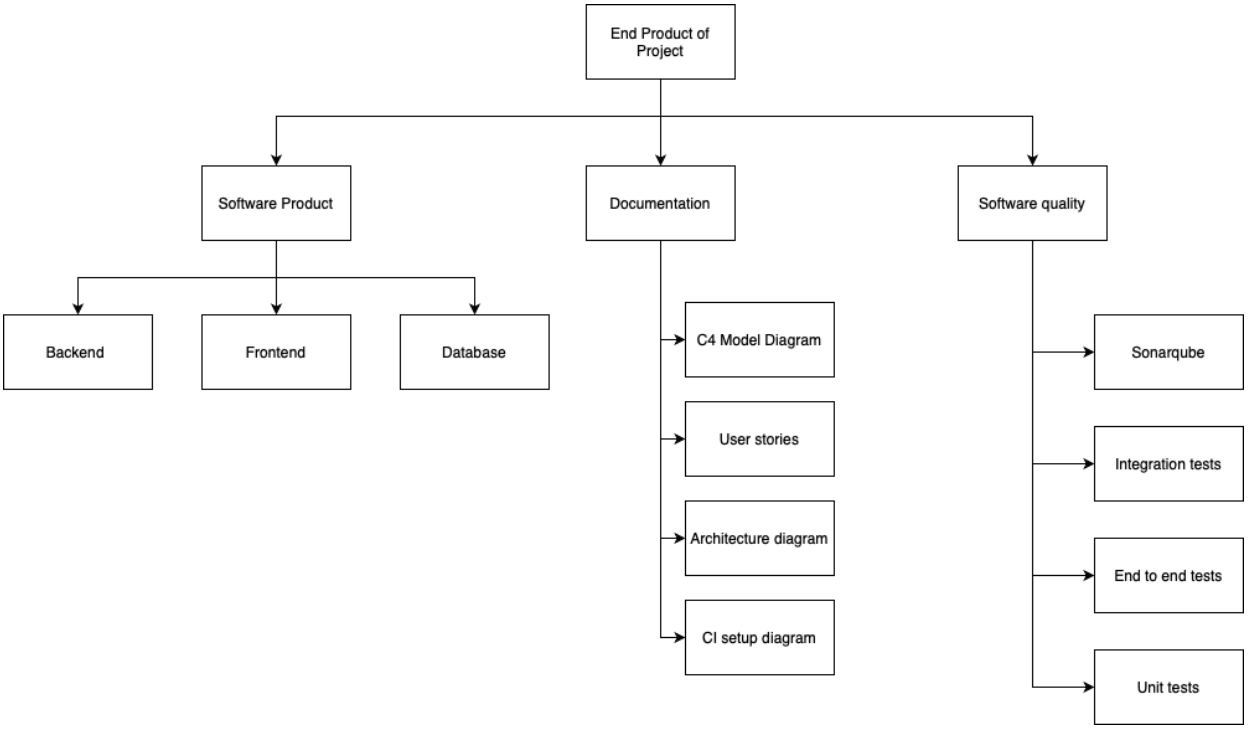
1.4 Strategy

This software will be developed using the Agile Scrum approach, which allows for changes and adjustments throughout the project lifecycle. By breaking the project into smaller pieces and delivering them incrementally, this way of working helps in identifying and mitigating risks early as well as allows for quick resolution of issues before they escalate. Scrum incorporates testing and quality assurance into each iteration.

*The daily scrum meetings will not be handled following the Scrum framework. Meetings between the developer and the teachers will be handled on a weekly basis in a suitable time slot.

*The reviews can be done by classmates or the developer.
*The sprint retrospective (technical) will be done by the teachers at the end of each sprint.

1.5 End products



Software Product: The fully functional system consisting of the backend, frontend and database which interact to deliver the core application features.

- **Backend**: The server-side logic and infrastructure responsible for processing requests, handling business logic, and managing data flow between the frontend and the database.
- **Frontend**: The user interface and client-side logic that enable users to interact with the application through a web browser or mobile interface.
- **Database**: The structured storage system where application data is persisted and retrieved, using MySQL in this case, managed via JPA/Hibernate ORM.

Documentation: Comprehensive material including diagrams, user stories, and CI setup instructions to guide the understanding and maintenance of the system.

- **C4 Model Diagram**: A high-level visualization of the software architecture, detailing the system's components, relationships, and deployment environment.
- **User Stories**: Descriptions of the functionality from the user's perspective, focusing on different use cases and how they interact with the system.
- **Architecture Diagram**: A technical diagram illustrating the structure and interactions of the system components, guiding both development and future enhancements.
- **CI Setup Diagram**: A visual representation of the Continuous Integration process, detailing how code is automatically built, tested, and deployed.

Software Quality: A set of practices, tools, and tests aimed at ensuring the robustness, maintainability, and performance of the software.

- **SonarQube**: A code quality tool used to identify bugs, vulnerabilities, and code smells in the codebase, ensuring adherence to best practices.
- **Integration Tests**: Tests that validate the interaction between different modules and components to ensure they work together as expected.
- **End to End Tests**: Full application tests that simulate real-world user interactions, verifying the system's functionality from start to finish.
- **Unit Tests**: Tests that validate the functionality of individual components or functions in isolation to ensure they behave correctly under various conditions.

2. Project organization

2.1 Stakeholders and team members

Name	Abbreviation	Role and functions	Availability
Samuil Kozarov *s.kozarov@student.fontys.nl* *+31 736862736*	*S.K.*	*Product developer*	*Availability: The whole duration of the project* *Days & Hours:* - *Monday (9:00 - 12:00)* - *Wednesday (9:00 - 12:00 | 13:00 - 16:00)* - *Friday (9:00 - 12:00)*
Michael Smith *m.smith@samauto.org* *+31 682 75 9594*	*M.S.*	*Business owner*	*Availability: Monday - Friday from 10:00a.m - 11:30p.m.*
Bart Rabeling *b.rabeling@fontys.nl* *+31885074484*	*B.R*	*Teacher / Tech Support*	*Availability: The whole duration of the project* *Days & Hours:* - *Monday (9:00 - 12:00)* - *Wednesday (9:00 - 12:00)*
Frank Coenen *f.coenen@fontys.nl* *+31885074348*	*F.C*	*Teacher / Tech Support*	*Availability: The whole duration of the project* *Days & Hours:* - *Wednesday (13:00 - 16:00)* - *Friday (9:00 - 12:00)*

2.2 Communication

The communication between the developer and the teachers can be done through two main sources - email and live meetings. The email communication can be established every day from 9:00 to 12:00 and from 13:00 to 16:00, through Microsoft Outlook or Microsoft Teams. Live meetings can be handled face to face only throughout the mentioned above (in the stakeholders table) available hours for each of the teachers. The cloud where all code changes will be uploaded can be found in the following link. Access will be granted to the developer and to the teachers as well.
GitLab repository: https://git.fhict.nl/I527531/dinemasterpro
The communication between the developer and the product purchaser will be established through Microsoft Outlook.

The development will take place as it follows:

- Monday from 9:00am to 12:00pm
- Wednesday from 9:00am to 12:00 pm and from 1:00 pm to 4:00 pm
- Friday from 9:00am to 12:00 pm

3. Activities and time plan

3.1 Phases of the project

The software development will last from 2nd of September until 17th of January and it will be splitted in 6 sprints each one 3 weeks long

1. Initiation (56 hours)

- **Problem Analysis** (4 hours)
- **Initial Planning** (4 hours)
- **User stories** (5 hours)
- **Initial Project Setup and GIT setup** (3 hours)
- **UML and PBS diagram** (5 hours)
- **Project Plan Creation** (5 hours)
- **Database SetUp** (10 hours)
- **Backend Setup** (15 hours)
    - Set up a RESTful API with at least 3 endpoints.
- **CI/CD Initialization** (5 hours)

2. Product development (160 hours)

- **Design Document** (15 hours)
- **Authentication and role based authorization system** (20 hours)
- **Role interface designs**
    - **Customer** (15 hours)
    - **Manager** (12 hours)
    - **Kitchen** (10 hours)
    - **Delivery** (10 hours)
- **Manager's side operations (CRUD) integration** (18 hours)
- **Client's side operations (Ordering, Payment, Order Status)** (20 hours)
- **Kitchen' side operation (Filtering, changing status)** (15 hours)
- **Delivery's side operation (Choosing orders and marking as "delivered)** (10 hours)
- **BackEnd to Database integration** (15 hours)

4. Testing and verifying code quality (55 hours)

- **SonarQube Setup** (4 hours)
- **Testing**
    - **SonarQube testing** (10 hours)
    - **Unit testing** (8 hours)
    - **End to end integration and testing** (12 hours)
- **Bugs fixing** (10 hours)
- **Code optimization and cleaning** (8 hours)
- **Security report** (5 hours)

5. Deployment and presenting(14 hours)

- **Final adjustments** (5 hours)

- **Final deployment** (8 hours)
- **Presentation** (1 hour)

Total Estimated Hours: 320 hours

Sprint 1 (50 hours): Initial setup, database, backend API, 3 endpoints, CI/CD, CRUD for Manager
Sprint 2 (55 hours): C4 Diagram, Dockerfile set up, Delivery role APIs, Customer Menu,Front end set up
Sprint 3 (55 hours): SonarQube set up, Ordering system, Connection and implementation to a real database
Sprint 4 (55 hours): Authentication and role based authorization, Order status implementation
Sprint 5 (55 hours): Testing (SonarQube, unit testing, integration testing), security report
Sprint 6 (50 hours): Final adjustments, final deployment, and presentation.

3.2 Time plan and milestones

Sprint	Phasing	Start date	Finish date
Sprint 1	1. Problem Analysis 2. Initial planning 3. User Stories 4. Initial Jira Backlog and GIT setup 5. Project Plan Creation 6. IntelliJ setup (working environment) 7. Backend Setup & Initial REST API (3 endpoints) 8. CI/CD Initialization 9. Fake (test) database implementation 10. Unit tests 11. Project architecture (Business, Controller, Persistence) 12. Implementation of main classes 13. Implementation of CRUD functionalities for Manager role 14. Implementation of CRUD functionalities for Items 15. Creating a fake (test) repository class with CRUD functionalities for the items	02.09.2024	20.09.2024
Sprint 2	1. C4 model diagram with context explanations 2. Implementation of Cross Origin Resource Sharing configuration 3. Frontend initiation 4. GitLab repository setup for front end 5. Implementation of Dockerfiles 6. Implementation of all Get methods for the Delivery people 7. Implementation of the filtering option for the kitchen staff 8. Implementation of the menu for the customers (design and basic functionalities)	21.09.2024	11.10.2024

	9. Unit tests		
Sprint 3	1. Implementation of a real database 2. Unit testing the business layer 3. Partial integration tests on the persistence layer 4. SonarQube installation 5. Conducting SonarQube tests 6. Implementation of the ordering system for the customers 7. Implementation of the front end for the manager to apply the CRUD methods	12.10.2024	08.11.2024
Sprint 4	1. Authentication 2. Role based authorization 3. UX feedback report 4. CI setup diagram with context explanations 5. Log in feature implementation 6. Storing in-memory, session storage or local storage 7. SonarQube tests 8. Unit test for more than 80% of the business layer 9. Implementation of the order status - for the customers and for the kitchen staff	09.11.2024	29.11.2024
Sprint 5	Testing and Verifying Code Quality 1. Security report for the OWASP top 10 security risks 2. Web socket feature for the app 3. UX to match the needs of the app 4. All tests of the Continuous Integration ot be passing 5. All SonarQube tests to be passing 6. Unit test for more than 80% of the business layer to be implemented and to pass 7. Code refactoring and cleaning	30.11.2024	20.12.2024
Sprint 6	Deployment & Presentation 1. Final individual track product with the MVP features implemented 2. Final revision 3. Code refactoring and cleaning 4. Project final adjustments 5. Project deployment 6. Final submission	21.12.2024	17.01.2025

4. Testing strategy and configuration management

4.1 Testing strategy

The testing strategies that will be used through the development process and will assure the quality of the final product are Integration tests, Unit tests, End to End tests and Sonarqube testing.

- **Unit testing:** Unit tests will focus on testing individual components or methods in isolation. This ensures that each function or class behaves as expected under various conditions.
  - **Tools**: JUnit for Java backend
  - **Goal**: Achieve at least **80% code coverage** for the business layer
  - **Automation**: These tests will be automated

- **Integration testing:** Tests that validate the interaction between different modules and components to ensure they work together as expected.
  - **Tools**: Spring Boot Test for backend services, Postman for API testing.
  - **Goal**: Ensure **seamless interaction** between the backend and database, as well as between frontend and backend.
  - **Automation**:

- **End to End testing:** Full application tests that simulate real-world user interactions, verifying the system's functionality from start to finish.
  - **Tools**: Cypress
  - **Goal**: The aim is to cover all user workflows
  - **Automation**: The majority of end-to-end tests will be automated to ensure that all critical user flows are continuously validated. They will run in the CI/CD pipeline to automatically verify that no breaking changes have been introduced when new code is pushed. [The developer keeps the right to implement manual testing if needed]

- **Sonarqube testing:** SonarQube will be used to analyze code quality by identifying potential bugs, security vulnerabilities, and code smells. It will also provide insights into code coverage.
  - **Tools**: SonarQube
  - **Goal**: Maintain a **good code quality** and address all critical security vulnerabilities and bugs.
  - **Automation**: SonarQube will be integrated into the CI/CD pipeline for continuous code quality assessment.

4.2 Test environment and required resources

A new test environment will be created in Docker. The product will be deployed there for testing.

Continuous integration environments will be created where pipeline tests will be run after each push to the repository. These tests will assure code execution with no error or other issues.

4.3 Configuration management

For the version control system, GitLab will be used:
GitLab repository: https://git.fhict.nl/I527531/dinemasterpro

Branching system:
For each major code implementation a new branch will be created and worked on. This will make the project tracking easier as well as stepping back in case of an issue.

- **Main** - the main branch where the application will be stored
- **Development branch** - a new branch of this type will be created every time a new major implementation is added to the project

5. Finances and risk

5.1 Risk and mitigation

Risk	Prevention activities	Mitigation activities
1 Laptop crash	Commit to git lab on a daily basis or after each change/update. Have a back up machine	Cool the device and leave it to rest for some time
2 Illness	Take care of your health and body condition	Take some medicines and rest a bit with no physical activities
3		