Software Engineering
< DLMCSPSE01 />

Project: HoppyBrew

Concept Phase

International University of Applyied Sciences

Written by
Asbjorn Bordoy

On the
18-Mar-2024

Revision 1.0

Abstract

The project aims to develop a comprehensive Beer Brewing Recipe Manager system, catering to brewing enthusiasts and homebrewers. This system facilitates the management of brewing processes and associated data through intuitive interfaces and robust functionalities. Users can create, share, and manage beer recipes, customize water and equipment profiles, schedule brewing sessions, monitor fermentation in real-time, generate reports, and more. The system ensures a seamless user experience by integrating with external devices like ISpindel for data collection and leveraging a database for secure storage and retrieval of brewing-related information. With an emphasis on user-friendly design and versatile features, the Beer Brewing Recipe Manager fosters innovation and tradition in the art of homebrewing.

Contents

List of Tables

List of Figures

1 Introduction and Goals

1.1 Conceptual Architecture Documentation for the HoppyBrew application.

This documentation: is intended to provide a high-level overview of the HoppyBrew application. The document is based on the template provided by (Hruschka and Starke, n.d.). Arc42 is a template for documenting software architectures. It is based on the ISO/IEC/IEEE 42010 standard, which is the international standard for documenting software architectures. The template is designed to be flexible and adaptable, and to be used in a wide range of software development projects. The template is divided into a number of sections, each of which covers a different aspect of the software architecture. The sections are designed to be used in a modular fashion, so that they can be used individually or in combination with other sections. The template is also designed to be easy to use, with a clear and consistent structure, and with a focus on the most important aspects of software architecture.

HoppyBrew: is a web application for managing brewing recipes and brew logs. The general idea of the application is to provide a user-friendly and intuitive interface for managing brewing recipes and brew logs. The application is designed to be compatible with a wide range of devices and browsers, and to integrate with other brewing tools and services, such as `iSpindel`. The application is targeted at beer brewer enthusiasts who want to manage their brewing recipes and brew logs in a simple and efficient way without the need for overpriced subscriptions fees like at ("Brewfather," n.d.), ("Brewers Friend," n.d.) or ("Beersmith," n.d.).

Note! The terminology `brew` and `batch` are used interchangeably in this document to refer to the same thing, i.e. a single brewing process.

1.2 Quality Goals

The top three quality goals for the architecture and design whose fulfillment is of highest importance to the major stakeholders of HoppyBrew have been identified as follows:

Table 1: Quality goals and priorities for the application.

Priority	Key word	Quality Goal
1	Usability	The application should be easy to use and intuitive, with a clean and modern user interface.
2	Compatibility	The application should be compatible with a wide range of devices and browsers. (mobile, desktop, tablet)
3	Integration	The application should integrate with other brewing tools and services, such as `iSpindel`.

These quality goals are based on derived summaries of `ISO/IEC 25010 quality model` provided in the lecture. The quality goals are derived from the most important stakeholders' expectations for the application. The quality goals are intended to provide a high-level overview of the most important quality attributes for the application, and to guide the architecture and design process in a way that ensures that these quality attributes are met.

The motivation behind these goals are to ensure that the application lives up to the expectations of the most important stakeholders, since they are the ones who will be the ones who influence the fundamental architecture and design decisions.

1.3 Stakeholders

In the architecture and design process of HoppyBrew, stakeholders play a pivotal role, providing essential requirements and constraints. Given that this project is part of a school assignment, the stakeholders are limited to the following individuals and their expectations:

Table 2: Stakeholders and their expectations for the application.

Priority	Name/Category	Expectations
Primary	Beer-brewer Enthusiast	Wants a user-friendly and intuitive application for managing brewing recipes and brew logs.
Secondary	Self-hosting Enthusiast and devellopers	Wants a high-quality, open-source application that is easy to maintain and extend.
Tertiary	The Software Architect	Wants a well-documented and well-structured application that meets the requirements of the assignment. On sucessful completion of the project, the Architect will be engolfed in a sense of pride and accomplishment.

It's important to recognize that since this is just a school project, the stakeholders are restricted to my own different personas.

1.4 Requirements Overview

The purpose of this section is to provide a high-level overview of the requirements for the application. The requirements are divided into two categories: functional requirements and non-functional requirements.

After having spiraled through how to structure the requirement section, I have come to see that there are two main ways to structure the requirements section. The first way is to structure the requirements section based on the use cases that have been identified for the application. The second way is to structure the requirements section based on the requirements provided by the stakeholders. The first way is more focused on the functionality of the application, while the second way is more focused on the expectations of the stakeholders.

So what what exactly is the difference between the two approaches one might ask?

> *"Functional requirements and use cases differ in several aspects, such as their level of detail, abstraction, and scope. Functional requirements tend to be more detailed, specific, and precise, while use cases tend to be more abstract, general, and flexible. Functional requirements focus on the system's functionality and behavior, while use cases focus on the user's goals and needs. Functional requirements cover the entire system and all its features, while use cases cover only a subset of the system and its functions."* ("Solution Architecture," n.d.)

In this document, I have chosen to reduce the concept of use cases to a single High-level use case diagram, and then focus on the functional requirements that are derived from the use cases. The reason for this is because the overall design of the application is relatively simple.

1.4.1 High-level Use Cases

The following use cases have been identified for the application:

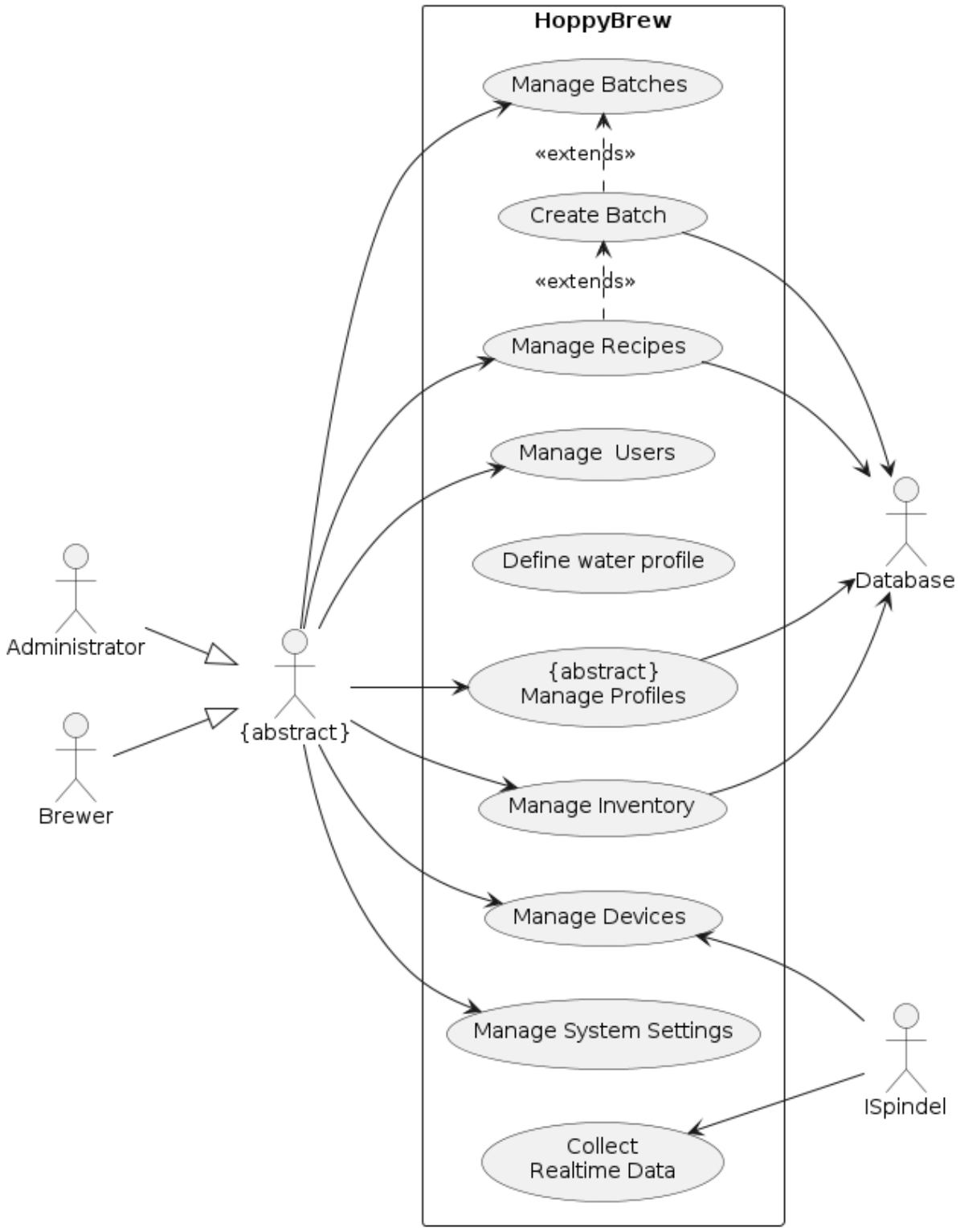


Figure 1: High-level use case diagram for the application.

1.4.1.1 Actors The following actors are involved in the use cases:

Table 3: Actors involved in the use cases for the application.

Actor	Description
Admin	The admin is responsible for managing the application and its users.
Brewer	The brewer is responsible for managing and creating brewing recipes and brews.
ISpindel	The iSpindel is responsible for providing data to the application.
Database	The database is responsible for storing and managing data for the application.

1.4.1.2 Use Cases

The following use cases are supported by the application:

Table 4: High-level use cases for the application.

Id	Use Case	Description
UC1	**Manage Users**	The admin can manage users, including creating, updating, and deleting users.
UC2	**Manage Recipes**	The brewer can manage recipes, including creating, updating, and deleting recipes.
UC3	**Create Batch**	The brewer can create a batch based on a recipe.
UC4	**Manage Batches**	The brewer can manage batches, including creating, updating, and deleting batches.
UC5	**Manage Profiles**	The brewer can manage profiles, including creating, updating, and deleting profiles.
UC6	**Manage Devices**	The admin can manage devices, including creating, updating, and deleting devices.
UC7	**Manage Inventory**	The admin can manage inventory, including creating, updating, and deleting inventory items.
UC8	**Manage System Settings**	The admin can manage system settings, including updating system settings.
UC9	**Collect Realtime Data**	The iSpindel can collect realtime data and send it to the application.

1.4.2 Functional Requirements

The functional requirements for the application are based on the use cases that have been identified for the application. The use cases are intended to provide a high-level overview of the functionality that the application should support. The use cases are based on the requirements provided by the stakeholders and are intended to guide the architecture and design process in a way that ensures that the application meets the expectations of the stakeholders.

It is important to note that the difference between a use case and a functional requirement is that a use case describes a specific interaction between a user and the system, while a functional requirement describes a specific function or feature that the system should support. And although the use cases are based on the requirements provided by the stakeholders, the functional requirements are based on the use cases. Which means that the functional requirements are taking a step further and are more detailed than the use cases.

The functional requirements for the application are as follows:

Table 5: Functional requirements for the application.

Id	Requirement	Description
FR1	**User Management**	The application should support CRUD operations for users.

Id	Requirement	Description
FR2	**Recipe Management**	The application should support CRUD operations for recipes.
FR3	**Batch Management**	The application should support CRUD operations for batches.
FR4	**{Profile Management}**	The application should support profile management
FR4.1	*Water Profile*	The application should support CRUD operations for water profiles.
FR4.2	*Mash Profile*	The application should support CRUD operations for mash profiles.
FR4.3	*Fermentation Profile*	The application should support CRUD operations for fermentation profiles.
FR4.4	*Equipment Profile*	The application should support CRUD operations for equipment profiles.
FR4.5	*Beer Style Profile*	The application should support CRUD operations for beer style profiles.
FR5	**Device Management**	The application should support CRUD operations for devices.
FR5.1	*iSpindel*	The application should support CRUD operations for iSpindel.
FR6	**Inventory Management**	The application should support CRUD operations for inventory items.
FR6.1	*Fermentables*	The application should support CRUD operations for fermentables.
FR6.2	*Hops*	The application should support CRUD operations for hops.
FR6.3	*Yeast*	The application should support CRUD operations for yeast.
FR6.4	*Miscellaneous*	The application should support CRUD operations for miscellaneous items.
FR7	**System Settings**	The application should support system settings.
FR8	**Recipe Library**	The application should be able to import and export recipes based on the BeerXML standard. ("BeerXML," n.d.)
FR9	**Realtime Data Collection**	The application should be able to collect realtime data from iSpindel.

Note!

- Requirements indicated with `{}` are abstract requirements that are further detailed in their respective sub-requirements.
- CRUD stands for Create, Read, Update, and Delete.

1.4.3 Non-functional Requirements

The non-functional requirements for the application are based on the quality goals that have been identified for the application. The quality goals are intended to provide a high-level overview of the most important quality attributes for the application, and to guide the architecture and design process in a way that ensures that these quality attributes are met. Unlike the functional requirements, the non-functional requirements not always so easily measurable, and are often more subjective in nature.

The non-functional requirements for the application are as follows:

Table 6: Non-functional requirements for the application.

Id	Requirement	Description
NFR1	**Usability**	The application should be easy to use and intuitive, with a clean and modern user interface.
NFR2	**Compatibility**	The application should be compatible with a wide range of devices and browsers.
NFR3	**Integration**	The application should integrate with other brewing tools and services, such as iSpindel.
NFR4	**Performance**	The application should be fast and responsive, with minimal latency.
NFR5	**Scalability**	The application should be able to handle a large number of users and data.
NFR6	**Reliability**	The application should be reliable and available, with minimal downtime.
NFR7	**Security**	The application should be secure, with user authentication and authorization.
NFR8	**Maintainability**	The application should be easy to maintain and extend, with clean and modular code.
NFR9	**Documentation**	The application should be well-documented, with clear and concise documentation.
NFR10	**Open-source**	The application should be open-source, with a permissive license.

Note!

- The non-functional requirements are based on the quality goals that have been identified for the application.
- The non-functional requirements are intended to provide a high-level overview of the most important quality attributes for the application.
- The non-functional requirements are intended to guide the architecture and design process in a way that ensures that these quality attributes are met.

1.5 Architecture Constraints

1.5.1 Assumptions

When starting a new project, we often rely on certain assumptions based on the information we have at the time. Here are the assumptions we've made for this application:

1. **Online Learning:** We assume that any missing knowledge can be gained through online resources and documentation. This means we believe we can learn what we need to know about the technologies used in the project without too much difficulty, and in a reasonable amount of time.### Constraints
2. **Deadline:** Since this is just a school project, we've got a deadline we need to meet. We've only got a certain amount of time to finish everything, including planning how the project will work. This deadline is set based on how much credit the project is worth and how much work we expect it to take. For example, since this project is worth 5 credits, we think it'll take about 150 hours to complete.
3. **Feature Limitations:** Because it's a school project, we're going to leave out some parts of the application that aren't super important. For instance, we might not implement every single type of action for all the different parts of the app. If we can do one type of action for one part, we'll assume we can do it for all the others.
4. **User Interface Simplification:** To keep things simple, we'll make the user interface as basic as possible while still showing how the app works. This means it won't look super fancy or work perfectly on every device, but it'll get the job done. We won't spend too much time on making it look pretty because that's not the main focus of the project.

1.5.2 Dependencies

The application is dependent on the following external services and tools:

1. **iSpindel:** The application is dependent on iSpindel for collecting realtime data.
2. **Database:** The application is dependent on a database for storing and managing data.
3. **GitHub:** The application is dependent on GitHub for version control and collaboration.
4. **Docker:** The application is dependent on Docker for containerization and deployment.

1.5.3 Risks

There are a number of risks associated with this project, which could potentially impact the success of the project. The following risks have been identified for the application:

1. **Technical Risks:** There is a risk that the technologies used in the project are not suitable for the requirements of the application.
2. **Time Risks:** There is a risk that the project will take longer than expected to complete, due to unforeseen circumstances.
3. **Skill development Risks:** There is a risk that the required knowledge for the project cannot be acquired in a reasonable amount of time.# System Scope and Context

1.6 Business Context

As indicated in the business context diagram below, the system only interacts with three external actors, namely the Administrator, The Brewer, and the ISpindel. The Administrator is responsible for managing the system, including adding new users, managing user roles, and monitoring the system. The Brewer is responsible for creating new brews, managing existing brews, and monitoring the progress of the brews. The ISpindel is responsible for collecting real-time data from the brewing process and sending it to the system.

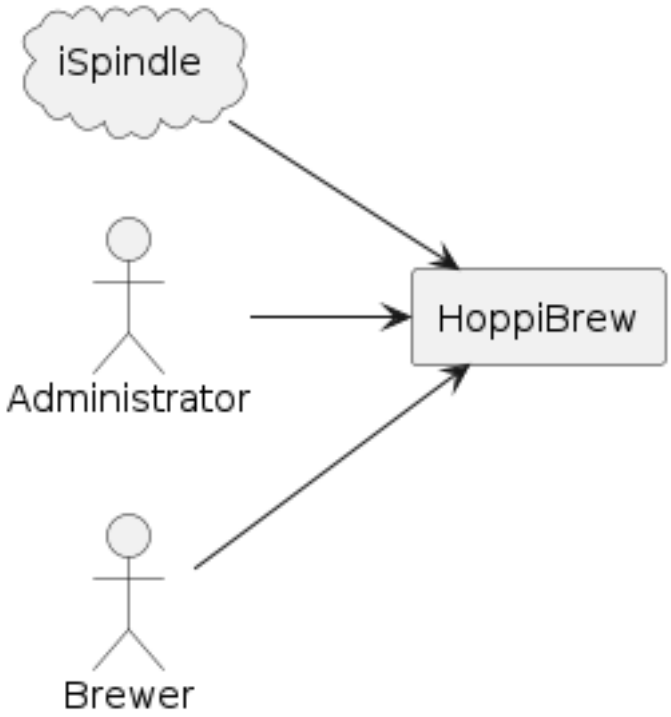
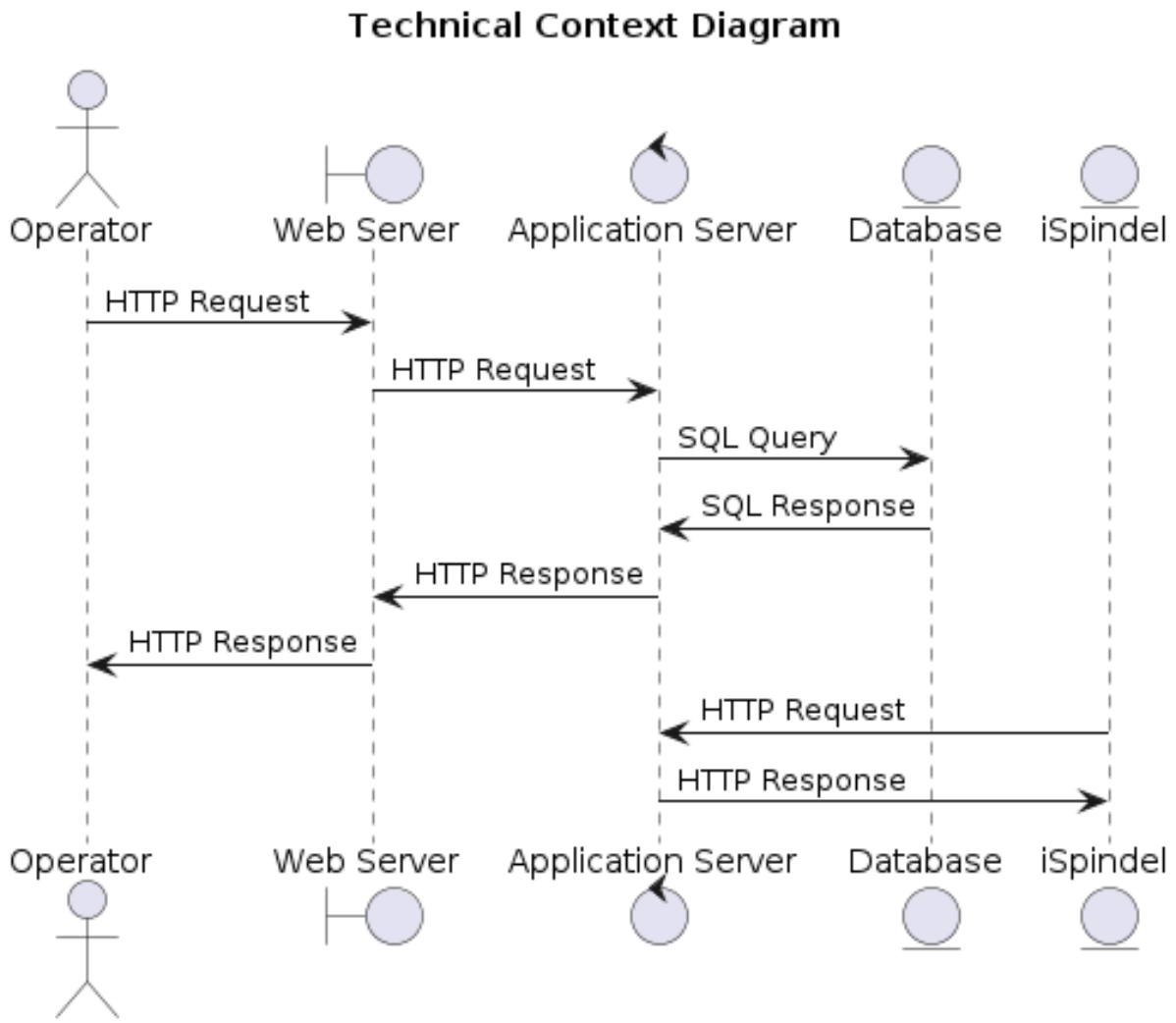


Figure 2: Business-Context-Vew

1.7 Technical Context

From a technical perspective, the system interacts with several external systems and services. The system is dependent on the iSpindel for collecting real-time data from the brewing process. The system is also

dependent on a database for storing and managing data. The system uses GitHub for version control and collaboration. Finally, the system uses Docker for containerization and deployment.



Solution Strategy

1.8 Technology Decisions

The application will be encapsulated in a containerized environment using Docker. This will allow for easy deployment and scaling of the application, as well as ensuring consistency across different environments. The application will be developed using a microservices architecture, with each component of the application running as a separate service. This will allow for greater flexibility and scalability, as well as making it easier to maintain and update the application.

1.8.1 Frontend Technology

After evaluating the requirements and constraints of the project, we have decided to use Vue.js as the frontend technology for the application. This is because Vue.js is a lightweight and flexible JavaScript framework that allows for rapid development of single-page applications. It provides a simple and intuitive syntax, making it easy to learn and use, which is ideal for a school project with a tight deadline. Additionally, Vue.js has a large and active community, which means there are plenty of resources and support available for developers. This will help us overcome any challenges we may face during the development process.

1.8.2 Backend Technology

- Python
- RESTful API
- (“FastAPI,” n.d.)(https://fastapi.tiangolo.com/)
- PostgreSQL
- pydantic

1.8.3 Database Technology

1.8.4 Deployment Technology

1.8.5 Development Tools

1.9 Top-level Decomposition

1.10 Key Quality Goals

1.11 Organizational Decisions

1.12 Motivation

1.13 Solution Approaches

1.14 Additional Considerations

2 Building Block View

2.1 Whitebox Overall System

<Overview Diagram>

Motivation
<text explanation>

Contained Building Blocks
<Description of contained building block (black boxes)>

Important Interfaces
<Description of important interfaces>

2.1.1 <Name black box 1>

<Purpose/Responsibility>

<Interface(s)>

<(Optional) Quality/Performance Characteristics>

<(Optional) Directory/File Location>

<(Optional) Fulfilled Requirements>

<(optional) Open Issues/Problems/Risks>

2.1.2 <Name black box 2>

<black box template>

2.1.3 <Name black box n>

<black box template>

2.1.4 <Name interface 1>

…

2.1.5 <Name interface m>

2.2 Level 2

2.2.1 White Box *<building block 1>*

<white box template>

2.2.2 White Box *<building block 2>*

<white box template>

...

2.2.3 White Box *<building block m>*

<white box template>

2.3 Level 3

2.3.1 White Box <_building block x.1_>

<white box template>

2.3.2 White Box <_building block x.2_>

<white box template>

2.3.3 White Box <_building block y.1_>

<white box template> # Runtime View

2.4 <Runtime Scenario 1>

- *<insert runtime diagram or textual description of the scenario>*
- *<insert description of the notable aspects of the interactions between the building block instances depicted in this diagram.>*

2.5 <Runtime Scenario 2>

2.6 ...

2.7 <Runtime Scenario n>

3 Deployment View

3.1 Infrastructure Level 1

<Overview Diagram>

Motivation
<explanation in text form>

Quality and/or Performance Features
<explanation in text form>

Mapping of Building Blocks to Infrastructure
<description of the mapping>

3.2 Infrastructure Level 2

3.2.1 *<Infrastructure Element 1>*

<diagram + explanation>

3.2.2 *<Infrastructure Element 2>*

<diagram + explanation>

...

3.2.3 *<Infrastructure Element n>*

<diagram + explanation> # Cross-cutting Concepts

3.3 *<Concept 1>*

<explanation>

3.4 *<Concept 2>*

<explanation>

...

3.5 *<Concept n>*

<explanation> # Architecture Decisions # Quality Requirements

3.6 Quality Tree

3.7 Quality Scenarios

4 Risks and Technical Debts

5 Glossary

Term	Definition
Actor	in use case parlance, are parties outside the system that interact with the system. They may be users or other systems. Each actor defines a coherent set of roles users of the system can play (UML, 1999). Cockburn (1997) distinguishes between primary and secondary actors. A primary actor is one having a goal requiring the assistance of the system. A secondary actor is one from which the system needs assistance to satisfy its goal.
Architecture	The term software architecture is used both to refer to the high-level structure of software systems and the specialist discipline or field distinct from that of software engineering. The architecture of a software system identifies a set of components that collaborate to achieve the system goals. The architecture specifies the "externally visible" properties of the components-i.e., those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on (Bass et al., 1998). It also specifies the relationships among the components and how they interact.
Conceptual Architecture	The intent of the conceptual architecture is to direct attention at an appropriate decomposition of the system without delving into the details of interface specification and type information. Moreover, it provides a useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and many users. The conceptual architecture identifies the system components, the responsibilities of each component, and interconnections between components. The structural choices are driven by the system qualities, and the rationale section articulates and documents this connection between the architectural requirements and the structures (components and connectors or communication/co-ordination mechanisms) of the architecture.
Features	Features are the differentiating functionality of a product. This functionality may not be available in other products, or it may not be available with the same quality characteristics.
Functional Requirements	Functional requirements capture the intended behavior of the system-or what the system will do. This behavior may be expressed as services, tasks or functions the system is required to perform.
Logical Architecture	The logical architecture is the detailed architecture specification, precisely defining the component interfaces and connection mechanisms and protocols. It is used by the component designers and developers.
Meta-architecture	The meta-architecture is a set of high-level decisions that will strongly influence the structure of the system, but is not itself the structure of the system. The meta-architecture, through style, patterns of composition or interaction, principles, and philosophy, rules certain structural choices out, and guides selection decisions and trade-offs among others. By choosing communication or co-ordination mechanisms that are repeatedly applied across the architecture, a consistent approach is ensured and this simplifies the architecture. (See Bredemeyer Consulting and Bredemeyer Consulting.)
Non-functional Requirements	Non-functional requirements or system qualities, capture required properties of the system, such as performance, security, maintainability, etc.-in other words, how well some behavioral or structural aspect of the system should be accomplished.
Product Line	Product lines consist of basically similar products with different cost/feature variations per product.
Product Family	Product families include a number of product lines targeted at somewhat different markets or usage situations. What makes the product lines part of a family, are some common elements of functionality and identity.

Term	Definition
Qualities	System qualities, or non-functional requirements, capture required properties of the system, such as performance, security, maintainability, etc.-in other words, how well some behavioral or structural aspect of the system should be accomplished.
Scenario	A scenario is an instance of a use case, and represents a single path through the use case. Thus, one may construct a scenario for the main flow through the use case, and other scenarios for each possible variation of flow through the use case (e.g., triggered by error conditions, security breaches, etc.). Scenarios may be depicted using sequence diagrams.
Use Case	A use case defines a goal-oriented set of interactions between external actors and the system under consideration. That is, use cases capture who (actors) does what (interactions) with the system, for what purpose (goal). A complete set of use cases specifies all the different ways to use the system, and thus defines all behavior required of the system–without dealing with the internal structure of the system.
Use Case Diagram	A use case diagram is a graphical representation of the use cases and their relationships to the actors.
User Interface	The user interface is the part of the system with which the users interact. It includes all the screens, forms, reports, and so on that the users use to interact with the system.
User Story	A user story is a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.
View	A view is a representation of a whole system from the perspective of a related set of concerns. Views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers, system engineers, and project managers. Views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers, system engineers, and project managers.
Viewpoint	A viewpoint is a specification of the conventions for constructing and using a view. A viewpoint specifies not only the kinds of models that are to be constructed, but also the rules governing the construction of those models.
Viewtype	A viewtype is a template for a view. It specifies the types of models that are to be constructed, and the rules governing the construction of those models.
Work Product	A work product is a document or model that is produced as part of a software development process. Work products are used to capture and communicate information about the system being developed.

Bibliography

"Beersmith." n.d. Beersmith. https://beersmith.com/.

"BeerXML." n.d. BeerXML. http://www.beerxml.com/.

"Brewers Friend." n.d. Brewer's Friend. https://www.brewersfriend.com/.

"Brewfather." n.d. Brewfather. https://web.brewfather.app/.

"FastAPI." n.d. FastAPI. https://fastapi.tiangolo.com/.

Hruschka, Dr. Peter, and Dr. Gernot Starke. n.d. "Arc42." arc42. https://arc42.org/.

"Solution Architecture." n.d. LinkedIn. https://www.linkedin.com/advice/0/how-do-you-leverage-functional-requirements.