<!DOCTYPE html>

Software Engineering | < DLMCSPSE01 /> | Project: HoppyBrew

Software Engineering

< DLMCSPSE01 />

Project: HoppyBrew

Concept Phase

Written by Asbjorn Bordoy

Date: 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.

Introduction and Goals

This document outlines the architecture and design of HoppyBrew, a web-based application for managing brewing recipes and brews. The application is designed to be user-friendly and intuitive, with a clean and modern user interface. The application is also designed to be compatible with a wide range of devices and browsers, and to integrate with other brewing tools and services, such as `iSpindel`.

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

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:

Priority	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`.

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.

Stakeholders

The following table lists all the stakeholders of HoppyBrew, along with their roles, contact information, and expectations. It is important to note that these stakeholders are the primary sources of requirements and constraints for the architecture and design of HoppyBrew.

Role/Name	Contact Information	expectations
Primary Stakeholder	Brewing Enthusiast	Wants a user-friendly and intuitive application for managing brewing recipes and brew logs.
Secondary Stakeholder	Developer/Contributor	Wants a high-quality, open-source application that is easy to maintain and extend.

Requirements Overview

HoppyBrew is driven by the following essential features and functional requirements:

“‘{r definePlantuml, include=FALSE} library(plantuml) x <- ’ left to right direction

actor Administrator as Admin actor Brewer as Brewer actor Database as DB actor ISpindel as ISpindel actor “{abstract}” as AbstractUser

’ x <- plantuml(x)

```
## Plotting to a file
To save the graph in a file, we simply specify the `file` argument in the plot command:
```

```{r exampleFile, include=FALSE}
plot(
  x,
  file = "./documents/01-Conception-Phase/png/testt.png"
)
```

And here is the file

Id	Requirement	Explanation
F1	**Manage and create brewing recipes**	The application should allow users to manage and create brewing recipes.
F2	**Manage and create brews and log their progress**	The application should allow users to manage and create brews and log their progress.

Plotting Plantuml graphics

Define plantuml code

First, we define a plantuml object based on some plantuml code “‘{r definePlantuml, include=FALSE} library(plantuml) x <- ’ (*) –> “Initialization”

if “Some Test” then –>[true] “Some Activity” –> “Another activity” -right-> (*) *else ->[false] “Something else” –>[Ending process]* (*) endif ’ x <- plantuml(x)

```
## Plotting to a file
To save the graph in a file, we simply specify the `file` argument in the plot command:
```{r exampleFile, include=FALSE}
plot(
  x,
  file = "./documents/01-Conception-Phase/png/test.png"
)
```

And here is the file
```

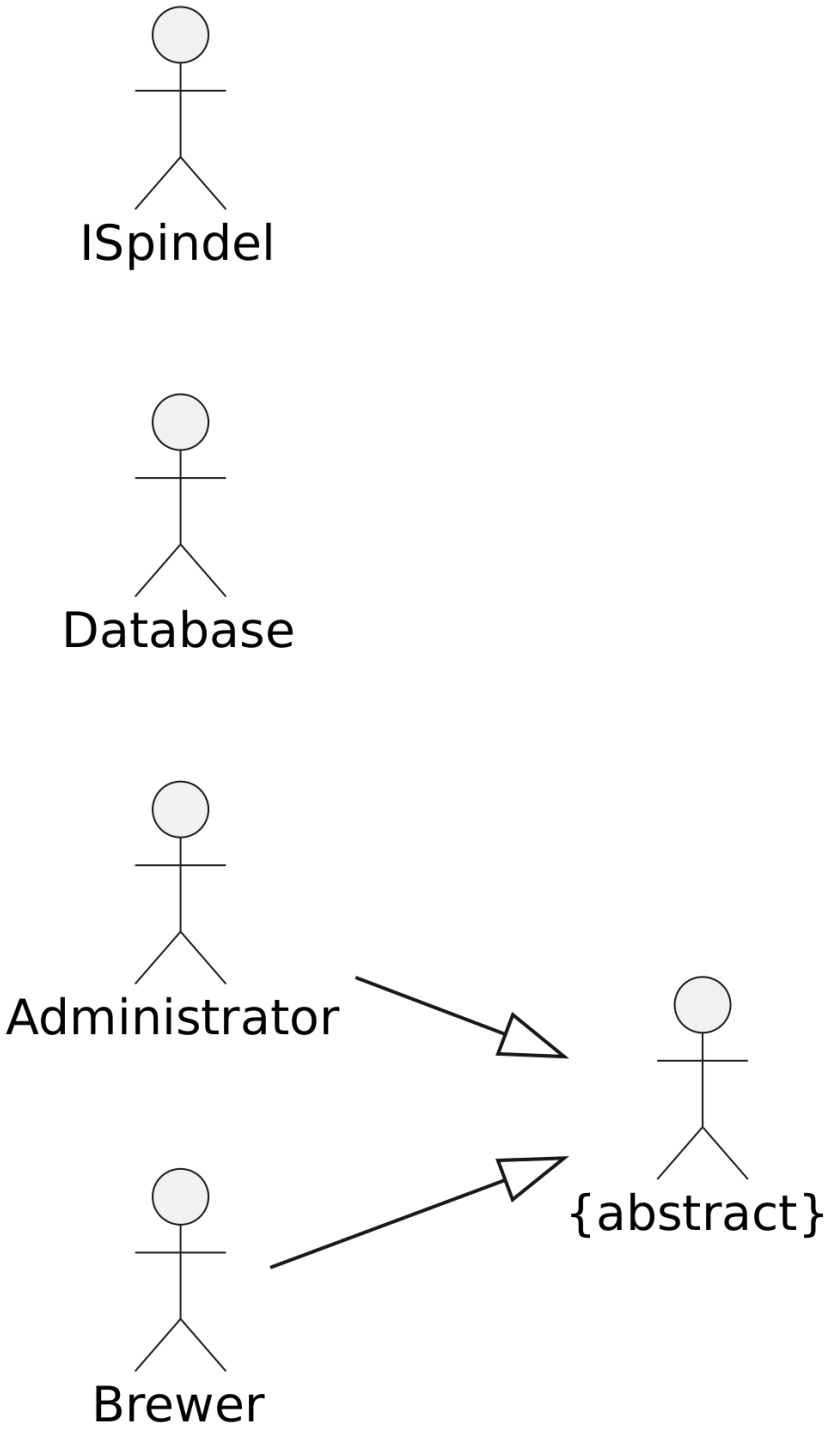


Figure 1: vignettes/test.png

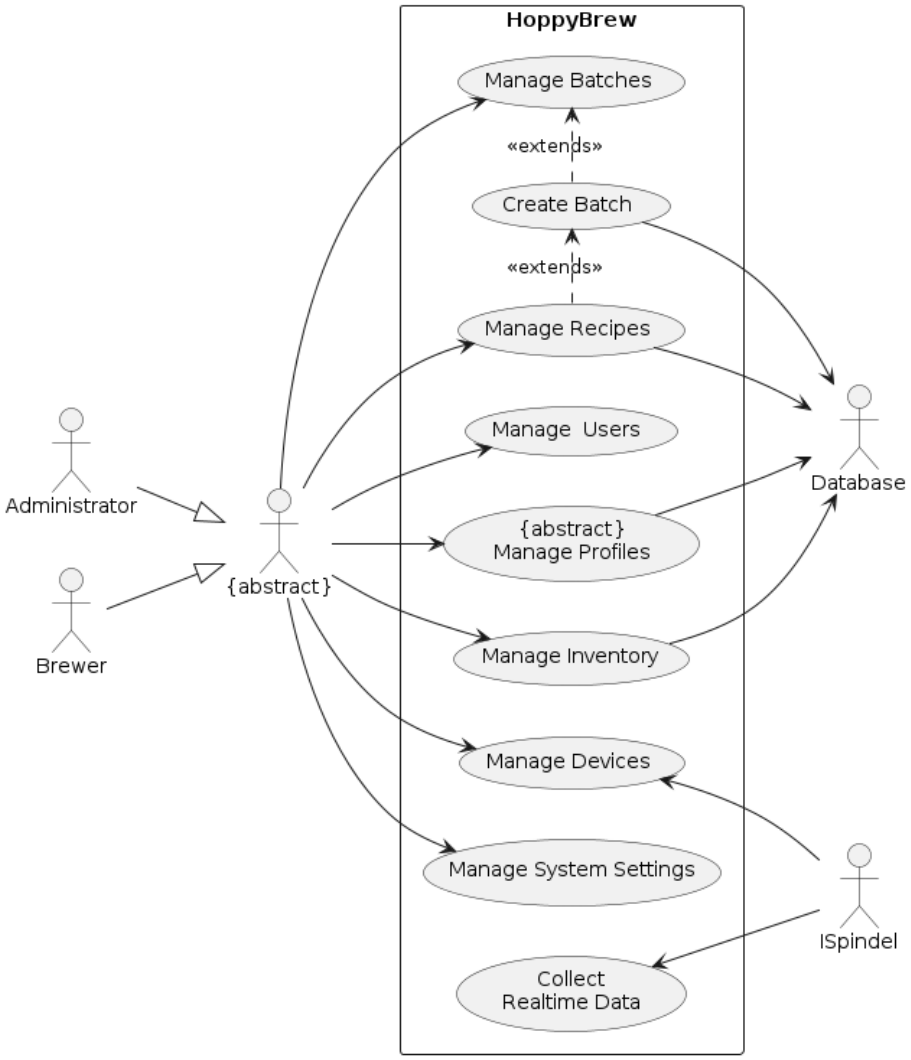
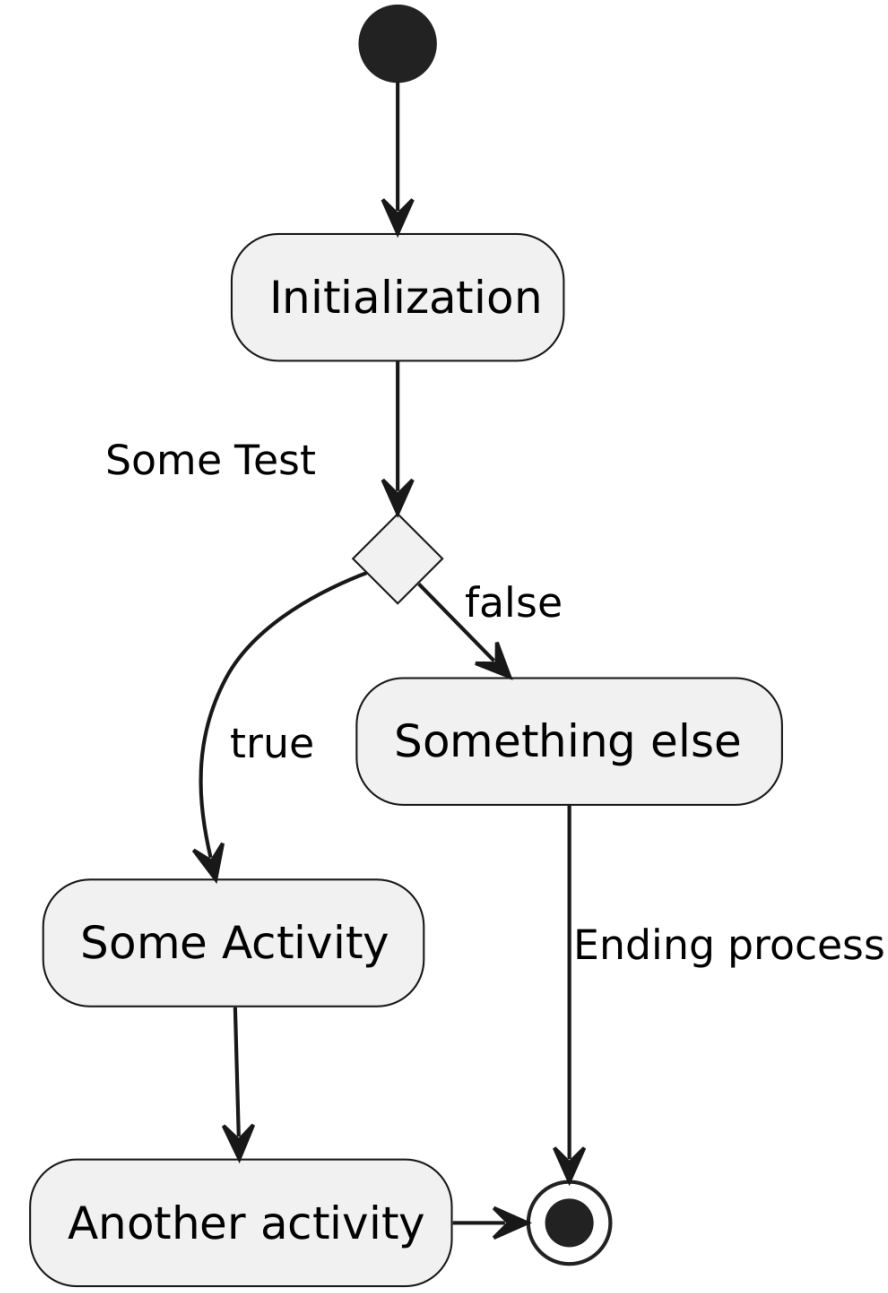


Figure 2: UseCases



Architecture Constraints # System Scope and Context

Business Context

<Diagram or Table>

<optionally: Explanation of external domain interfaces>

Technical Context

<Diagram or Table>

<optionally: Explanation of technical interfaces>

<Mapping Input/Output to Channels> # Solution Strategy # Building Block View

Whitebox Overall System

<Overview Diagram>

Motivation
<text explanation>

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

Important Interfaces
<Description of important interfaces>

<Name black box 1>

<Purpose/Responsibility>

<Interface(s)>

<(Optional) Quality/Performance Characteristics>

<(Optional) Directory/File Location>

<(Optional) Fulfilled Requirements>

<(optional) Open Issues/Problems/Risks>

<Name black box 2>

<black box template>

<Name black box n>

<black box template>

<Name interface 1>

...

<Name interface m>

Level 2

White Box *<building block 1>*

<white box template>

White Box *<building block 2>*

<white box template>

...

White Box *<building block m>*

<white box template>

Level 3

White Box <_building block x.1_>

<white box template>

White Box <_building block x.2_>

<white box template>

White Box <_building block y.1_>

<white box template> # Runtime View

<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.>*

<Runtime Scenario 2>

...

<Runtime Scenario n>

Deployment View

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>

Infrastructure Level 2

<Infrastructure Element 1>

<diagram + explanation>

<Infrastructure Element 2>

<diagram + explanation>

...

<Infrastructure Element n>

<diagram + explanation> # Cross-cutting Concepts

<Concept 1>

<explanation>

<Concept 2>

<explanation>

...

<Concept n>

<explanation> # Architecture Decisions # Quality Requirements

Quality Tree

Quality Scenarios

Risks and Technical Debts

Glossary

Term	Definition
<Term-1>	*<definition-1>*
<Term-2>	*<definition-2>*