Busy Bee

Analysis and Design Document

Student: Carla-Maria Rusu

**Group: 30431**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 18/03/2020 | 1.0 | Document inception. | Carla-Maria Rusu |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

Busy Bee is an event planner and organizer which can be used by an individual or a group of people. It is meant to aid in planning events by giving users the ability to create new events, add contributors, set the contributors’ access rights, provide descriptions for events, set goals, divide tasks, provide dates, location, deadlines, checklists and other sensible information.

Upon completing a task, it will be automatically moved from the “in-progress” section to the “completed” one. The event creator has the ability to set the access rights of members: view only and edit. The event board will persist until a member with edit permission rights deletes it.

Busy Bee is a web application; thus it requires an Internet connection.

# Elaboration – Iteration 1.1

# Domain Model

![A close up of a piece of paper

Description automatically generated]()

# Architectural Design

## Conceptual Architecture

The conceptual architecture pattern chosen for this project is the Micro Services architecture as it is one of the most suitable architectures for web applications. The idea of this pattern is that each of the microservices implemented is completely independent of each other. The microservices basically run on different servers, which enables asynchronous microservice modification without interference with other microservices.

A close up of a logo

Description automatically generated

Domain-driven design (DDD) is the selected architectural style; it is an approach to software development for complex needs by connecting the implementation to an evolving model. The word domain refers to the subject area on which the application is intended to be apply. The reason why this style was chosen is that it enables the creation of complex designs based on the models of the domain. Another feature of DDD is the Building Blocks, namely several defined high-level concepts used to create and modify domain models (such as entity, service, repositories, factories, etc).

A screenshot of a cell phone

Description automatically generated

## Package Design

![A close up of a map

Description automatically generated]()

## Component and Deployment Diagrams

![A close up of a map

Description automatically generated]()

2‑1 Component Diagram

![A picture containing clock

Description automatically generated]()

2‑2 Deployment Diagram

# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

# Data Model

*[Create the data model for the system.]*

# Unit Testing

*[Present the used testing methods and the associated test case scenarios.]*

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

The system is tested according to the use cases present in the Use Case Model document.

# Future improvements

A future improvement would be the integration of a bill splitter system such that the users can benefit from an economic management viewpoint as well.

# Bibliography