# Lifetime Project

Alexandre Zua Caldeira

July 7, 2015

# Contents

# List of Figures



List of Theorems

## Introduction

# Chapter 1

# Architecture

## 1.1 Vaadin Architecture Overview

Our system is a **vaadin based java web application**. It is composed by a server-side frontend, a Java EE 6 backend running on glassfish, and a data subsystem currently using MySQL. Let's start overviewing the generic vaadin architecture, as depicted in (Figure 1.1).

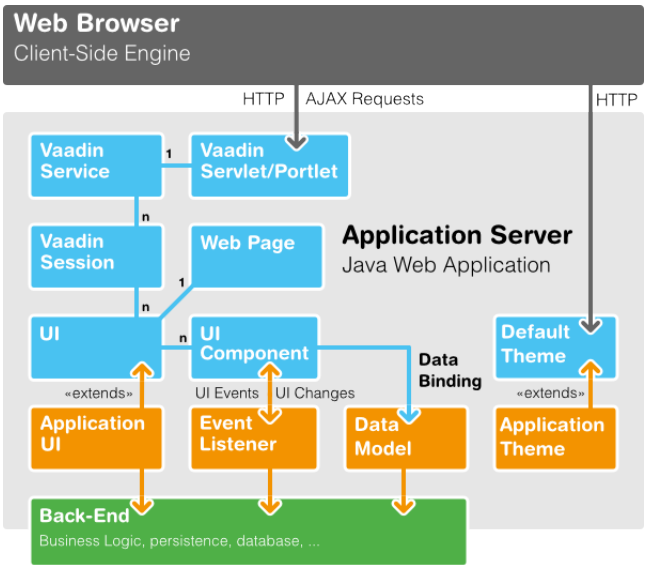


Figure 1.1: Vaadin Java Web Application Architecture

1. Web Browser. Contains the client-side engine of the web application. User interaction will result in request been sent to the server.

2. Vaadin Servlet/Portlet. Every request is intercepted and handled by a servlet. Servlets are server-side components that handle incoming requests, redirecting them as expected. They are associated with a Vaadin Service.

3. Vaadin Service manages the application sessions, keeping request integrity among distinct sessions.

4. Vaadin Session, manages user session and related data. During its lifetime a session may interact with one or more UIs.

5. UI. This is the top-level abstract class defining an application frontend. It intermediates all interaction between the user, the brwoser and the communication with the back-end. Our architecture cuts in providing an application user interface that extends the functionality of UI.

6. Application UI, a generic term referring a concrete implementation of UI. In our case that is the `LifetimeUI`. `LifetimeUI` is responsible to create the views to the user, depending on the url passed to the browser.

7. UI Components, we generally call them *Views*, are the visible part of the application. They present the user with the set of possible actions; according to this architecture they must also register listeners to listen for user initiated events. In lifetime, the top abstract class defining a view is the class `LifetimeView`, responsible to show the application visible part as well as to listen to all events inside it's internal structure.

8. Listeners react to diverse user actions (internally converted as events). Important subsystems depend on listener logic to make the application progreed. **Progress** is an important system property, that must be specified, enforced and tested. We use *session types* to specify navigational progress, ensuring that use cases start, run and terminate as expected.

9. Themes allow modular definition of appearace properties of the web application, using `sass` or `less` as css-based languages.

10. Back-End. Services offered to clients are in the business logic subsystem, and persistent storage for application data in the persistence subsystem. Together with series of restfull web-services and message-queues they compose the application back-end.

11. Data Model, implements a variation of the *MVC design pattern*, the *MVP design pattern*, alliaviating the burn of having the controller to update both the model and the views. By associating a data model, this pattern allow us to have the user data automatically synchronized to a server-side persistent data transparently, using a process known as *data binding*.

The purpose of this chapter is to determine how to extend the vaadin architectural pieces to produce a specific architecture for lifetime. In the next sections we will specify lifetime's specific impementation of the vaadin architecture.

## 1.2 Lifetime Frontend Subsystem

The front-end subsystem is composed by composed by two main blocks: `LifetimeUI` hierarchy and `LifetimeView` hierarchy. According to the architecture listeners and data models belong to `LifetimeView` architectural block, and together they provide the means users access application use cases. The front-end subsystem defines the `LifetimeUI` class hierarchy, the `LifetimeView` class hierarchy, and a factory to create these views. This configuration is presented in 1.2.

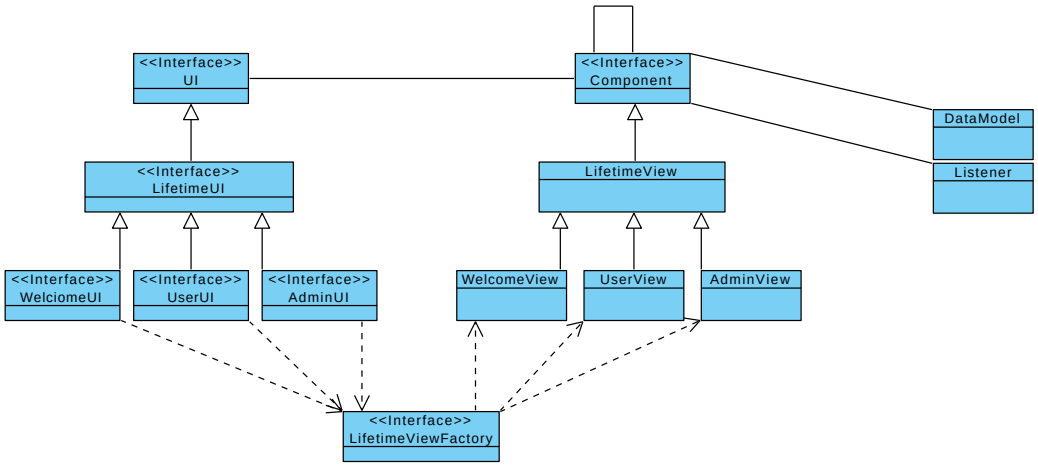




Figure 1.2: Lifetime Frontend

### 1.2.1 LifetimeUI

`LifetimeUI` extends from vaadin `UI` class. Our system features multiple user roles, depending on the service subscription level. At least the following roles must exist in lifetime:

- Guest, to access unsecure parts of the app, like the welcome page, accessible to everyone
- User, to access user area
- Admin, for super-user app access

The `LifetimeUI` and subclass are reponsible to enforce among others the security requiriements of the program. For each role we provide a subclass of `LifetimeUI`, to correctly handle client requests:

- `WelcomeUI`, handles requests to the unsecure part of our app
- `UserUI` handles requests to the user area,
- `AdminUI` handles requests to the admin area,

The `LifetimeUI` (and the subclasses) are reponsible for initiating the correct `LifetimeView`, according to the user role interacting with the system.

### 1.2.2 LifetimeView

`LifetimeView` extends from vaadin `Component` class, and they are initiated during the `init` method of the a corresponding `LifetimeUI` subclass. They are created by a role-dependent view factory. In our application we consider the following views:

- `WelcomeView`, to access unsecure services like registering a new user account or reading the app contact page
- `UserView`, to access user services, like vitae
- `AdminView`, to help administrating the application

## 1.3 Lifetime Backend Subsystem

## 1.4 Lifetime Storage Subsystem

# Chapter 2

# Design

## 2.1 Guest Services

### 2.1.1 Requirements

**Requirement 1** (Web App Location)**.** lifetime is accessed via a web browser, in the address http://localhost/vitae.

**Requirement 2** (Unrestricted Guest Services Access)**.** lifetime welcome page is open and accessible without restrictions to any visitor; unregistered visitors are also known as **guests**. Services offered to guests are specified by the uses cases defined in Section 2.1.2.

**Requirement 3** (Ubiquos Home Access)**.** lifetime welcome page must be accessible from everywhere. It also allows a user to shortcut any process, going to the application startup point.

**Requirement 4** (Automatic Update)**.** Dynamic data must be update automatically, using a.k.o *push* mechanism, and notified to the user.

### 2.1.2 Use Cases

The startup use case is depicted in Figure 2.1. The startup use case **includes** the following use cases:

**Use Case 1** (Register)**.** This use case is the only way to non-previleged become previleged users. The register use case consists on gathering minimal user data to create an account in the system. These data must be validated against predefined business constraints.

**Use Case 1.1** (Add first names)**.** Users input their firstnames.

**Use Case 1.2** (Add last names)**.** Users input their last names.

**Use Case 1.3** (Add email)**.** Users input their email.

**Use Case 1.4** (Add password)**.** Users input their password.

**Use Case 1.5** (Repeat password)**.** User input their password a second time.

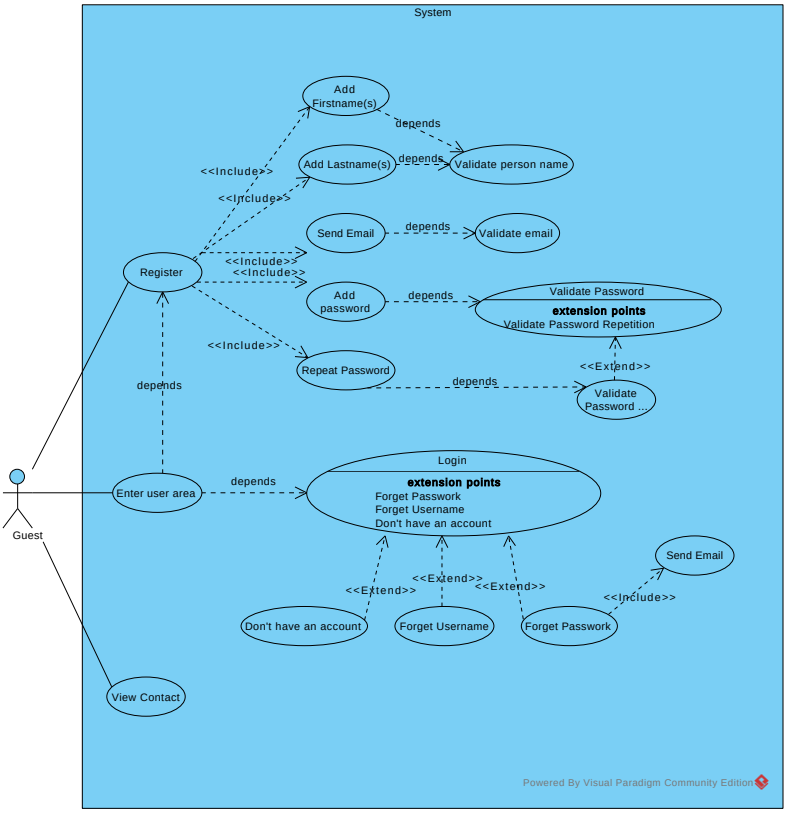


Figure 2.1: Startup Use Case Diagram

**Use Case 1.6** (Confirm data)**.** User sends registration data.

**Use Case 1.7** (Clear data)**.** User clears registration data.

**Use Case 1.8** (Cancel)**.** User cancel the registration process.

**Use Case 1.9** (Validate person name)**.** Validates a person name.

**Use Case 1.10** (Validate email)**.** Users input their email.

**Use Case 1.11** (Validate password)**.** Users input their password.

**Use Case 1.12** (Validate password repetition)**.** User input their password a second time.

**Use Case 1.13** (Validate)**.** User sends registration data.

**Use Case 1.14** (Clear data)**.** User clears registration data.

**Use Case 2** (Enter User Area)**.** Access to lifetime's previleged services for private clients. It depends on the login use case and it's extensions:

**Use Case 2.1** (Login)**.**

**Use Case 2.2** (Forgot username)**.**

**Use Case 2.3** (Forgot password)**.**

**Use Case 2.4** (Don't have an account)**.**

**Use Case 3** (View Contact)**.** For any visitor, gust or previleged, to access corporative, legal and communication services.

**Use Case 3.1** (Kontakt)**.**

**Use Case 3.2** (Impressum)**.**

### 2.1.3 Class Diagram

### 2.1.4 Activity Diagram

### 2.1.5 Tests

**Test 1** (Test App Location)**.** Test that the application is running in the expected location

**Test 2** (Test Security Context)**.** Test that the application welcome page allows access to all kind of users.

**Test 3** (Welcome Services)**.** Test that the welcome page has the following services:

1. Registration process to welcome new previleged users
2. User Private Area, where previleged services are made available to users
3. Contact page and formular, with lifetime's corporative information.

**Todo 1** (Change location)**.** Address should be http://localhost/lifetime

## 2.2 User Services

Users services are made available after successful login. Login dynamically changes the a visitor's role from guest to previleged user. For loged in users lifetime offers the services described in the section 2.2.2.

### 2.2.1 Requirements

### 2.2.2 Use Cases

The use case diagram for accessing user services is depicted in Figure 2.2, featuring use cases below.

**Use Case 4** (Access vitae services)**.**

**Use Case 5** (Access timeline services)**.**

**Use Case 6** (Access yellow pages services)**.**

### 2.2.3 Class Diagram

### 2.2.4 Activity Diagram

### 2.2.5 Tests

## 2.3 Administration Services

### 2.3.1 Requirements

### 2.3.2 Use Cases

### 2.3.3 Class Diagram

### 2.3.4 Activity Diagram

### 2.3.5 Tests

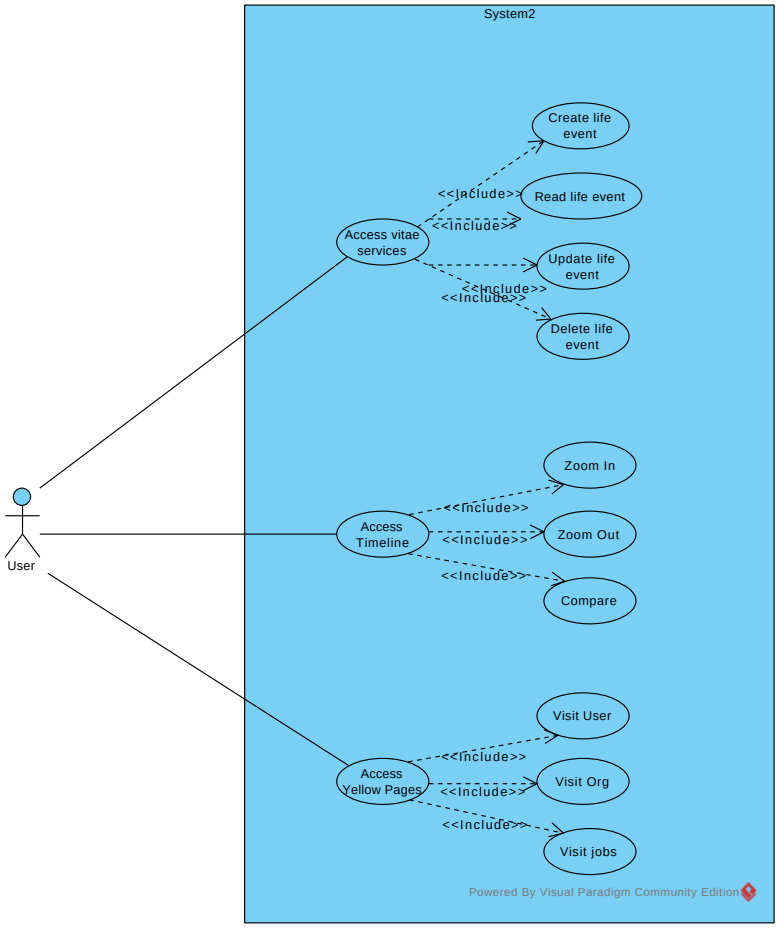


Figure 2.2: User use case diagram

# Chapter 3

# Implementation

## 3.1 Architecture

## 3.2 User Interface

## 3.3 Backend Services