Leopold-Franzens-Universitt Innsbruck

Institut fr Informatik
Datenbanken und Informationssysteme

Development of an OpenSource Feedbackcommuncationplatform

Bachelor-Arbeit

Martin Karrer

betreut von
Wolfgang Gassler und Eva Zangerle

Innsbruck, April 12, 2016

Abstract

Within this work, the development process of an open-source feedback collecting tool with the ability to communicate with the feedback giving person and the maintainer. Most feedback applications only offer a one-way communication from the feedback giving person - to the feedback requester. The approach to develop this system, with the ability to communicate or discuss a feedback, which is extendable, scale-able and especially easy to maintain and install is described in detail. All used techniques and patterns used in this project to implement the single-page-application will be described in this thesis. It tires to show the usage of a functional language in everyday life and all advantages and disadvantages which occurred in this project. The presented application is a minimal but extendable and scale-able open-source platform for requesting and discussing feedback.

Vorwort

Contents

Chapter 1

Introduction

Ever thought about getting honest feedback? This does not work if it is not possible to collect this feedback anonymous. To achieve this, the system has to accept feedback with user information and anonymous without. The application presented in this work can be used to get this information and the feature to discuss.
It is common to use an e-mail mailbox or some online discussion boards to collect feedback. Most of them does not scale for the user. An example: Holding a talk and want to know how it was for the audience of more than 400 people? Send an e-Mail with the feedback does not really scale, because if you have a feedback friendly audience you will have 400 messages in your inbox. Managing Feedback in a discussion board is a huge hassle to find and manage those feedback, without thinking about the ability to discuss a specific feedback.

Chapter 2

Requirement Specification

In the following, a description of the software requirement of Constructeev is given. This chapter describes mainly the requirements, while Chapter 5 gives a deep insight into the development of the application.

2.1 General Requirments

The Application needs to be accessible from all kind of devices, started by mobile-phones, tablets, notebooks and desktop workstations. Web technologies, like Javascript, CSS and HTML ensures that the user only needs to have a HTML5 compatible Browser, which is pre-installed on all of the devices listed above. All feedback relevant data need to be stored permanently on a storage medium and should be accessible at any time. It is important to keep the data in a structured way and it should be possible to change the database adapter and migrate data vom one to another database system without any hassle.

The Application needs to be spitted into two segments in a so called client- and server-application. Both applications should run on as many platforms as possible without any constraints to productivity and scalability.

2.1.1 Server Application

The server application should be scalable and runnable on almost any operating system and platform. Furthermore it has to provided a uniform interface to the client application, which will be in this case a JSON-based REST-like API. The datastorgae has to be a common SQL-Database like mysql or Postgresql. It should also be possible to exchange the database at installation time.

2.1.2 Client Application

Same as the server application, the client application needs to run also on almost any platform but targets end-user hardware with an display and a HTML5 compatible Browser. AngularJS is used to build a single-page-application with a model view controller pattern on client side. Asynchrone API calls will communicate with the application server.

2.2 Server System Requirements

The server system has easy to be set up and also easy to maintain. The pure application resulting from this project is also running as a public service available to anyone. So scalability is also an important key point. A channel with more than 16.000 feedback should be as responsive and useable as one with only 20 feedback. Not only the amount of data should scale, also the number of requests should be easy to handle by adding more and more servers.

2.3 Channel Components

To create a easy to use application it is important to be minimalism. A channels task is to hold a specific quantity of feedbacks for a specific topic set by the admin of the channel. There is always one admin per channel. Users which need more than one channel will get seperate login hashes for each channel. This makes sharing a company channel or community channel easy.

2.4 Feedback Components

2.5 Scalability

2.6 Administrator View

Chapter 3

Related Work

3.1 Arsnova

3.2 Feedbackbutton

Feedbackbutton is a small widget button which can be added to any website. When those button is clicked a pop-up windows is opened and a small formular is presented to the user, where you have to fill in your name, e-mail, category, and a comment. The written and sent feedback are stored in a DB on the company server. The tool is commercial, closed source and the user does have to pay a fee per month based on the amount of feedback which will be sent. For 50 feedback you have to pay 6$ per month 400 feedback 12$, 1200 feedback 18$ and for unlimited feedback 99$ per month. There is no way to customize the form or the send you feedback in an anonymous way.

3.3 Feedbackify

http://www.feedbackify.com

Chapter 4

Used Technologies

In this chapter a brief overview of the technologies used in this project is given, because most of them are not widely used or known. First an overview of the functional language called Elixir and about some open source frameworks/modules used to build the application with this fairly new language. The remaining chapter will describe the more common used front-end technologies and the server platform.

4.1 Elixir

Elixir is an alternative language for the Erlang Virutal Machine (ErlangVM) that allows the programmer to write code in a compacter and cleaner way than in Erlang itself. You write your code in Elixir and compile it to a BEAM-Bytecode and run it like any byte-code from Erlang in the BEAM symmetric-multi-processor Environment (beam.smp). BEAM stands for Bogdan/Bjrn's Erlang Abstract Machine, which tried to compile the Erlang Code in early Erlang version to C to gain more performance. Nowadays BEAM is a performant and complete process virtual machine and does not compile any code to C, because the effort is too high for the minimal performance gain on modern CPU Systems. Elixir is an open source project started by José Valim and now maintained and developed with more 373 contributors. The source code of elixir can be found on the Github Repository at `https://github.com/elixir-lang/elixir`
One major advantage of Elixir as a young programming language is that the well tested, proven and hardened BEAM Virtual Machine is used underneath. A second advantage is that Elixir produces the same byte-code as Erlang, hence you can use Erlang libraries in Elxiir and vice versa. There is nothing that can be done in Erlang that can't be done in Elixir and usually the Elixir code is as fast and performant as it's Erlang counterparts.

Elixir provides a huge set of boilerplate and reduces duplication in code and also simpifies some important parts of the standard-library and provides some syntactic sugar and a nice tool for creating and packaging applications.
Another cool feature offered by Elixir is Metaprogramming, whereby it is easy to extend and define the language itself and basically write code which writes code for you. With macros you can abstract and uniform Elixir code parts to restructure and abstract more complex code away from the programmer.

4.1.1 Phoenix

Phoenix is a web framework written in Elixir and implements a server-side MVC pattern, which depends on some other projects in the Elixir and Erlang Ecosystem:

- **cowboy:** lightweight super fast Erlang web server
- **ecto:** a query and database wrapper
- **PhoenixHTML:** working with HTML and HTML safe strings.
- **Plug:** a specification and conveniences for composable modules in between web applications
- **poison:** a pure Elixir JSON library
- **Poolboy:** a worker pool factory
- **Ranch:** a socket acceptor pool

Phoenix is very similar to other modern web-frameworks like Ruby on Rails or Python's Django. The core developer team of phoenix tries to provide the best of both with Elixir and the ErlangVM. Simple and elegant code and a high performance real-time application running over multiple CPU's or even Nodes.
Let's look into the Phoenix pipeline for and the layers a single request will pass:
After a request was accepted by the cowboy webserver it will be hand over to the **Endpoint**, which applies all necessary function (also from plugs) to the request before it will be passed into the **Router**. The Router parses the incoming request, discovers the assigned Controller and provide some helper functions for routes and path and url's according to the controller. Also a pipeline can be specified in the controller, which makes encapsulation of different scopes and API's easy. The Router will pass the request to the assigned **Controller**. In this segment the functions, also known as actions, are called to provide the

main functionality. The controllers main task is to prepair data and pass it into views, invoke the view rendering and or perform http redirects. After the **View** got all the data it renders a template and provides self defined helper function which can be used in the template. The View acts like an presentation layer in Phoenix. **Templates** are pre-compiled and in are only string concatenations in the background, thus blazing fast.

4.1.2 Cowboy

Cowboy is written in Erlang and is optimized to a low latenzy and low memory usage. It uses Ranch for managing connection it is pure Erlang and is easy to integrate on every platform without extra dependencies. Cowboy provides a complete HTTP Stack with support for HTTP1.0, HTTP1.1, SPDY and Websockets. The Cowboy project has a small and clean codebase, is well tested and provides a rich documentation.

4.1.3 ETS

ETS is an abbreviation for **E**rlang **T**erm **S**torage. Erlang term storage was build to hold a very amount of data inside the Erlang runtime, to have constant access time to the data. Inside the ETS data is stored in dynamic tables and each table is hold by one process. Those tables only have the lifetime of a process, so if the process terminates the data will be destroyed. Via message passing it is possible to request data from one process to another. Message passing works also between nodes, so all data is available in the whole cluster.

4.1.4 Ecto

Ecto is not only database wrapper it also provides macros for a nice query DSL inside Elixir, so you do not have to escape queries. A Adapter for the database you want to use is needed. Ecto repositories are wrappers around the desired database. With Ecto repository, it is possible to create, update, destroy and drop custom queries. Changesets are also brought by Ecto and provides filters and casting functionality for external parameters. Another security layer in front of the database.

4.2 PostgreSQL

PostgreSQL is a widely known open source relational database system. It runs on all major platforms like Windows, Unix deviates and Linux. It has full ACID (Atomicity, Consistency, Isolation, Durability) and supports all standard SQL statements like foreign key, joins, views, triggers and procedures.

4.2.1 Schema

4.3 AngularJS

4.4 Semantic-UI

4.5 Server-Platform

Chapter 5

Implementation

Chapter 6

Usability and Responsive Design

Chapter 7

Performance and Scalability

Chapter 8

Conclusion

Appendix

A.1 Subsection Appendix