Overview

In this lesson, we'll look at a quick overview of an app and how its components were integrated.

WE'LL COVER THE FOLLOWING

Introduction

The example in this chapter is a classic assurance application designed to help office workers interact with customers.

It was created as **a prototype for a real insurance company**. The example shows how a classic application can be implemented as a web application with frontend integration.

The example shows **the division of a system into several web applications** and the integration of these applications. ROCA is used as the basis for this. This is an approach for implementing web applications (see the last chapter), which has a number of fundamental advantages, especially for frontend integration.

The example was created as a prototype **showing how a web application can**

be implemented with ROCA and what advantages it offers.

The INNOQ employees Lucas Dohmen and Marc Jansing implemented the example.

Let's look at each web application the system is divided into.

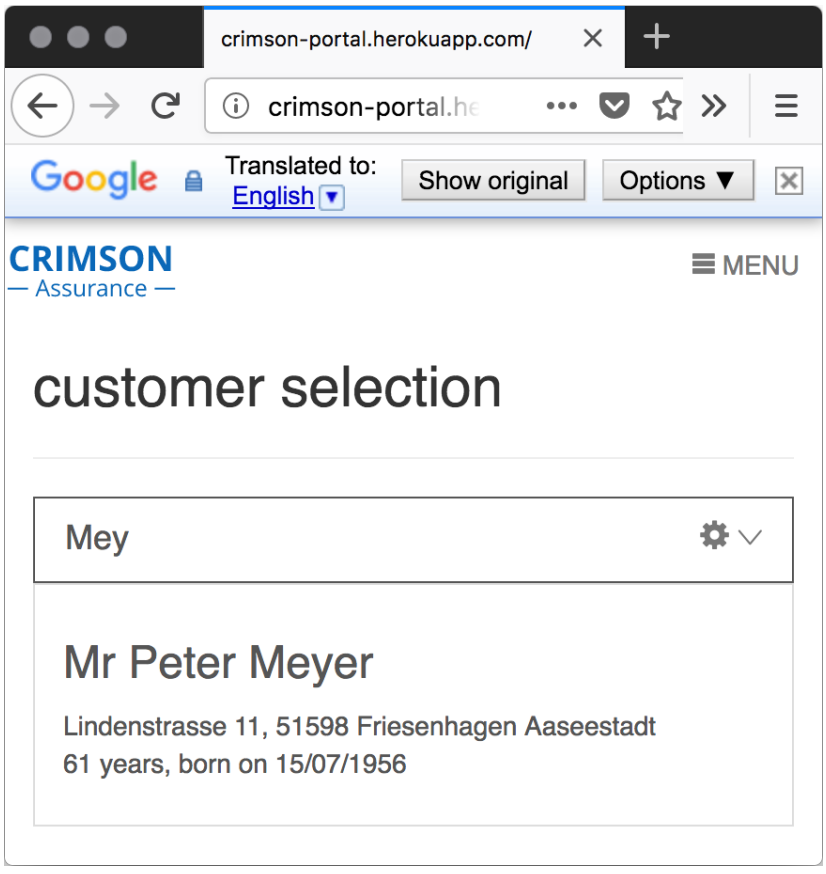
Search

The insurance application is available at https://crimson-portal.herokuapp.com/. It can also be run on a local computer as a Docker container.

The application is in German. However, nowadays browsers can translate web pages into other languages. The screenshots were taken with the Firefox browser and Google Translator's translation from German to English.

An input line appears on the main page to search for customers. Have a look at the screenshot below.

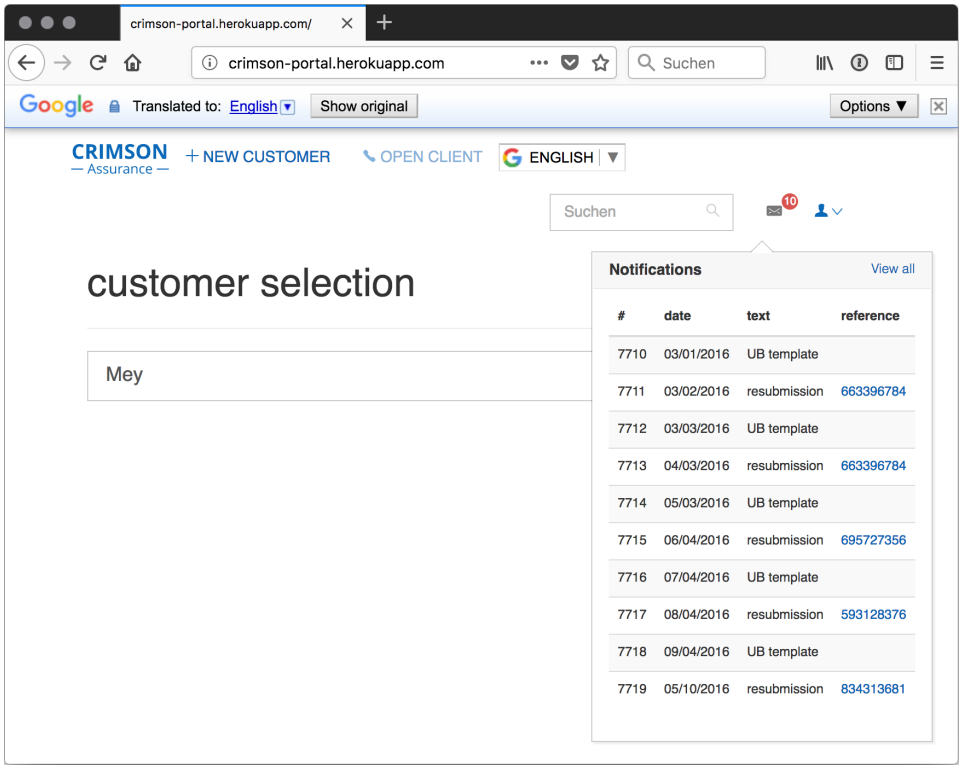
While the user enters the customer's name, matching names are suggested. For this, the frontend uses jQuery.



Screenshot Insurance Application

Postbox

Another functionality is the *postbox*. With a click on the postbox icon on the main page, the user gets an overview of the current news. The overview is displayed in the current main page with JavaScript. Have a look at the figure below.



Overlayed Postbox

The entire application **uses a uniform frontend**.

However, if you look at the address line, you will notice that several web applications are used.

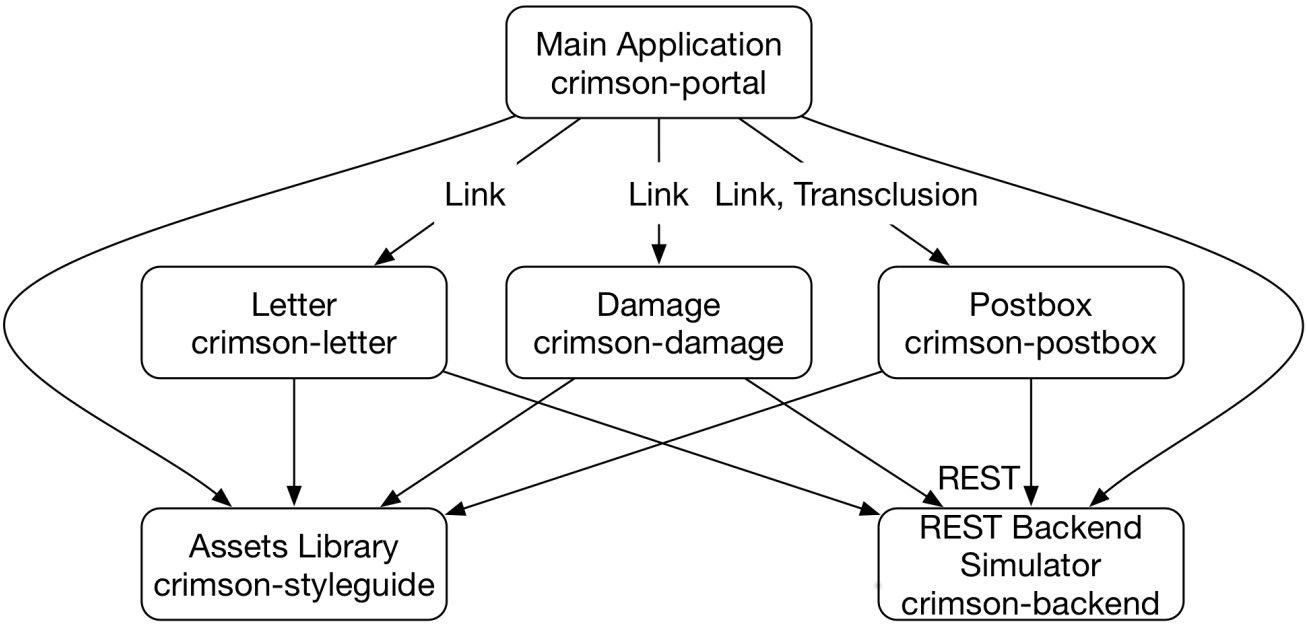
There is **one web application each**.

- for the main application (https://crimson-portal.herokuapp.com/)
- for reporting damages (**damage** application) (https://crimson-damage.herokuapp.com/)
- for writing letters (**letter** application) (https://crimson-letter.herokuapp.com/)
- for the postbox (**postbox** application)(https://crimson-postbox.herokuapp.com/).

Nevertheless, the frontend has the same look and feel for all these

applications.

Structure of the application



Overview of the Insurance Example

The drawing above shows how the different applications are integrated.

Main application

The *main* application is used to search for customers and to display the basic data of a customer. The code is available at https://github.com/ewolff/crimson-portal. There you will also find instructions on how to compile and start the application. The application is written in Node.js.

Assets

The *assets* at https://github.com/ewolff/crimson-styleguide contain artifacts used by all applications to achieve a consistent look and feel. The other projects refer to this project in the `package.json`. This allows the npm build to use the artifacts from this project. npm is a build tool specialized in JavaScript. The asset project includes CSS, fonts, images, and JavaScript code. Before the assets are used in the other projects, the build optimizes them in the asset project. For example, the JavaScript code is minified.

Damage

The *damage* application for reporting a damage is also written in Node.js. The code can be found at https://github.com/ewolff/crimson-damage.

Letter

The *letter* application is written in Node.js as well. The code is available at https://github.com/ewolff/crimson-letter.

Postbox

The code for the *postbox* can be accessed at https://github.com/ewolff/crimson-postbox. The postbox is implemented with Java and Spring Boot. To be able to use the shared asset project, the build is divided into two parts. The Maven build compiles the Java code, whereas npm is responsible for integrating the assets. npm then copies the assets into the Maven build.

Backend simulator

Finally, the *backend simulator* can be found at https://github.com/ewolff/crimson-backend. It receives REST calls and returns data regarding customers, contracts, and so on. This simulator is also written in Node.js.

In the next lesson, we'll discuss how this app was integrated.