Table of Contents

Dedication

Abstract

Glossary

Chapter 1. Introduction

This chapter describes the goals that motivated the development of this software. It also specifies the format of this document.

1.1. Motivation

Back in 2012, I was researching about the death era of monolithic applications, and how the industry was moving towards a separation of front-end and backend. I talked to the community about this research in the conference session *"Embrace the front-end revolution"*, given at Codemotion Rome in March 2013.

Traditionally, software developers have been used to create applications where the views and the business logic, although probably separated in software layers according to best practices, are packaged together in the resulting application deployable artifact. This approach has several disadvantages:

- **Front-end** and **backend** developers have to work in the same project, using probably a server-side based framework such as Spring MVC for Java EE or Rails for Ruby. This will require so called full-stack developers, as the views are not simply HTML, but a framework-specific view templating technology.
- **Deploying** and **scaling** such monolithic applications is complicated, as it all have to consider the whole package.
- **UI changes** cannot be made and deployed independently of the business logic, due to the way of packaging and deployment. This reduces or even prevents doing frequent interface changes to improve converstion, A/B testing, etc.

Those problems were not significant to the development industry because the majority of the developers were considering themselves **full-stack developers**: proficient in doing backend and front-end. However, some years before a technology was created and it changed the game rules: **Node JS**.

Node JS not only enabled for the first time using the same language for front-end and backend (Javascript), but also set the foundation for a large number of tools, such as Grunt or Gulp as bulding tools, Bower as dependency manager and Yeoman as project creation and scaffolding tool. Those tools attracted a large number of developers that were highly skilled in HTML5, CSS3 and Javscript, and for the first time in forever had a specific toolset where no backend was involved.

On the other hand, backend development was experiencing the rise of the new buzzword of the moment: **microservices**. After several years since REST architecture was first introduced, and more and more teams trying to adopt such architecture with relative success, the industry finally agreed on a preferred way of building the business logic: they should be REST API's, producing and consuming JSON over HTTP.

The last necessary contributors in this shift were the frontend development frameworks, like Ember but specially Angular JS (built by Google), that effectively enabled the posibility of creating applications where the UI was entirely executed in the browser, and the communication with the server was using the mentioned JSON over HTTP.

The API's that the backends were converted into had a special requirement per the definition of REST (*REpresentational State Transfer*): they had to be **stateless**.

More or less, developers managed to create stateless busines logic, using the front-end frameworks to keep client state and ensuring that such state was sent to the server on each and every round-trip. However, authentication and authorisation was different. **Security** in a web application has traditionally been a stateful service: it heavily relies on the HTTP Session, which itself represents a state in the server side. Therefore, there was a friction when applying stateless restrictions to operations like login or logout.

The result of this developent ended up in the development of **Spring Security REST**: a plugin for Grails framework, on top of Spring Security, that offered developers a way to implement authentication and authorization in a stateless way, as well as it made the necessary transformations over core Spring Security to make it more REST-friendly.

1.2. Goals

The goals of this project are closely related to the motivations described above, and are the following:

1. Make core Spring Security, which only offered form-based authentication using the HTTP Session, more REST-friendly.
2. Offer developers a way to implement authentication and authorization in a stateless way.
3. Analise, design, implement and test a robust solution, stable and highly covered by tests.
4. Contributing back to the community by creating an open-source solution.

This document describes in detail all the stages of the process followed to achieve the mentioned goals.

1.3. Rest of the document

asdasd

Chapter 2. State of the art

2.1. Introduction

UNIVERSIDAD CARLOS III DE MADRID
homo homini sacra res
Universidad Carlos III de Madrid

Figure 1. Cool caption

2.2. Similar solution #1

2.3. Similar solution #2

2.4. Comparision

Chapter 3. Analysis, design, development and deployment

3.1. Introduction

3.2. Analysis

3.3. Design

3.4. Development

3.5. Deployment

Chapter 4. Evaluation

Chapter 5. Planning and budget

5.1. Planning

5.2. Budget

Chapter 6. Conclusions and future improvements

6.1. Conclusions

6.1.1. Product

6.1.2. Process

6.1.3. Personal

6.2. Personal improvements

Appendix A: Documentation

Bibliography