UNIVERSITY OF REGINA

ENSE 477: CAPSTONE PROJECT

REQUIREMENTS AND SPECIFICATIONS

Telport: Sasktel Telecommunications Portal

Authors
Dakota FISHER
200 344 336

Quinn BAST
200 352 973

Supervisor
Dr. Yasser MORGAN

University of Regina

Last Modified
February 6, 2019

Revision History

Revision Version	Revision Author	Revision Date
1.0	Dakota Fisher	February 4, 2019

Contents

List of Figures

List of Tables

1 Introduction

This document outlines and defines the requirements and specifications. The initial problem statement and proposed activities are suggestions given by Sasktel to provide a manageable scope. The decisions on infrastructure, aside from the Broadworks API and telephone account access are left up to us to decide.

1.1 Problem Statement

SaskTel requires a communications portal that can interwork with a Telephony Application Server and our core network to present communications and feature capabilities through a browser. This will allow for the exploration of new communications service models.

SaskTel has been pursuing the deployment of a new communications core along with the Cisco/Broadsoft Telephony Application Server branded as Broadworks. One of the drivers is to enable a richer customer experience through a converged architecture that exposes rapid development to enables new capabilities. Broadworks exposes the Application Programming Interface to access service control tools and user information. These tools and information can be used to create new communications applications or add additional value to existing applications.

The main objectives for SaskTel is to gain exposure to new and innovative communications service experience for our customers and to promote the potential internally for aligning resources, time and effort in enabling applications.

1.1.1 Proposed Activities

- Establish communications portal and platform interworking
- Gain a high-level understanding of the communications network architecture
- Gain an understanding of a method to access and use TAS APIs
- Establish base interworking and web browser communications portal
  - Access and exposure to TAS API
  - User registration through IMS
  - User presentation and interaction via portal
  - Exposure to 4-5 basic features to validate interworking i.e.
    (listed only as suggestions)
    * Call forwarding
    * Display of call logs (All, Incoming, Outgoing, Missed).
    * Simple call blocking by using a slider. Simple drop down to show numbers blocked (allow for unblocking).
    * Directory. Searchable by typing any string of characters.
- Enable WebRTC communications
- Enable the use of WebRTC for internet communications directly through the portal
- Create innovative communications experience
- Explore feature capabilities and experiment with innovative communications capabilities
- Demonstrate and showcase the ability to grow and share knowledge.

1.1.2 Skills Required

General Knowledge Requirements:

- Network Platform and service specific knowledge
- Software engineering skills
  - Client / server operation
  - IP Network Protocols (SIP, RTP, UDP, HTTP ...).
  - Internet Methods (Using XML, JSON, REST, ...).
  - Web Browser Programming (HTML, CSS).
  - Programming Languages (Java, Python, Ruby)

API Technology Comments:

- Most popular approach to delivering web APIs is REST (Representational State Transfer).
- API returns data in either XML or JSON.
- Most popular implementation is REST+JSON (not a standard but widely accepted within the industry).

API Usage Comments:

- Must consider how a resource will be manipulated not just retrieved.
- Should have strong understanding of both client-side and server-side programming.
- Should have strong understanding and experience with HTML and CSS (Cascading Style Sheets) for web programming, development, and design.
- Should have strong understanding and experience with Java Script.
- Should have a strong knowledge and experience with dynamic programming languages making Python and Ruby emerging industry favorites (Python+Django or Ruby+Rails).
- Should have knowledge and experience of the user and use of the product interface (in regard to forming the interface).

2 Design Decisions

During the creation of the project, we had full reign of the architecture and languages in which we wanted to use in order to create the solution for SaskTel. As a result, we did a significant amount of research on the different technologies available in order to make the best decisions for the project. While making these decisions, there were two different sides of the project to focus on. The backend would manage interfacing between SaskTel's servers as well as manage user credentials for the system, while the frontend would display the interface to the user. Both the backend and the frontend had may different decisions which led us to the current solution.

2.1 "Backend"

When creating the backend, initial considerations were to use a basic LAMP stack, which would include PHP as the backend language. This decision was initially chosen because it is the most commonly known web stack. Because of this, it was likely that SaskTel would have a number of employees, if required, could take over the project and understand how to make changes. After our initial meeting with SaskTel, it was indicated that there were no restrictions on the technologies that we could use for the project, and therefore, instead of using PHP, we opted to use a python backend as python was our team's preferred coding. At the same time, this allowed us to look into new frameworks to gain additional experience with new technologies. The leading technologies of python backends being Django and Flask.

2.1.1 Server side scripting framework

The choice of the server side scripting language was determined to be python after the first meeting with SaskTel. After researching the different frameworks that were available for the backend, two competitors stood out, Django and Flask. Extensive research was done into both of the technologies. Django was a full-fledged backend system, allowing for object-relational

models, MVC configuration, admin interface, templates, caching, and much more straight out of the box. Django is feature-rich, however, the general consensus is that Django is generally more restrictive than Flask because of their additional functionality. For example, Django wouldn't let you switch out their admin portal or ORM frameworks for ones that you prefer (or, if they did, it would be difficult and tedius to do so). Flask on the other hand, was a minimal backend providing less functionality, but was extremely good at providing a stable backend server, as well as allowing more freedom over which libraries to use. After trying a simple project in both Django and Flask to test the waters of both frameworks, Flask was chosen over Django. This decision was because Flask is a smaller and more stable framework while providing the ability to customize. Being more comfortable with Flask, it was the chosen as the backend server side scripting framework for this project.

2.1.2 Backend Framework Decision Timeline

Date	Design Decision	Reasoning
September 14th, 2018	Technology stack	Our initial thoughts for the technology stack were to use a basic LAMP stack. This decision was made because it would favor SaskTel, as it was the most common technology stack and would allow anyone familiar to be able to make changes if required.
September 24th, 2018	Technology stack	After meeting with SaskTel it was clear that there were no restrictions on the technology that we chose to use to implement the solution. As a result, we chose to use a Python based backend as it was the language of choice.
October 18th, 2018	Server-side frameworks	Created sample repositories for trying out the difference between Flask and Django.
October 23rd	Flask backend	After testing between Django and Flask in sample repositories, Flask ended up being the most comfortable environment to develop in. As a result, Flask was chosen to be the backend framework for the project.

2.2 "Frontend"

Information about initial frontend thoughts (HTML) conversion to choice of JS framework.

2.3 Frontend frameworks

Explain our findings between different frontend frameworks and which we ended up choosing. (Maybe won't need all the additional subsections)

2.4 Node.js (In Use)

2.5 JavaScript

2.5.1 React (In Use)

2.5.2 Angular2 + Typescript (Not In Use)

2.5.3 Vue.js (Not In Use)

2.5.4 Frontend Framework Decision Timeline

Date	Design Decision	Reasoning
Date	Design Decision	Reasoning
Date	Design Decision	Reasoning
Date	Design Decision	Reasoning
Date	Design Decision	Reasoning

3 Automated testing

3.0.1 Mocha (In Use)

3.0.2 Jest (Not In Use)

3.0.3 Chai (In Use)

3.0.4 Sinon (In Use)

3.0.5 Enzyme (In Use)

3.0.6 Selenium (In Use)

3.0.7 Cypress (Not In Use)

4 Tools

The following sections denote tools that are utilized or were considered to be of use during the project's development. Programs and tools, alongside their effective alternatives are discussed to get an understanding of what, and why tools were picked or not picked to be used to benefit the work flow of the project. Every reference to an application in the following sections has a direct hyperlink reference if clicked on using an electronic medium.

4.1 Github

Git is a powerful source version control solution that allows for multiple developers to collaborate on a project. The Git provider used during this project is Github. Not only does it allow for collaborative features, it also allows for parallel streams of development of many features. The tool also allows for public display of the project, to allow for open source development if the repository is not kept private. A key feature and benefit of using Git is that it allows for projects to commit checkpoints and milestones in development, such that if an update isn't beneficial, it can easily be reverted to a previously confirmed viable version.

4.2 Toggl

Toggl is a time tracking web application that allows teams to track their hours along with the project, and tags that correspond to the work that they are doing. This allows for us to track documentation time seperate from programming time and research time. While still keeping a total cumulative track of our time spent on the project. Ideally, the time tracking entries are relative to our commits on Github. Of the options online at the time, this is the only option that we felt met the requirements we needed to easily and painlessly manage our project time.

4.3 GitKraken

The tool GitKraken is a powerful Git version control client that has an easily manageable interface. It also has the ability to see how each branch correlates to one another, and resolve any conflicts within the application directly. There isn't any particular reason to use this tool over another alternative such as GitHub Desktop, SourceTree, or GitBash, but it definitely was a nice discovery for our quality of life work flow.

4.3.1 Glo Boards

The company that created GitKraken, also has a product called Glo Boards. The application is functionally equivalent to Trello. Both programs allow for a team to manage issues on a GitHub repository easily and visualize changes into KanBan board styles. The decision to use a KanBan board software instead of just the issue tracking software on GitHub was because it provides additional work flow improvements while still allowing GitHub issues to still work normally.

4.4 Google Suite

4.5 Discord

4.5.1 Alternative: Slack

4.5.2 Alternative: Skype

4.5.3 Alternative: Messenger

4.5.4 Alternative: WhatsApp

4.5.5 Alternative: Hangouts

5 Security Considerations

The main concern of our application was the security of a user who would login to our application. When a user logs into a website, their credentials should never be seen or be accessible except by the authorization agency. In our case, the authorization agency is SaskTel's API. Therefore, our server acts as a proxy between SaskTel's API and itself, forwarding authentication requests to SaskTel's servers.

Security was a huge concern, and therefore many different security methods were researched. These methods included LocalStorage, SessionStorage, Redux, IndexedDB, JSON Web tokens, CSRF Tokens, Two Factor Auth, OAuth, and 3rd Party Logins. With so many different methods of 'security' avaliable, we wanted to make sure that we were using the best of the best, leaving no room for vulnerabilities.

5.1 Missing the obvious

While reading SaskTel's API documentation, there was no indication of how a user logs into their API. Generally, when logging into a website, the user provides their username and password, and once the username and password have been confirmed, the server will provide the user's browser a 'token'. The browser sends this token every time the user makes a request so that the server can acknowledge that the user has already logged in. This prevents the user from having to enter their username and password in while navigating around the website.

Because of this knowledge, after examining the available REST endpoints we assumed the endpoint named "LoginToken" was used to generate a token for the user to use for repeated requests. After looking at the endpoint's response, this looked like exactly what we needed to authenticate a user with SaskTel. However, the only problem was that "the generated token has an expiry of 60 seconds." This caused a large problem for our application. How were we

going to keep this token alive for the user to be able to access SaskTel's servers? Generally a user won't change pages or request data every 60 seconds, thus after 60 seconds of idle activity, this token would expire. If the token expires, the user would need to login again, meaning we would need to send their username and password. We didn't want users to have to login to the site every 60 seconds as that would be extremely annoying and immediately make a user not use the application. At the same time, we did not want to store the user's credentials on our server as they would have to be stored in plain text due to the repeated required use to refresh the token which is a huge security violation to the users of the website. This proved to be a difficult challenge.

Various ideas were devised including sending a refreshed token every 50 seconds to get a new token before it expired, storing the passwords in the browser, and many other options; none of which we thought were secure enough for a user to be comfortable with. After a couple of weeks, it became clear that our supposed "challenge" was not an issue at all. While examining the different types of data that SaskTel responded with from their various endpoints, we found that SaskTel's servers responded with a cookie alongside the regular HTTP request. This cookie turned out to be a token that the user could use to access SaskTel's endpoints with an unlimited expiry. Once this information had been found, authenticating a user became extremely easy using the methods outlined below. To this date we still don't know what the 'LoginToken' endpoint does or why it exists.

5.2 Security Research

5.2.1 JSON Web Tokens

5.2.2 LocalStorage

5.2.3 SessionStorage

5.2.4 Redux

5.2.5 IndexedDB

5.2.6 CSRF Tokens

5.2.7 Two Factor Authentication

5.2.8 Third Party Login

5.2.9 OAuth

5.3 API Communication

5.4 Data Integrity

SaskTel->XML->JSON->Client SaskTel->XML->XML->Client->JSON

6 User Stories

The following sections denote common use cases for the program that we brainstormed and consider to be requirements by usability and basic principle. The basic structure for user stories is as follows. "As a <type of user> I would like to < be able to do task> so that I can < metric improvement to life>". User stories are referenced by the number following the chapter value, which means user story 1 will refer to sub-section x.1.

6.1 Log In

As a *SaskTel Customer*, I would like to be able to *Log into the program* so that I can *use the program and check my information*.

6.2 Log Out

As a *SaskTel Customer*, I would like to be able to *Log out of the program* so that I can *ensure that anyone using my computer can't access my account*.

6.3 Check Call Logs

As a *SaskTel Customer*, I would like to be able to *check my call logs* so that I can *check when I got calls, and see who called me/when*.

6.4 Turn on Call Forwarding Always

As a *SaskTel Customer*, I would like to be able to *always forward calls sent to my phone* so that I can *answer them on my main phone and not worry about my secondary phone*.

6.5 Turn on Call Forwarding Busy

As a *SaskTel Customer*, I would like to be able to *forward calls when my phone is busy* so that I can *feel comfortable that those calling me get sent to someone who can answer*.

6.6 Turn on Call Forwarding Selective

As a *SaskTel Customer*, I would like to be able to *forward phone calls to another phone during a scheduled time* so that I can *so that phone calls can be forwarded when I have planned events in the way of answering my phone*.

6.7 Turn on Call Forwarding No Answer

As a *SaskTel Customer*, I would like to be able to *forward calls when I'm away from my phone* so that I can *answer them on my cell phone when I'm away from my land line*.

6.8 Turn on Call Forwarding

As a *SaskTel Customer*, I would like to be able to *forward phone calls to another number* so that I can *be sure that important calls reach me on my cell phone or work phone*.

6.9 Turn on Do Not Disturb

As a *SaskTel Customer*, I would like to be able to *decline all calls to my phone number* so that I can *have some time without worrying about phone calls interrupting me*.

6.10 View my Profile

As a *SaskTel Customer*, I would like to be able to *view my account information* so that I can *know that I am using the right account*.

6.11 Call a Phone Number

As a *SaskTel Customer*, I would like to be able to *call a phone number from my computer* so that I can *easily make phone calls without using my phone*.

6.12 Start a Call to a Phone Number from my Phone

As a *SaskTel Customer*, I would like to be able to *call another phone number from my phone* so that I can *more easily call phone numbers I find on my computer*.

6.13 View Feature Access Codes

As a *SaskTel Customer*, I would like to be able to *check what star codes I can use* so that I can *understand what I can do with my phone when I can't log into the application*.