UNIVERSITY OF REGINA

ENSE 477: CAPSTONE PROJECT

REQUIREMENTS AND SPECIFICATIONS

Telport: Sasktel Telecommunications Portal

Authors
Dakota FISHER
Quinn BAST

Supervisor
Dr. Yasser MORGAN

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

4.1 Git

4.2 Toggl

4.3 GitKraken

4.3.1 Glo Boards

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 plaintext 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

After resolving our issues indicated above, it now boiled down to how we were going to keep the user's information secure while they used our application. There are a significant number of ways in which security can be implemented and thus, research on the various methods of browser and backed security began. Our research on security took upwards of 3-4 weeks before we started to implement any of the features that we had found. We wanted to be certain that our research and chosen method of security offered the user the most protection while using our

application. After our extensive research we chose to use JWTs (JSON Web Tokens) along with CSRF (Cross-Site Request Forgery) tokens to provide users the most security. The following sections outline our finding about various security measures and explain why we chose to use JWTs and CSRF tokens.

5.2.1 JSON Web Tokens

https://jwt.io/introduction/JSON Web Tokens are the industry standard for sending information between two parties. A JWT, as the name implies, sends a JSON object of information along with any requests that need user validation. This JWT Token will hold information such as the user's ID, or username so that the backend knows who is sending the request without requiring any other information. The backend application uses a secret hash key which is used to encode the JWT when it is sent to the client. Because the backend server is the only thing that knows the secret key, only the backend is able to decode the JWT to extract the information. This authorization technique was the most beneficial and is a fairly secure method of both authenticating and sending user information inside of a cookie. Because JWTs are both the industry standard the most secure method of authentication that was found, JWTs were used to securely verify a user's session.

5.2.2 CSRF Tokens

While JWTs are an extremely powerful way to send information between two parties, a JWT alone is not a secure method of communication. If a hacker gains access to your JWT, the hacker is able to send requests to the website acting as if they are you. Consider this scenario: You are logged into the application and leave the page to browse the internet. You access a malicious site which knows how the urls of our web application to make requests. The malicious site sends a request to our application hoping that you were previously logged into the application. The request would go through because the request would appear to be coming from you. In order to prevent this type of activity, a CSRF token is included on the webpage of the website

which is randomly generated for each user. This token is impossible to guess by the malicious website and ensures that without the additional token, the request is invalid. Therefore, the use of an additional CSRF token in conjunction with JWTs make our application secure for a user, not having to worry about stumbling across malicious sites or man-in-the-middle attacks.

5.2.3 Flask-Session

https://pythonhosted.org/Flask-Session/Flask-Session is a Flask extension that allows configuring session variables inside of the Flask application. This is useful for a backend server to obtain information about a specific client's session when they make a request to the server. However, the creation of a REST (Representational State Transfer) API means that the server should not hold any information about the 'state' of various sessions or users. Having the backend server stateless ensures that the same behaviour is applied based on the information that is provided to the server. Because we wanted to create a REST application that was stateless in nature, Flask-Session was not chosen to store information about the user's authentication.

5.2.4 Secure Socket Layers

SSL is a requirement for secure communication between web applications. SSL enables a web application to use the HTTPS (HTTP Secure) protocol for sending requests, thereby ensuring that any traffic to and from the server is securely encrypted during transit. Without the use of SSL, requests made to a backend server are not encrypted when they are sent. This means that the information is visible as it gets sent, allowing the packets to be inspected as they travel across the wire and their information extracted by anyone who is tracking information travelling through their network. Because SSL is such a secure method of transportation, we obtained an SSL certificate and applied it to our server to enable secure communications between our web application.

5.2.5 Two Factor Authentication

Two Factor Authentication (2FA) is an extra security layer on top of current authentication schemes. 2FA ensures that anytime a user tries to login to a website, and additional randomly generated code is required to access your account. This code gets sent to the user's mobile device to ensure that the person logging into the web application is truly the user. 2FA was an initially planned feature for the application, as this would greatly increase the security of our application. However, as we began developing, 2FA became too much overhead to add into the application. In order to implement 2FA, on top of figuring out how to implement it into our application, we would also need to store user specific information about 2FA, such as each user's phone number. We chose to forgo this feature because it would save a large chunk of development time, as well as prevent our application from storing a user's personal information.

5.2.6 Third Party Login

Third Party Login is the ability to login to the application through an external website such as Facebook, or Google. This is extremely useful, as usually a user is already logged into Facebook or Google while they are browsing the internet. Therefore logging into the application would be as simple as clicking a single button and authorizing the application. However, this option was also forgone. Our development team decided that logging in to our web application was as simple as typing in your phone number and password. Everyone knows their own phone number, so signing into the application is already relatively easy. Additionally, figuring out how to develop and manage third party login information would require additional development time that we could not allocate.

5.2.7 OAuth

5.2.8 Auth0

5.3 API Communication

5.4 Data Integrity

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

6 User Stories

6.1 Log In

6.2 Log Out

6.3 Check Call Logs

6.4 Turn on Call Forwarding Always

6.5 Turn on Call Forwarding Busy

6.6 Turn on Call Forwarding Selective

6.7 Turn on Call Forwarding No Answer

6.8 Turn on Call Forwarding

6.9 Turn on Do Not Disturb

6.10 View my Profile

6.11 Call a Phone Number

6.12 Start a Call to a Phone Number from my Phone

6.13 View Feature Access Codes