ECE590 Cloud Computing Technical Project Report

Guodong Hu & Qing Guo
Department of Electrical and Computer Engineering

1. Overview

This is a technical project report for our Sharebox. In this report, we start by clarifying our goal and motivation for the project. We then reason through our technical approach adopted in this project and explain the system in details. Challenges in the development are also mentioned to provide a whole picture of the development process. The final state of the system and future plan for the project are also included in the end of this report.

2. Goal & core features

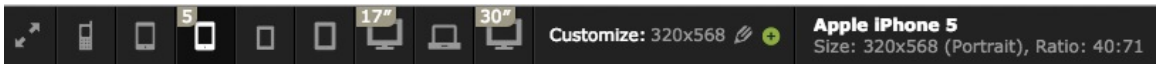
With the advancement of semiconductor technology, computers and mobile devices have dominated our lives for quite a long time. With the ubiquitous Wi-Fi access, it is a good practice to back up files when possible. By building Sharebox, we could backup our files in the remote server without concerning about the lost of files. However, upload and download functionalities are clearly not enough for a web application, which aims to be robust and user-friendly. We want to make the web-app secure enough to let users trust and enjoy our service. To achieve this goal, we should first of all enable the mail confirmation service to defense against malicious registration by robot. Similar functionality such as locking the account should also be implemented to avoid brute force attempt to break into the file sharing system. In case users may forget their passwords, retrieving password functionality should also be added. Since it is a web application, it is also important to make it responsive to fit for different size of screen, be it a mobile screen or desktop screen. Encrypted password and interceptor are also favored to prevent unauthorized users breaking into the system. Other core features of the system include the response to every user operation and share link functionality.

3. Technical Approach

Since Qing Guo has rich experience in front end development while Guodong Hu has interest in backend development, we decided to use Ruby On Rails as our main technical approach for development. The model view controller pattern of Rails enables us to separate the tasks easily and only focus on our own work. For example, Guodong Hu spent most of his time in the models and controllers folder in the app while Qing Guo enjoyed most of his time manipulating front end technology in views folder. In addiction to MVC, Rails has default structures for database, web

pages and web services, which are easy to use and save a lot of codes. Another reason for our adoption of Rails is that it encourages the use of web framework such as jQuery and Bootstrap. The active record pattern also makes the operation of database easier and comfortable.

In the front end, we use “viewport resizer” to simulate a mobile devices viewport (Viewport resizer is a browser-based tool to test website’s responsiveness). There are many options for different popular mobile devices in the world (e.g. iPhone5, iPad3).



We adopt Bootstrap 3 to create the websites. Bootstrap 3 is an open-source framework which provides numbers of components for us. The grid system of Bootstrap has been widely used in our application. Grid system is a powerful mobile-first layout strategy for building websites of all shapes and sizes. It’s based on a 12-column layout and has multiple tiers (extra-small, small, medium, large, extra-large). You will know how grid system works by seeing the following table.

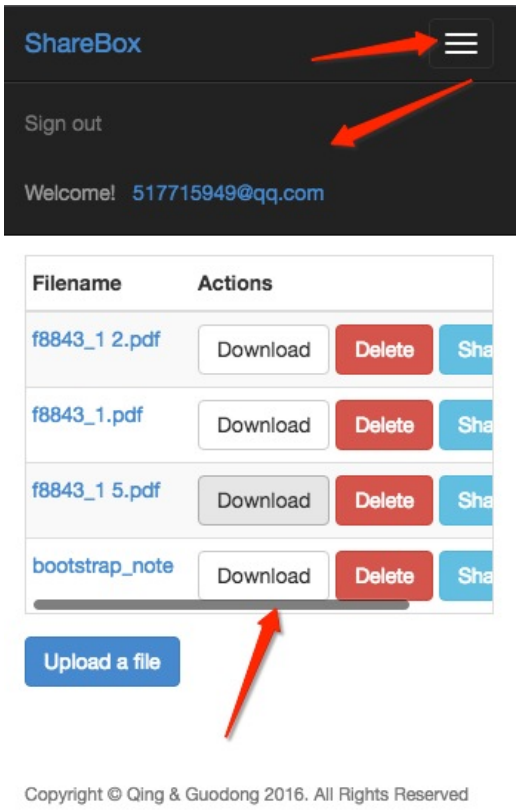
	Extra small <544px	Small ≥544px	Medium ≥768px	Large ≥992px	Extra large ≥1200px
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints			
Container width	None (auto)	576px	720px	940px	1140px
Class prefix	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`
# of columns	12				
Gutter width	1.875rem / 30px (15px on each side of a column)				
Nestable	Yes				
Offsets	Yes				
Column ordering	Yes				

（The figure above is from: http://v4-alpha.getbootstrap.com/layout/grid/ ）

For the navigation bar, all the items will be collected into a single dropdown button, as you can see from the picture below. That is because a fluid navigation bar component has been integrated in our websites.

In main page, we adopt responsive table structure instead of common fixed table. This flexible table can adjust its size according to viewport width so that nice and

clean web pages can be guaranteed in different mobile devices. Even if the viewport size is pretty small, a scroller will appear in the table so that user can move the table easily.



We chose the AWS S3 service for managing our file system because it was free and it provided enough services for our project. Besides, we had heard a lot about how big AWS was and how powerful it was in class. It was a good opportunity for us to get to know AWS by ourselves and get hands-on experience in it. Another consideration was that AWS provided many security features to help us make our system robust.

We first focused on the user authentication design. While we initially attempted to build the whole authentication system from the ground up, it was not a long time before we realized how complex the system might be if we wanted to keep it robust enough. After careful scrutiny and consideration, we found devise was what we were looking for. Devise is an open source project providing a flexible authentication solution for Rails and it can be found in Github. It is Rack based and a complete MVC solution that is compatible with Rails. It has default setting for simple user authentication and can be customized and configured with more complex and useful functionalities by adding modules of it. The modules we have used is Database Authenticatable, Confirmable, Recoverable, Registerable, Validatable and Lockable.

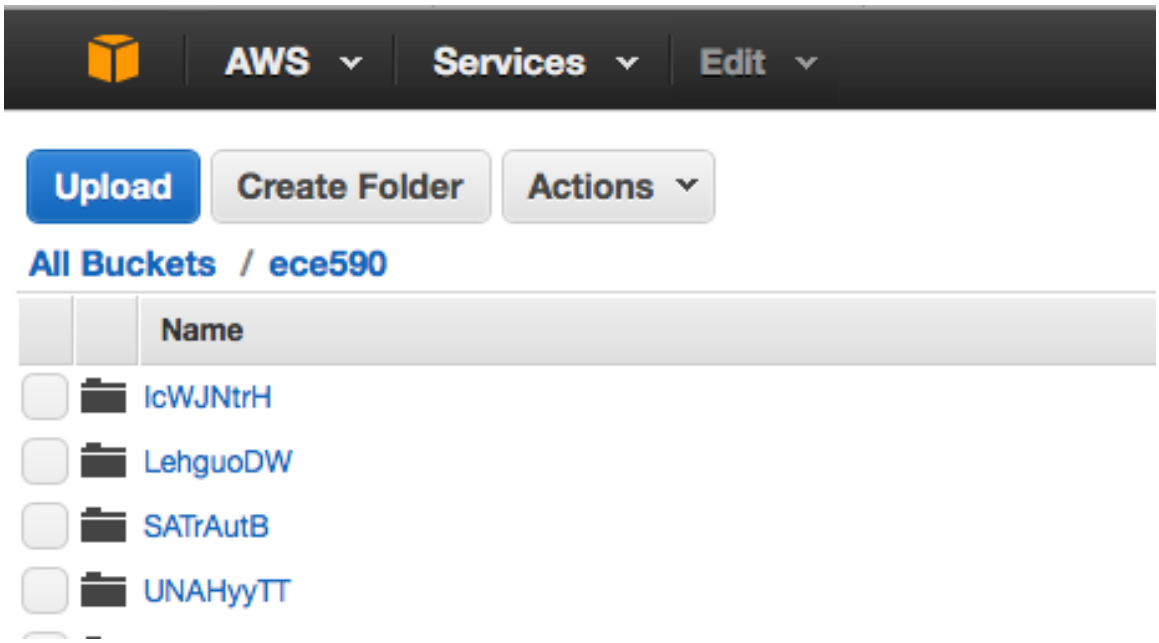
Specifically, the Database Authenticatable module handles the hashing and storing passwords in database and authenticity of user while signing in; the Confirmable module enable us to send email to a user with confirmation instruction and verify whether the email is confirmed before the user logging into the system; the Recoverable module resets a user's password by sending a reset email; the Registerable module guide users through the registration process and can be customized to allow users to edit and delete their accounts; the Validatable module provides validation of the email and password in the sign up process and can be customized to force user to make a strong password; the Lockable module is used to lock a user's account after a specific number of failed sign-in attempts and it is how we defense again the robot's brute force sign in attempts.

To manage the user's file effectively and securely, we create a random name folder for every user in AWS S3 bucket when it's the first time the user log in. The name of folder is comprised of random length of lower case and upper case alphabet. By doing so, we hide user's information when the user wants to get the share link of his file and it may protect the leak of user's information if our AWS account is compromised since at least no specific user's information will be targeted. The ruby code for this implementation along with the AWS S3 bucket representation are shown below:

```ruby
def index
  if current_user.folder.nil?
          current_user.folder = generate_random_user_folder_name
          current_user.save
  end

  @uploads = Upload.where("userId = ?", current_user.id)
end

def generate_random_user_folder_name()
      letters = [('a'..'z'), ('A'..'Z')].map { |i| i.to_a }.flatten
  return (0...8).map{ letters[rand(letters.length)] }.join
end
```



4. Challenges

There were mainly three challenges in this project. The first challenge was to deploy the app in Heroku. Since we had never used Heroku before, we didn't realize that Heroku used Postgres as its default database system and didn't support SQLite3, which was used by Rails by default. We found the solution in Heroku Dev Center and followed the instruction in Heroku Posgres documentation. We first changed the gem file in our app to install pg gem when the app was deployed in heroku and changed the adapter in our database configuration file to progresql. After a successful database migration, the problem was solved.

The second challenge was to improve the security of user account. We tried to do as much as we could to deny unauthorized sign in and defense against brute force attempts to get our users' information. At first, we attempted to implement all functionalities such as password encryption, email confirmation and lock. However, we found it tricky and not plausible. We started some researches about improving security in Rails and found devise a best fit for our project. Then the rest of the work was to learn how to customize and configure devise in our rails app.

The third challenge and the most important one was to understand the importance of managing securely the AWS Security Credentials. Since we managed our app development via github, the updated code would be pushed to the github repository. To test the connection to AWS S3 bucket, we wrote the AWS_SECRET_ACCESS_KEY and AWS_ACCESS_KEY_ID in our aws configuration file and forgot to hide it before

pushing the code to github. The next morning, one of us received a call from AWS team and knew that our AWS account was compromised. After we logged into the account, the bill was going to 30,000 dollar and we were extremely shocked by that. Luckily, with the help from AWS team, we made our account clean and secure again by stopping and deleting the unauthorized usage. Through this process, we understood how IAM could be used to limit the access to AWS account and most importantly, never write the AWS ACCESS KEY ID into the file.

5. Final State of System & Future Plan

Now our app supports all the core functionalities mentioned in the second section of the report. It will ask the user to click the confirmation link in the registered email box when he/her sign up and will lock the account if the user fail to type in the correct password five times when signing in. Restring password is also supported by sending instruction email to users. The interception functionality is also implemented to avoid illegal break in. The responsive feature is also tested by Viewport Resizer, a useful Google Chrome extension. Though we are now facing some problems which are related to Postgres when pushing our last version of web app to Heroku, it is expected to be fixed in a short time.

In the next iteration, we will be first focusing on timeout functionality, which we believe can improve security of our file sharing system. Specifically we want to automatically log out the user's account if he or she does not do any operation in a specified period of time.

The second functionality we want to implement is to give user more flexibility and control in the file sharing system. We want to allow users to delete their account if they feel they would not use their accounts anymore. We also want to build a more functional file system to allow user to build folder in their Sharebox. And hopefully, the file sharing system could automatically categorize the files uploaded by users based on the file type and save time for users.

Another important feature we may implement in the future is to set free usage limit for every user and charge the user for going over the free limit. It is also desirable to let users set alarm for their usage of the file sharing system and get notified when they are going over the limit they have set.

6. References

1. Direct to S3 image Uploads in Rails
https://devcenter.heroku.com/articles/direct-to-s3-image-uploads-in-rails
2. AWS S3 Documentation v1
http://docs.aws.amazon.com/AWSRubySDK/latest/AWS/S3/S3Object.html
3. Heroku Postgres
https://devcenter.heroku.com/articles/heroku-postgresql

4. Getting Started with Rails
http://guides.rubyonrails.org/getting_started.html