Senior Design Final Report

Amazon Web Services Project

Version 1.0 - 05/11/2021

Team Members:
Pasindu Siriwardena
Nancy Rosa
Saul Rugama
Luis Lucero
Junli Wang
Carlos Mendoza
Rongfeng Tan
Bryan Perez
Jesus Gracia
Jorge Sanchez (Grad Student)

Faculty Advisor:
Dr.Jiang Guo

Liaisons:
Mahan Hajianpour
Jeffrey Lipeles

Table of Contents

1. Introduction:

1.1. Background:

Commonwealth Casualty Company (CCC) was founded in 2010 with the sole commitment to provide affordable insurance to everyone in the community. The products and services offered at CCC include property and casualty insurance that cover Auto, Homeowners, Renters and Roadside protection. The services are constantly improving and are customized to each and every customer to provide innovative solutions when needed. The current system that manages the inner workings of CCC is created by a small team of developers/engineers and kept locally. However, moving forward the team wants to transfer all of the workings to the cloud. Working closely with California State University Los Angeles, the team at CCC has presented a project that involves taking the current system and moving it to the cloud. There is also the addition of the generation of documents using a templating engine, PebbleTemplates, that will allow for the generation of contractual agreements with customers.

Amazon Web Services or AWS, is a cloud computing platform that offers over 200 fully featured services. AWS offers reliable, scalable and inexpensive services that can be accessed as long as proper credentials are given. For this collaboration with CCC, this project uses 3 main services from AWS (S3, RDS and EC2).

- S3 is used as a simple object storage system that is accessible through a web service interface
- RDS is a distributed relational database that runs on a web service hosted on the cloud
- EC2 is a web service that allows applications to run on virtual computers on the cloud.

1.2. Design Principles:

As the web application grew more and more complicated, it was designed so that its different parts and technologies can be found separate from one another as much as possible. The use of Spring for framework, allows for all our dependencies to be in one place. From here, one can see a list of most of the technologies used for the web application. With it being a web application, with use of bootstrap, the front end of the application was designed to make for simple use for the user. All the data displayed to the user whether from RDS or S3, is displayed using Pebble Templates. This would only pose a problem to the web application if the application is not fed proper credentials to get access to and from AWS. The user can then read and write data on RDS or S3 through the web application.

1.3. Design Benefits:

The separation of technologies in the application makes for easier debugging. With the addition of more technologies, it makes the application susceptible for bugs in different places. Separating the technologies and their integrations as much as possible makes for easier debugging. Making the application a web application makes for a simpler way of running the app, just needing an internet browser. The use of AWS allows for availability of resources if more are needed. Having AWS available in the web application also makes for simpler reading and writing of data for users. No need to give the users credentials to have access to AWS, as they are embedded in the web application. This also makes for easier version control files in S3, as files with the same name are labeled with an upload date and time, and gives the user access to see the different versions.

1.4. Achievements:

In the allotted time in both semesters beginning Fall 2020 to Spring 2021, we were able to incorporate the knowledge we gained throughout our academic careers with a lot of new technologies for this web application to reach its requirements. We started off simple by first trying to understand what Pebble Templates did and how we can use it for our application. We then worked our way up to incorporate it in our web application to display data from a database in AWS RDS, and files in an AWS S3 bucket. The application can read and write data from AWS. To do so, we incorporated Pebble Templates. With the use of Java Persistence API (JPA) the web application can take the data input by the user and store it in the RDS database. This would not be possible if we did not use AWS-java-sdk dependency to use in Java to read and write the data from RDS. It does so similarly with an S3 bucket. The team successfully implemented conversion of the document templates from html to pdf files that can be sent to a client in the database via email.

2. Related Technologies:

2.1. Existing Solutions:

This project involves a combination of multiple technologies to create a final product that can be used in industry. Knowing this we had some options when deciding which technologies to use so that we would create the most efficient and secure end product. Some of the existing parts of our project involve the templating engine itself.

After careful consideration of many templating engines, we decided to use 'PebbleTemplates' as our templating engine because it allowed for a much faster creation of documents. When compared with other engines such as 'Thymeleaf' and 'Freemarker', PebbleTemplates provide a much more stable generation when used on a higher scale. PebbleTemplates is also easy to integrate into our SpringBoot framework as it only requires a single dependency to function.

2.2. Reused Products:

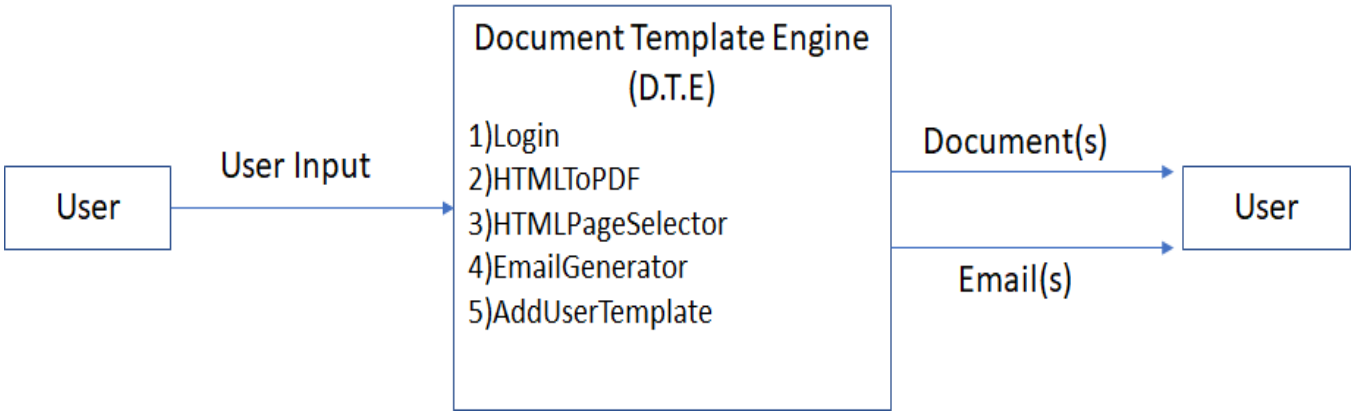
In terms of reused products, the use of an open source html document comparison was the only product. It allowed the application to internally compare the document contents and display the differences between them. Other than that this project contained everything that was created from the ground up. All the AWS cloud computing services were created from scratch and then linked to the project. The application itself was created on SpringBoot and is a standalone application that is held on EC2 which can be used from anywhere. The local MySQL database, which is ultimately connected to an RDS database, is also created from scratch using mock data that would be enough to create clients and the documents pertaining to those clients.

3. System architecture:

3.1. Overview:

The architecture for this project was simple as it was a standalone application that communicated with AWS cloud services with the help of an authorized user.

Below is an image of DFD 0 for our application.



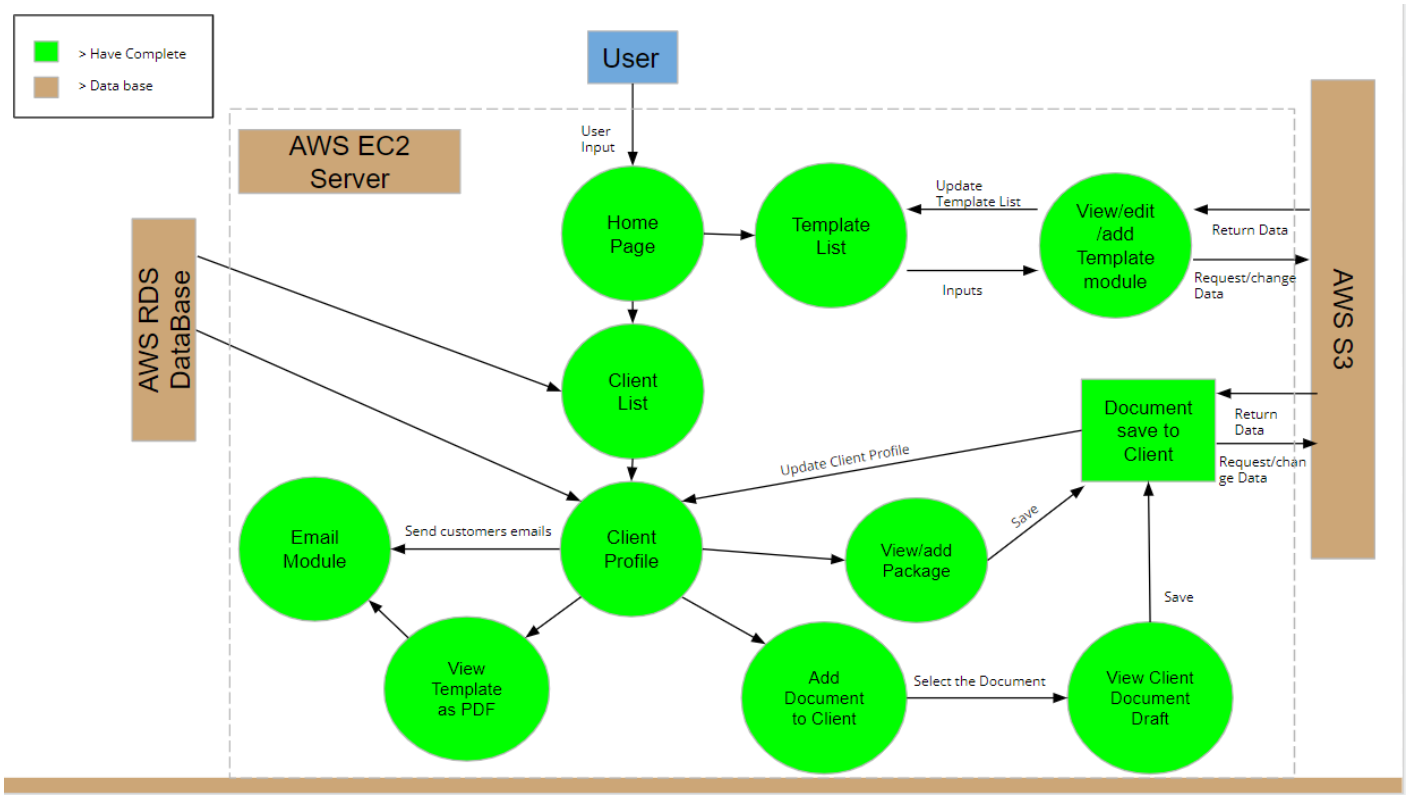
DFD Level 0

Our project will implement already existing features from the current system at CCC. This includes a login module, and a collection of users and their policy information. Its current features include adding policy information to existing users and adding new customers. Some of the added features not in the old version are:

- Generating a Doc/PDF from any selected template
- Incorporating DocuSign into the Doc/PDF to allow digital signatures
- UI to view and modify templates
- Version Control including archival view and comparison of templates & generated documents
- Dashboard of current documents available for generation by product and state
- Provide a searchable/filterable list of all documents available to be generated by the system & allow filtering by state, product, dynamic/static
- Be able to send email with existing template-generated documents.

3.2. Data Flow:

Below is the Data Flow Diagram Level 1 of our application:



DFD Level 1

Home Page: The home page automatically redirects to the client list.

Template List: This page is responsible for getting the buckets available in the AWS S3 bucket and displaying it as a list to the user. From here, the user may click a button to redirect to the View/Edit/add Template module.

View/Edit/add Template Module: This webpage allows the user to view a particular template that is available from the template list. Users may choose to edit a template and any change done to the template that is saved is sent to the S3 bucket. Users may upload a template from their local machines to the S3 bucket.

Client List: This webpage displays a table of the clients that are in the AWS RDS database. The user may redirect to any one of the client's profiles.

Client Profile: This webpage retrieves more details of the client chosen from AWS RDS and displays it to the user. It also shows the user the documents that have been linked to this client, if any. From this module, the user may redirect to 4 different modules: Email module, View Template as PDF, Add Document to Client, and View/add Package.

Email Module: The user may compose an email from this module to send to the client from the Client Profile where the button to redirect to this module was clicked. After writing a subject and a body for the email, the user can also attach any document(s) linked to the client. Once ready, the user clicks send to send the email to the client's email retrieved from RDS.

View Template as PDF: This module will show the user a pdf file of the chosen template filled in with the client's information.

View/add Package: The user can see any available package to link the client. A package is a bundle of document templates to link to the client, without having to add the document to the client one at a time. Viewing a package will show the user the list of document templates in that package. Adding a package is choosing from the available document templates to bundle together.

Add Document to Client: The user may choose from the list of available document templates to link it to the client. Linking a document creates a version of the document template with the client's information in the document template.

View Client Document Draft: This is an option before adding the document template to the client, it shows the user the document filled in with the client's information, it is intended to show the user the document before choosing to link it to the client.

3.3. Implementation:

When implementing this system, the idea was to have the AWS services as the most important parts of our projects. As shown in DFD 1 above, the entire system is encapsulated by EC2 to show where the system is located. Then led to the creation of the webpage that allows the user to interact with the two other services from AWS. The web page allows a user to fetch and populate documents from S3 and also add them to clients that have been allocated in the RDS database. The implementation of this project was meant to combine these three services into one system that could be run from anywhere. Each of the four main parts of our system were created so that their most efficient and advantageous aspects would be reflected.

- PebbleTemplates
    - Easy to use templating engine that performs well under heavy data. PebbleTemplates also work efficiently with SpringBoot and only uses a single dependency to begin the connection.
- EC2
    - Fast and efficient cloud hosting for our application and acts as the foundation. Easy integrated with other services and only requires a single jar file to get up and running.
- S3
    - A file structure system that provides easy access to documents and the ability to version control these documents. Very easily accessible as long as the correct endpoint is given.
- RDS
    - A relational cloud database that is easy to set up and scale when needed. It is connected to a local database so that changes done locally are logged onto the cloud.

4. Conclusions:

4.1. Results:

We have created a standalone application that is hosted on an EC2 server which creates contractual agreements for clients. This application uses the user data stored in an RDS database, that is automatically version controlled by AWS, and populates templates that have been fetched from S3 to create documents. The application is split into two parts, the front end web application and the backend data/template storage.

We have populated the RDS database with mock client data that has primary and foreign keys for each entry, so that it can display a real client data system. This makes creating the documents and polluting them easy as we can reference real documents from CCC to mimic all the needed syntax and styles. An authenticated user may add a new client to the company and it will be reflected on the database in real time. The same applies to the templates that are in S3. Any edits that are done to any templates will be saved on S3 as a new version of the template. These versions can be deleted whenever so that the original template is the only one remaining.

4.2. Future:

Improvements to our standalone application can be done to create a smoother and more secure product. Our system could be improved in web application, but also the backend which is AWS. The two work hand in hand so any improvements and/or changes will have an effect on both sides. The following are improvements that can be done:

- In terms of the templating engine, the current implementation recreates a file on our system and then populates it. Since this is very time consuming, a better approach would be to directly retrieve the documents from S3 and populate it in memory and then display it to the user. This would be the best approach because it can save a lot of time and memory, and when dealing with a large amount of data, this becomes very valuable.
- S3 documents are currently set to be publicly available with the correct endpoint. Instead of having this, there should be some sort of security in place so that only authorized users can view the documents. Currently the system does allow for admin rights or any members rights, but that can be implemented directly through AWS to allow only some members to be able to access parts of the system.

5. References:

Amazon Elastic Compute Cloud Documentation:
https://docs.aws.amazon.com/ec2/?id=docs_gateway

Amazon Relational Database Service Documentation:
https://docs.aws.amazon.com/rds/?id=docs_gateway

AWS SDK for Java Documentation: https://docs.aws.amazon.com/sdk-for-java/index.html

Amazon S3 Documentation:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

Bootstrap Documentation:
https://getbootstrap.com/docs/5.0/getting-started/introduction/

GitHub Docs: https://docs.github.com/en

Html2Pdf Documentation:
https://github.com/spipu/html2pdf/blob/master/doc/README.md

iText 7 Java Documentation:
https://itextpdf.com/en/resources/api-documentation/itext-7-java

MySQL Connector/J 8.0 Developer Guide: https://dev.mysql.com/doc/connector-j/8.0/en/

Pebble Templates Documentation: https://pebbletemplates.io/

Spring Framework Documentation:
https://docs.spring.io/spring-framework/docs/current/reference/html/

Upload and Delete files with Amazon S3 and Spring Boot:
https://springbootdev.com/2018/08/02/upload-and-delete-files-with-amazon-s3-and-spring-boot/