Jake Stocker

Capstone Final Report

I was part of the creation of a Workflow application and specifically charged with creation of the Backend’s Restful Client. The Workflow application is an application that RoviSys will use when creating proposals for future clients. Desired functionality would allow RoviSys to quickly switch out different parts of a proposal. A proposal will have different key data elements that must be stored. Some pieces of the proposal will be repeated between different proposals. Front-End ideally will create a drag and drop Graphical User Interface that allows the user to quickly swap out different parts in the proposal and to save the proposals in a word like format. Our Job as the Backend team is to handle the storage of the data from the front-end for quick storage and retrieval. The front-end team chose to use React JavaScript Facebook’s front-end language. The Backend-Team was charged with creating a REST interface for interaction with the Database. A REST interface was chosen primarily for the fact that it is easy to modify and different front-ends can quickly be swapped out as the interface is not tightly coupled. A REST interface has “endpoints” for data each endpoint does one of four operations: Create, Update, Delete, Read. A front-end only needs to hit one of these endpoints to transact data which are created within the Controllers of the REST client. Models within the REST Client organize the data that will be stored. We used Entity Framework in order to create Models that can quickly create tables mapped to them in the Backend.

We created several Entity Relationship Diagrams over the course of each iteration until we finally had a final design to store the data in. Each iteration we added more Model/Controllers and revised our ERD to reflect how we changed the storage and REST endpoints of our data. Every part of a proposal is broken down into a Section or a Subsection. Each Section can contain none or many Subsections. A Subsection cannot exist without a Section. Each Section and Subsection contains Text, and a relative ordering to the proposal as a whole. Each Section or Subsection Revision is stored as a collection entity under Revisions allowing for one to roll back revisions if necessary. Sections and Subsections are governed by a Template that keeps track of data such as line spacing it is related to another Font entity that stores the currently used Font.

In later iterations it was necessary to create some kind of authentication for login. We used Owin which is an authentication process created for .net. Specifically, it used as an algorithm to create a Token for users that send a properly registered username and password. The token passed back can be used to then access protected content such as the RestEnd points associated with proposals. User registration was also created under unprotected content, of course, and user look up by id was added as well. The authorization server for Owin had to be properly configured and the correct data had to be passed for the token to be properly created.

First, I will talk about what I created in this project I am only going to reference only things in the final code for brevity. First, I made all the Models and all the Controllers. Commenting was done by others and a Logger was added to them to log actions everything else within the code I did. I also created all the Unit Tests used in the final code. Integration Tests were not done by me. I configured our Owin server to properly take user credentials and pass back a token for front-end to be able to access protected areas. I also did the account creation. I created a total of 8 controllers that dealt only with data . I also created 8 models to go with those controllers with the final code in this [ticket](https://svn.cs.kent.edu/S2016/trac/ticket/54). Earlier I explained the functionality of the models and controllers I created. Next, I created the unit tests each test tests the 5 endpoints (Create, Read, Update, Delete) of each controller. They do so without accessing a database and mocking all the data. I created an in memory ADT(Abstract Data Type) that simulates the DB and mocked the WorkflowRestfulContext (File which configures the DB) as TestworkflowRestfulContext. In all I created 40 unit tests each testing the CRUD end points and testing to ensure the endpoints fails and passes when appropriate using assertions. The unit tests use the AAA method Arrange, Act, Assert. Go [here](https://svn.cs.kent.edu/S2016/trac/ticket/177#ticket) to see the appropriate ticket referencing Final Demo code. The last section I worked on was using OWIN oauth authentication. OWIN takes 3 key elements and uses HMAC encryption to create tokens for accessing protected content. First I created the controllers with the necessary endpoints for accounts. First I created an accounts endpoint for account creation and looking up users the endpoints were under the accounts controller. Next I created an AudienceController which is an endpoint that randomizes audienceIds which is one of 2 necessary components for creating the token that will be put in the HTML header for access to protected areas. I configured a port number at the endpoint Owin/oauth to run on and configured it to create JSON tokens returned for the requested endpoint using [​CustomJWTFormat](https://svn.cs.kent.edu/S2016/svn/S2016/WorkFlow/WorkFlow/BackEnd/FinalDemo/WorkflowRestful/WorkflowRestful/Providers/CustomJWTFormat.cs,) and CustomOauthProvider. To see all the code referenced in the final project go [here](https://svn.cs.kent.edu/S2016/trac/ticket/120). Front-End had some issues with connecting to the Owin/oauth server so I created a new endpoint. I created a new endpoint under /api/accounts/login that basically calls the owin/oauth server returns the token and automatically places the token into the header for the front-end at the endpoint but they were still experiencing issues mostly related to cors. Which is a problem caused by the fact that our 2 solutions are separate and not under the same project.

Next, I will talk about what other group members did. First Mark Bank did a lot early on with documentation and corrected some Cors errors, however, he got sick and we didn’t see him again after iteration 2. I will exclude him for the simple fact that he got sick. Eric Pisac created an ERD to expand upon the ERD I did and did some work for iteration 2 of the unit tests. He created a method that tested using the DB he wrote to the DB and then deleted it. Around the same time I released my Unit Tests part 3 ticket which did not use a DB and mocked data. We used this code rather than having Eric continuing creating his unit tests that write and delete directly to the database. I give him a 10 percent contribution. Next, is Kristin Harris. She created a piece of documentation for what a RESTful client is early on. She also did the ARC (Advance Rest Client Tests) these tests were documented and used in the final iteration demo to show our code worked. ARC simulates the frontend hitting our rest endpoints and through these we can ensure the correct data is being returned/edited/created. Some bugs were found using these tests which I fixed by editing the Models/Controllers that I created. In the final Iteration Kristin created a Logger which is used in the final demo code and logs warnings/errors/actions to .txt file. This will be handy when multiple people are using our program and they want to find errors. Kristin gets a 20% contribution. Peter Tennku Created 2 ERDs and our final iteration ERD that will be used in our final Demo. An ERD (Entity Relationship Diagram) pictorially shows all the relationships between the tables stored in our Database. He also created Integration Tests which tests multiple pieces of our system vs Unit tests which I created and only test one piece of the system at a time. For his integration tests he created a mock database and also tested the login features for proper functionality, however, the integration tests were not finished and were cut from the final demo. He also did most of the documentation of the things I created such as the Owin Login process and the documentation to how RESTful client works as a whole while I did the code.

Peter gets a 25% contribution. I give myself about 40% contribution as I wrote most of the required functionality code wise in our RESTful client, all authentication code, and all the unit tests.