Software Engineering (423): Senior Capstone Portfolio: Spring 2021

Montana State University: Computer Science Department
Johnny Egan | Wes Robbins | Spencer Reap

Section 1: Program

For our Software Engineering Senior Capstone our class was approached by a local Bozeman startup, SpecialProject. In their own words SpecailProject offers a platform that is, "helping video creators and independent studios earn subscription revenue and reach audiences worldwide." We were asked to build a commenting API. The API allows the user to access functions such as comment, upvote, downvote, and reply to a comment on a SpecialProject post. The API populates a live database hosted by the school server which is then accessed by the SpecialProject team and implemented into their interface.

- SpecialProject Commenting API
- API Source Code
-

Section 2: Teamwork

We have used the agile development framework in order to facilitate collaboration throughout the project. This has included meetings three times a week. Once every two weeks we have a sprint planning meeting and sprint review and the rest of the meetings are short scrum meetings. Using this framework has allowed us to track our progress and easily delegate tasks to the members of our group.

We have also used several tools that have enabled teamwork. Foremost, we have used GitHub to host our version control and track and update issues with the API. This has allowed us to easily integrate individual contributions to the codebase of our API. Each team member has their own branch as well as one collaborative developer as well as an additional main branch for final versions. We have also used Trello in order to keep track of designated tasks. Trello has also been helpful for tracking progress made during each sprint and keeping track of items on the backlog. Lastly we have used group messaging in order to quickly and easily communicate between group members.

Team Members:

Team member 1:

This team member worked on setting up the live server via the course website and connected SQL database also served by MSU. Team member 1 also worked on the functionality to add, edit, and delete comments to the databases as well as how they can be displayed on the website. Team member 1 also implemented the upvote/downvote functionality. This team member also tested integrating the API into programs coded in several different languages including Python, C++, and Java.

Team member 2:

Team member 2 for the first spring worked with team member 3 to help design the UML diagram to give an outline of how the API will interact with our database, as well as how the comments will be structured within the database. For the second sprint, helping design the edit/delete functions, updating the routes, and controller files which were implemented by team member 1. Team member 2 also focussed on the *Developer* documentation page for the current site, with some minor formatting.

Team member 3:

For the first sprint team member 3 worked with team member 2 to build the UML diagram to help organize and plan the structure of the API. The UML planning included how the API would interact with the database as well as what functions would be present in the API. Team member 3 primarily focused their time on the CSS and HTML formatting of the API. The formatting was done separately and then integrated into the working framework after sprint 2. Team member 3 also played a role in editing the database to give it a more appealing and organized layout. Team member 3 also played a key role in creating and conducting the user testing for the front end and back end API.

Section 3: Design Pattern

For the commenting function we implemented an Observer pattern. The user would be considered the subject would be the creator of the post, with each commenter playing the role of the observer, changing the state of the post by adding, deleting, and editing their comments, as well as, up/downvoting the comments. Once the interaction has been submitted, the database is then updated and the change in state is recorded.

Additionally, we used the Model View Controller(MVC) design pattern. This design pattern is baked into the *Adonis* framework, so it was required that we used MVC when working with Adonis. The 'Model' in our software was the direct interface between our program and the SQL database. The 'model' would make queries or post new data to the database when requested. Changes in the 'Model' would result in an update to the 'View' which is the user-facing part of the software. For example, when a new comment is added, our 'display database' page is updated to show this new comment. The 'Controller' in our application was what was responsible for the logic in our application. There were two primary functionalities of the 'Controller' in our software. The first was to take input from the user and then build a request for the model. The second was to sort information that was returned from the model. For example, one of the functionalities of our software was to return comments that were sorted in whatever way the user specified. Not all sorting methods could be done directly with a database query so we created a method in the controller to sort comments before returning json data to the user.

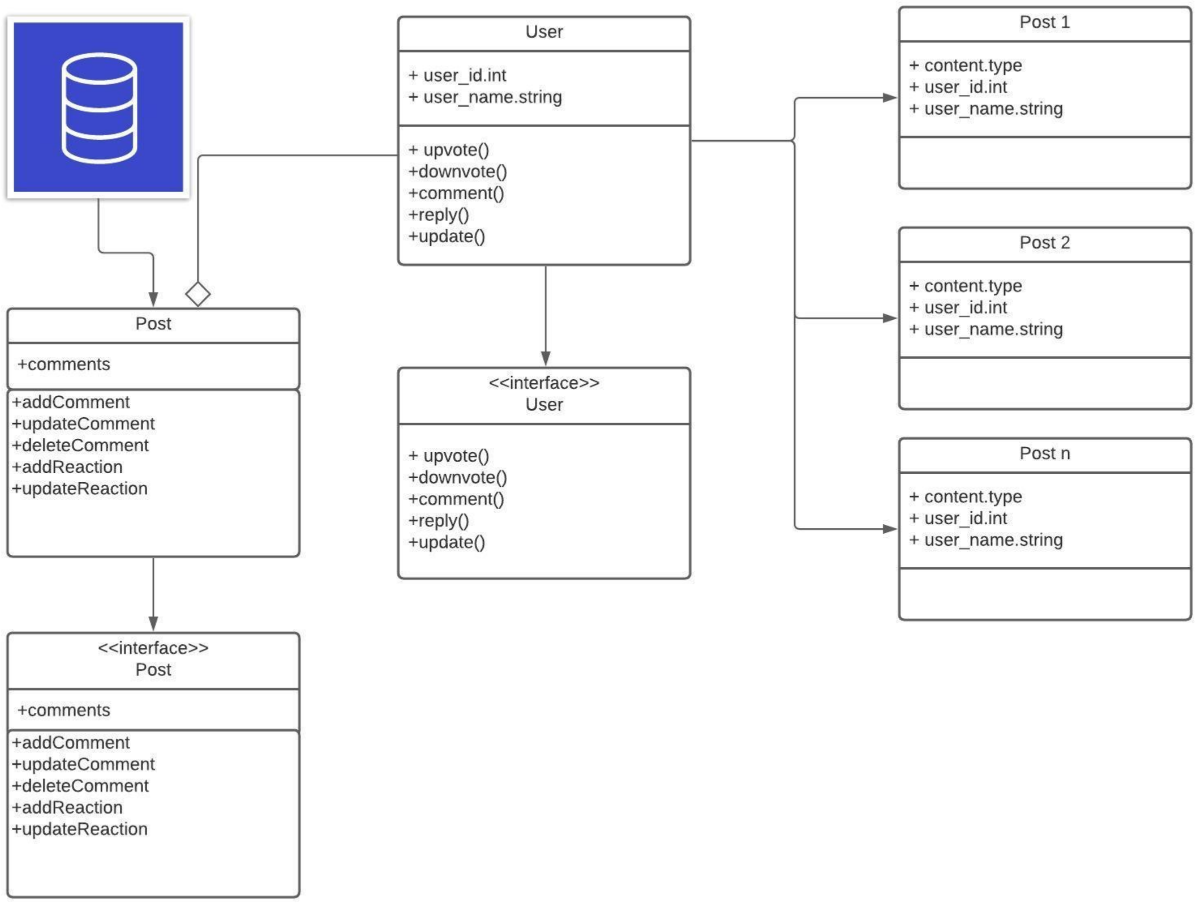
Section 4: Technical Writing

- Developer Documentation
- User Documentation
- User Testing

Section 5: UML

UML Design Diagrams:

First Draft UML



Section 6: Design Trade-Offs

As the special team requested, ideally this comment structure would have a feel to the Reddit commenting function. One issue with adding an infinite number of possible replies is that the database could easily become very large, and thus slowing down the webpage depending on how many comments have been added. Another issue that’s very similar is an issue of keeping track of up/down votes as the number of comments, and their threads grow.

Another design trade-off that was discussed was whether we should focus more on creating a REST API or software that also had a fully implemented front-end. The advantage of just creating an API was that we could move on the features and functionality of the API. The advantage of doing additionally doing a front-end is that we would have got to better show off a good use case for the API. After discussing it within our group we decided to focus on creating an API. However, we did still create a basic front end to show that the API works. We were also able to integrate the documentation into our API.

Section 7: Software Development Life Cycle Model

We chose to use the Agile model to develop our Capstone Project. The iterative aspect of the model seemed appealing and useful for this project due to the fact that we would be working in two week sprints. Every two weeks we conducted similar steps on a similar schedule as prior sprints. Our typical sprint included planning, designing, developing, and finally implementing.

One advantage of the Agile method that we noticed early was that some features would be available within the first or second sprint. This assisted us while creating the undeveloped features because we had start to finish experience in doing so already. By having features implemented early we were able to see what was and wasn't working. This was helpful in preventing major refactoring later on in the semester. This model also allowed us to capitalize our time usage very efficiently due to the fast pace.

However, there were some disadvantages of the Agile method. The first being that important documentation was sidelined unintentionally for the first sprint due to our focus on the first release. This was only a minor disadvantage. A more significant disadvantage of the Agile method was the decreased amount of time to communicate with the SpecialProject team. With the fast paced two week sprints there was typically only time for one formal meeting per sprint.

Sources:

- https://www.specialproject.io/
- https://adonisjs.com
- https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design