Coding Careers

Project Part 4: Final Report

Taylor Andrews, Ian Char, Lauren Mitchell, Alan Moy, Peilun Zhang

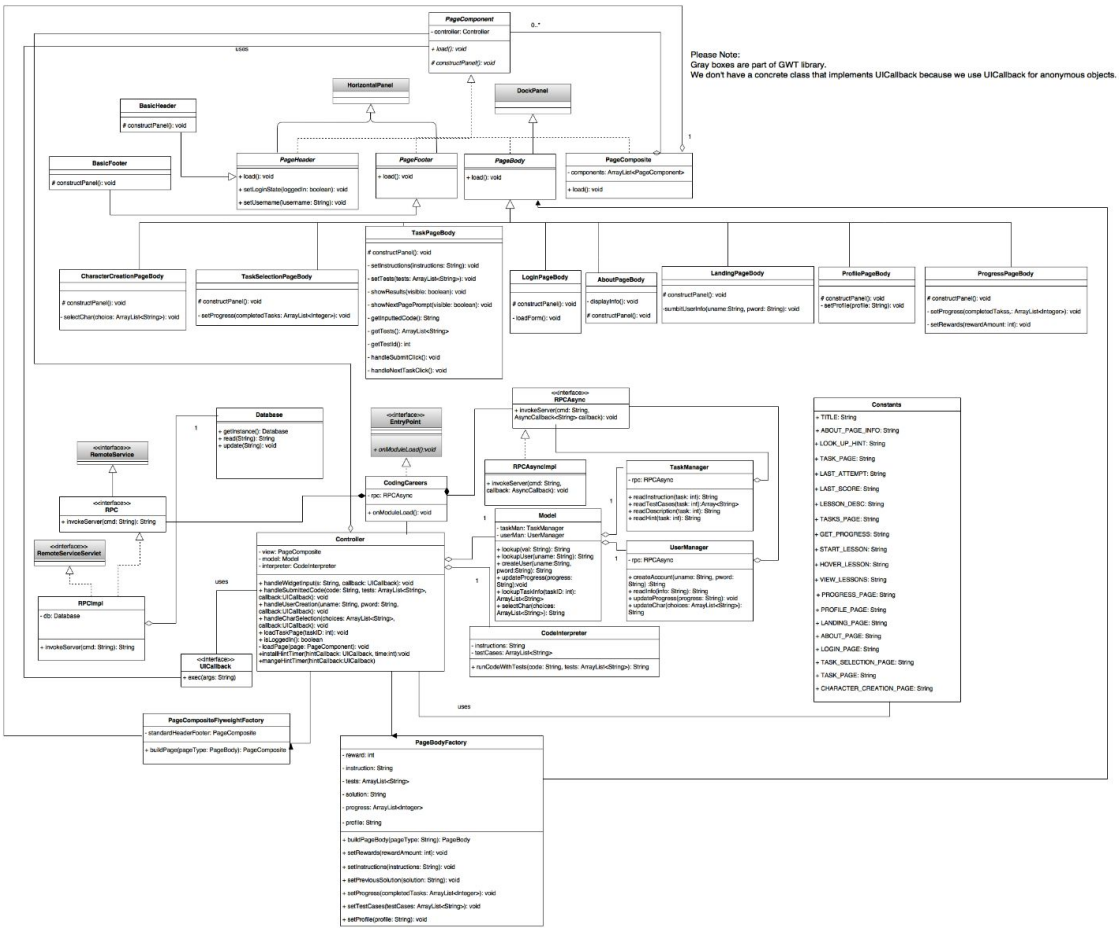
1. What features were implemented?

We did user login (U-01), code submission (U-02), lesson selection (U-03), user logout (U-04), code input (U-05), the about page (U-06), account creation (U-07), continuing to next task (U-08), code reset (U-10), and viewing user progress (U-12).

2. Which features were not implemented from Part 2?

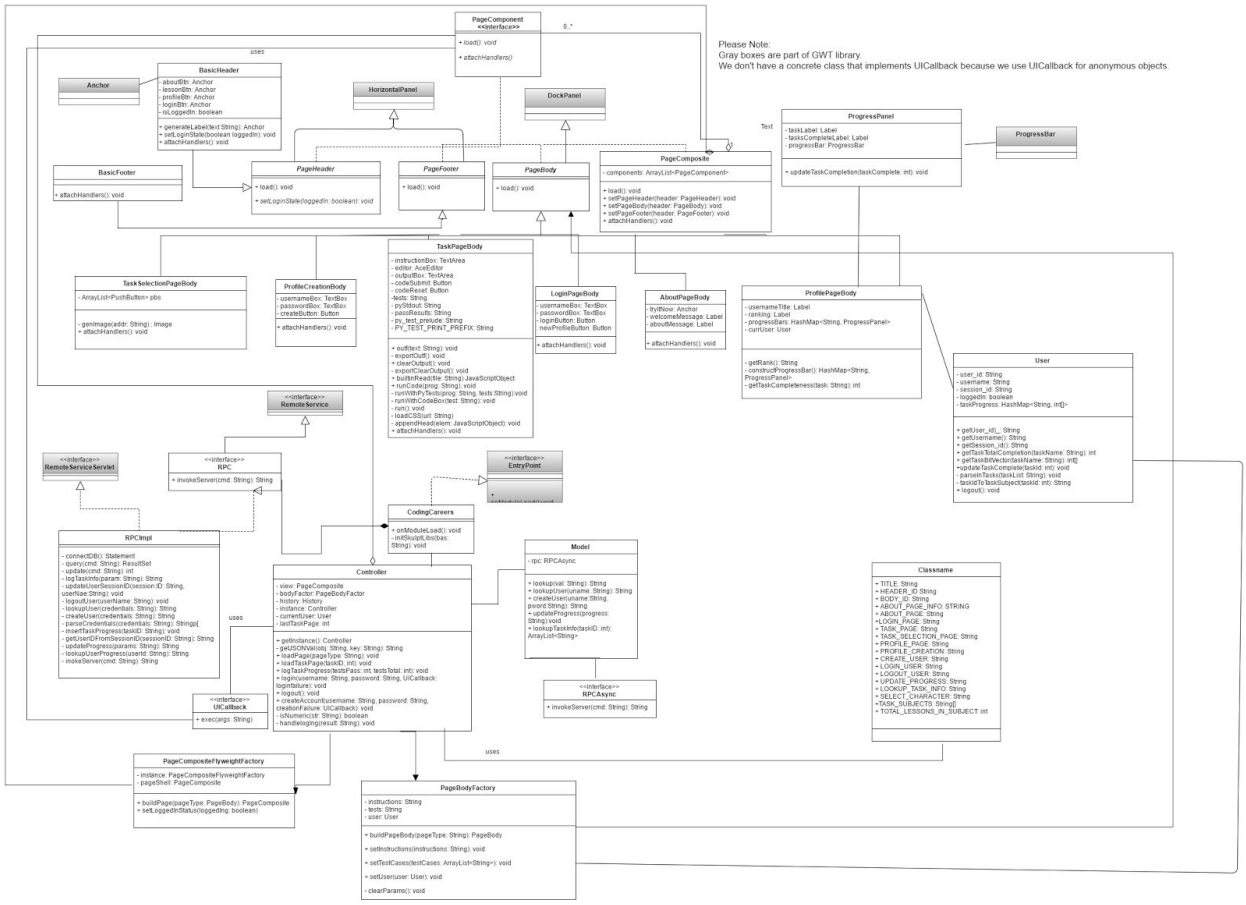
We did not do character selection (U-09), and giving the user hints (U-11).

3. Show your Part 2 class diagram and your final class diagram. What changed? Why? If it did not change much, then discuss how doing the design up front helped in the development.



Original Class Diagram

https://drive.google.com/file/d/0B-WAj1ufusdOMXA2Mkh3QjRhajQ/view?usp=sharing



Final Class Diagram

https://drive.google.com/file/d/0B-WAj1ufusdOc3NRWTNJQmg4ZDg/view?usp=sharing

Our class diagram did not change much. Most of the changes that did occur were removing classes corresponding to features of our program that did not get implemented. This was also true for some methods which were redundant, or had easier means of implementation. The other reason our class diagram changed was some of our classes actually made things more complicated. They were “middle men” and needed to be removed in order to reduce bulkiness in our code.

That being said some classes were actually added to our system. For the most part they were added were added because they gave greater data encapsulation. For instance, there were elements of the page that were repeated and held their own information, such as progress bars that we broke off into separate classes.

Planning the design out before developing was helpful because we didn’t end up with a mass of spaghetti code. It also meant that when one person in our group was implementing part of our code, everyone else knew exactly what they were talking about. There was no confusion and it was very easy to coordinate tasks.

4. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

Yes, we made use of factory, singleton, composite, and flyweight design patterns. The singleton pattern was used for our database. We thought this pattern was valuable since we only wanted one database instance, and the option of having synchronicity ensures users can be using Coding Careers on multiple platforms at once.

The composite design pattern was used for our webpages. Because Google Web Toolkit (GWT) only uses one HTML page, every time the user needs to go to a new page we must load in the new header, body, and footer of the page. Rather than load each of these components separately, we can wrap them all into a composite class that will call each load method on the components when we want to load the page. Thus, each page is a composite of a header, footer, and body. This allowed us to change parts of the webpage while keeping the overall structure constant and made our code much more organized.

We used a factory method for creating page bodies. We leveraged this by having all of our page bodies extend one abstract class PageBody. By identifying the type of page body we want, the factory will return a PageBody that suited our specifications, allowing us to load different page bodies dynamically. This was helpful because it allowed us to easily add and remove the different pages of our website, like task selection, without having to change other aspects of our code.

The flyweight pattern was used in combination with the factory pattern for creating page bodies. This flyweight factory was used to do the actual building of our webpages via construction of aforementioned page composites. Overall, this was a benefit to our system because most pages reused the same page header and page footer. Therefore, these aspects of the composite could be cashed and we could switch out the page body.

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

Overall this project taught us that analysis and design make implementing our ideas smoother, and our final product much more future proof. While creating Coding Careers we learned the importance of intelligently reviewing our system requirements for validity and clarity. We realized when requirements are murky the product is equally murky, and it becomes difficult to assess what the client wants. In addition, our requirements need to be concise, because otherwise we can’t verify if Coding Careers

is complete. Looking at things from the user's perspective, as was the case while establishing use cases, taught us how to identify the important parts of our system. We were also able to scope our project more effectively.

As we began to design Coding Careers we learned design and design analysis is just as important as requirements and requirements analysis. Making our software flexible was a priority since the project might be expanded on in the future. Our initial plans didn't take into account the open-closed principle, but we realized using polymorphism we could allow for more lessons and features to be easily added to Coding Careers. As our project expanded we also learned the value of good abstractions. For example, having an abstract *PageBody* class let us easily add more types of pages to our website.

A critical aspect of our design process was learning different patterns and assessing if they should be applied to our project. Though we wanted to avoid over-use of design patterns, like Singleton, we needed to make sure we obeyed SOLID principles and adhered to high cohesion/low coupling. One learning experience was our constant re-use of page elements, like headers and footers, leant itself well to a flyweight factory rather than re-creating elements each time.

The coding part of Coding Careers was actually accomplished fairly easily. Our class diagram was comprehensive, and already mapped out the structure our implementation should take. If we hadn't planned so extensively it is likely our coding process would've been a lot less efficient.