CSCI 4448
Liz Boese
4/27/16

Project Part 4 - CoCal

By
Adrian Chen, Justin McBride,
Justin Schiller, Warren Ferrell, and Zach Lamb

1. What features were implemented?

Users can create a new account or login to see their own personal calendar. Users can create new events, join and create groups, and find nearby events. Groups have moderator functionality which allows certain members to have more control over their groups, such as accepting and rejecting membership and monitoring the group's private calendar. Users can also edit/remove events they have created in addition to categorizing their events by cost, type, and privacy (public, group, or private). All users can report any event for inappropriate content in which case the administrators are notified to handle the situation accordingly.

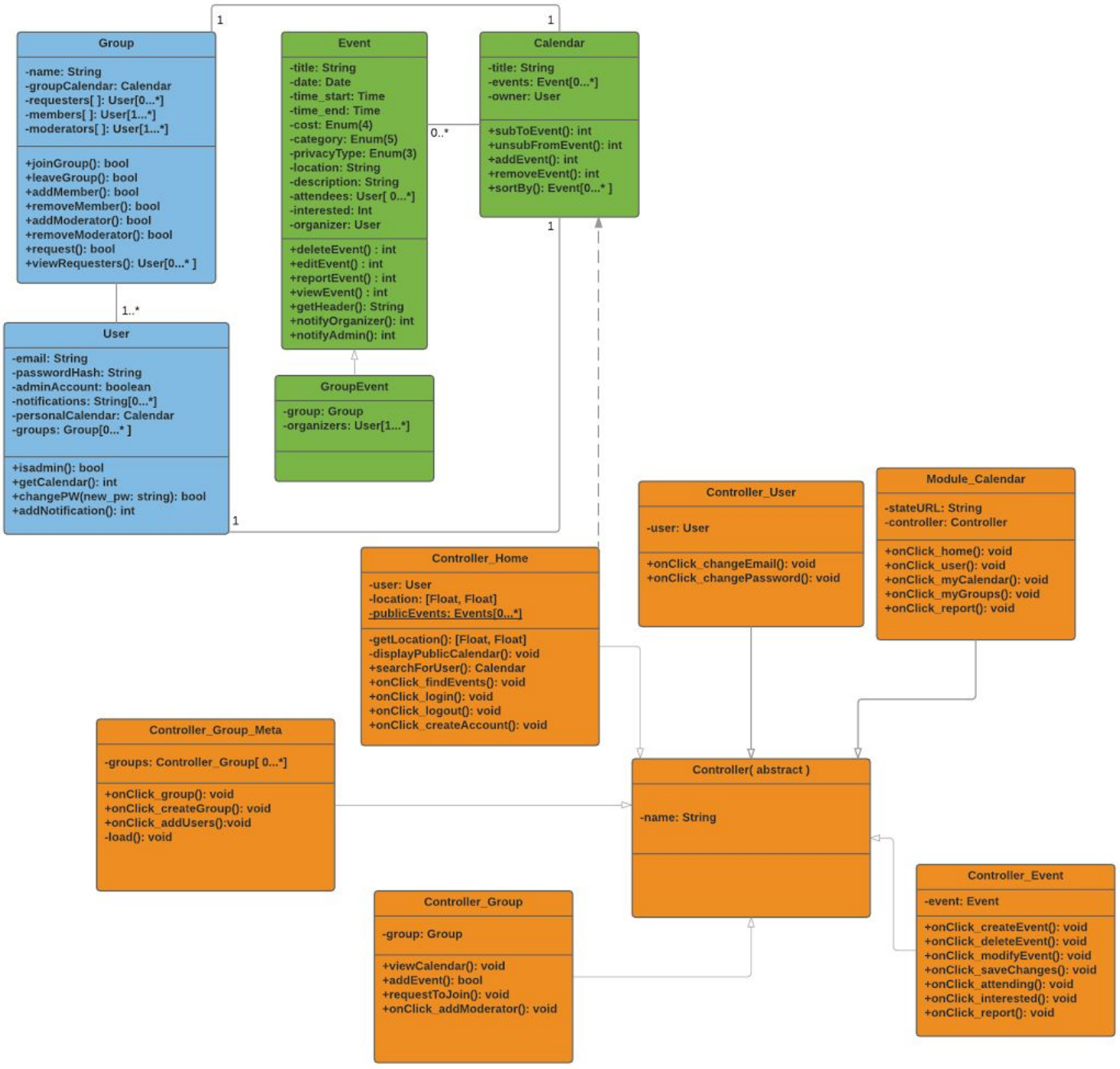
2. Which features were not implemented from Part 2?

Email is not validated, as we decided to validate name and password instead. We also did not implement password recovery as a result of this change (without a validated email it is very difficult to reset a password). We decided to change from an email notification system for admins to become aware of reported events to use an application-hosted notification system that could also be used to notify users if their events had been modified or deleted.

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.

Features that did not get implemented were not included in our final class diagram for clarity. Our first class diagram was limited by our understanding of Angular and nodeJS at the time. The MEAN stack makes extensive use of the MVC design pattern and while our original class diagram matched the controller portion of our application decently well, we did not include the model layer or view layer (which in the final product consists of html pages). To maintain our original class structure, we would have needed to use an object-relational mapping database in place of our original database. The final class diagram is significantly different from the original because we decided to go ahead and use this popular application full-stack instead of forcing our software to conform to the original diagram. As a result of this, we were able to learn many design patterns inherent to MEAN. Even though our final diagram does not mirror the first, it has extensive similarities and the first helped guide the changes that ultimately resulted in our current class state.

COCAL CSCI 4448 PART 2 CLASS DIAGRAM



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?

At a higher level we used the MVC architecture pattern which is what the MEAN stack is built around. This allowed us to separate the components from each other by making use of the MVC design patterns (observer, strategy, and composite). We also used the factory design pattern to create objects for user registration and authentication. We could have better used design patterns, however. For example, we could have used template to create skeletons for much of the functionality that is similar throughout the application, such as notifications. We also could have used a factory to create our events, as they have many different types and categories that would predispose them to being polymorphist.

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?

Analysis and Design has shown us how to visually see how software works and how we can better organize it. This also helps us work smarter not harder because of the possibility of code reuse, easier to maintain systems ,and possibly debug code faster. We also learned that there is value in diagramming a system before implementing it in spite of the fact that our final product did not match up precisely with our initial specifications. The design process allows developers to see the interactions between different parts of the system before writing any of the code which helps to smoothly connect the different components. I.e. if you have a user class and a calendar class, data structures can be embedded into both classes that would otherwise be unnecessary in order to facilitate communication between them. We also made heavy use of refactoring throughout the course of our project. While we set out to use object oriented principles in our code base, we did not always implement them in the most efficient way (we

created a blob) and we had to refactor to more streamlined, efficient files in order to make adding and modifying files easier.