Name of Project: CU alpine club website
Group Members: Will Culkin, Max Schwarz, and Dmitri Tarasov

Final State of the System Statement:

The features that were implemented into the system include a login page where users can log in to their account. It also includes a registration page where users can create a new account. Users can also browse their profile page once they are logged in. A discussion board was implemented that allowed users to interact with each other by posting comments to a forum. There was a photo catalogue that was implemented that allowed users to select photo catalogues based on certain criteria, we used different years for this. It however was not fully fleshed out and users can't currently add photos to it as we did not connect it to the backend. This was done because storing photos was more complicated than expected and we wanted to focus on other aspects of the project. We did implement a way for users to sign up for trips. Depending the status of their log in users will see one of three screens. The view only, signup, and the create trip pages. The website forces users to sign in before signing up for a trip and it also lets special users sign up for trips and create new trips. When a trip was created, the type of trip was recorded to give users that wanted to sign up for the trip a fill-out form related to the trip type. This is working except for adding photos to new trips that are created. We ended up not doing anything for the home page as it was just cosmetic and not the focus for this project. The project also included a dynamic navigation bar. This was used to help user navigation throughout the website, but also to implement the state pattern on the frontend. Instead of having a navigation bar that brought users to the same trip page and then having if else statements to check for login status, and having to check in every page, the navigation bar handled this. In the one place it checked login status and based on the login status it would display different links. This meant that when a user was logged in it would send them to the view and sign up for a trip page. This worked well because we didn't need to check the status of the system in every component and we could reuse components however we wanted and it was all in one place. For example when making the create trip page we could reuse the view component because special users can still sign up for other trips, but were able to add a create trip section. We also did not get around to implementing a system for users to pay for memberships primarily due to limits in time and since we didn't get it running outside of local it didn't seem prudent.

Future Developments:

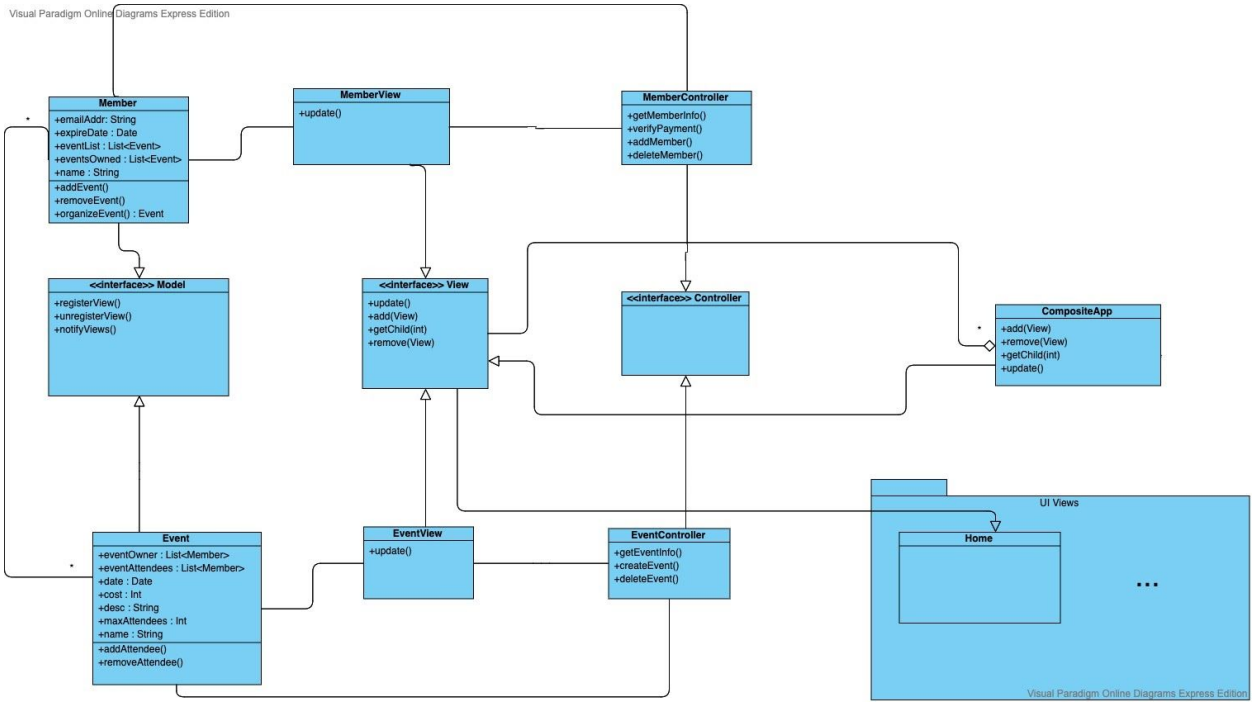
With more time one thing to add would have been pagination so that not every document is pulled out of the db at a fetch request but only the most recent ones.

Another would be to have the entire CRUD functionality for each page as most of the pages are just create and get, with only the trips backend having put functionality to edit which users are on the trip.We would also like to refactor some of the pipeline between client and server in order to be able to dockerize the components so that the system components can communicate through different links. A use case for this project that didn't get implemented was to purchase memberships through the app. This is left to future work. Authentication also did not get finalized, as we would like to use a third party such app such as Okta.

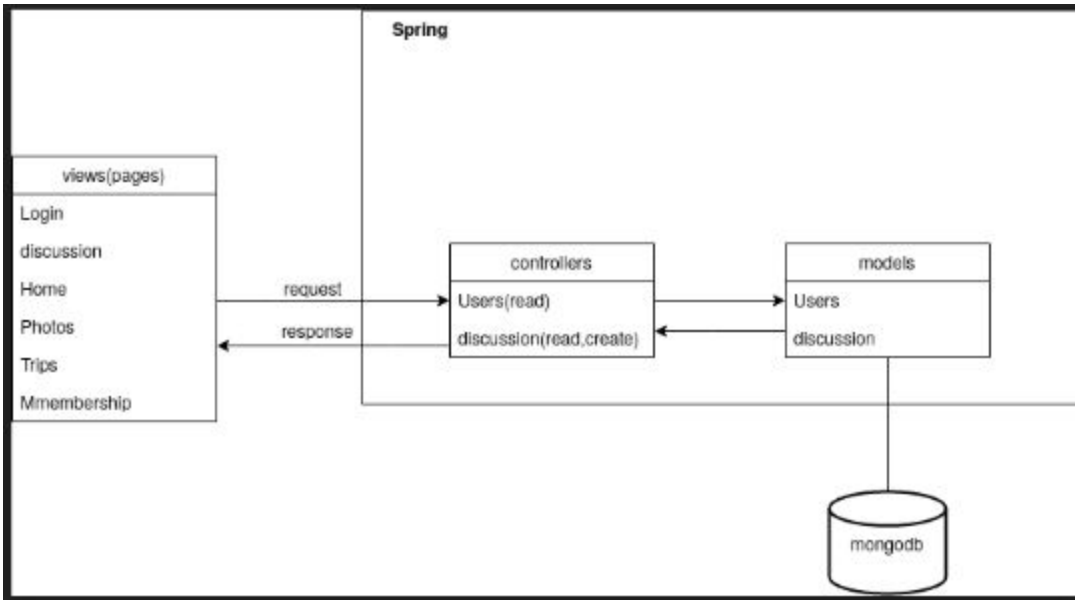
Differences from the Initial Plan:

The architectural diagram described the primary architecture rather well because the individual pages had their own link to a concrete mongodb collection. Thus each page escentially had its own controllers for the CRUD functionality which accesses one or many of the tables to compile the response. The primary difference is with the class diagram given how Spring's framework basically leads to having a separate model and controller for each collection for the database. Thus the database requirements had a heavy role to play in the final design as the database design was based on individual entities like a single user, trip, or post. Another difference is the addition of the discussion forum, which did not exist in our earlier designs. Various member variables also changed or were added as the need arose, such as users objects having a password.

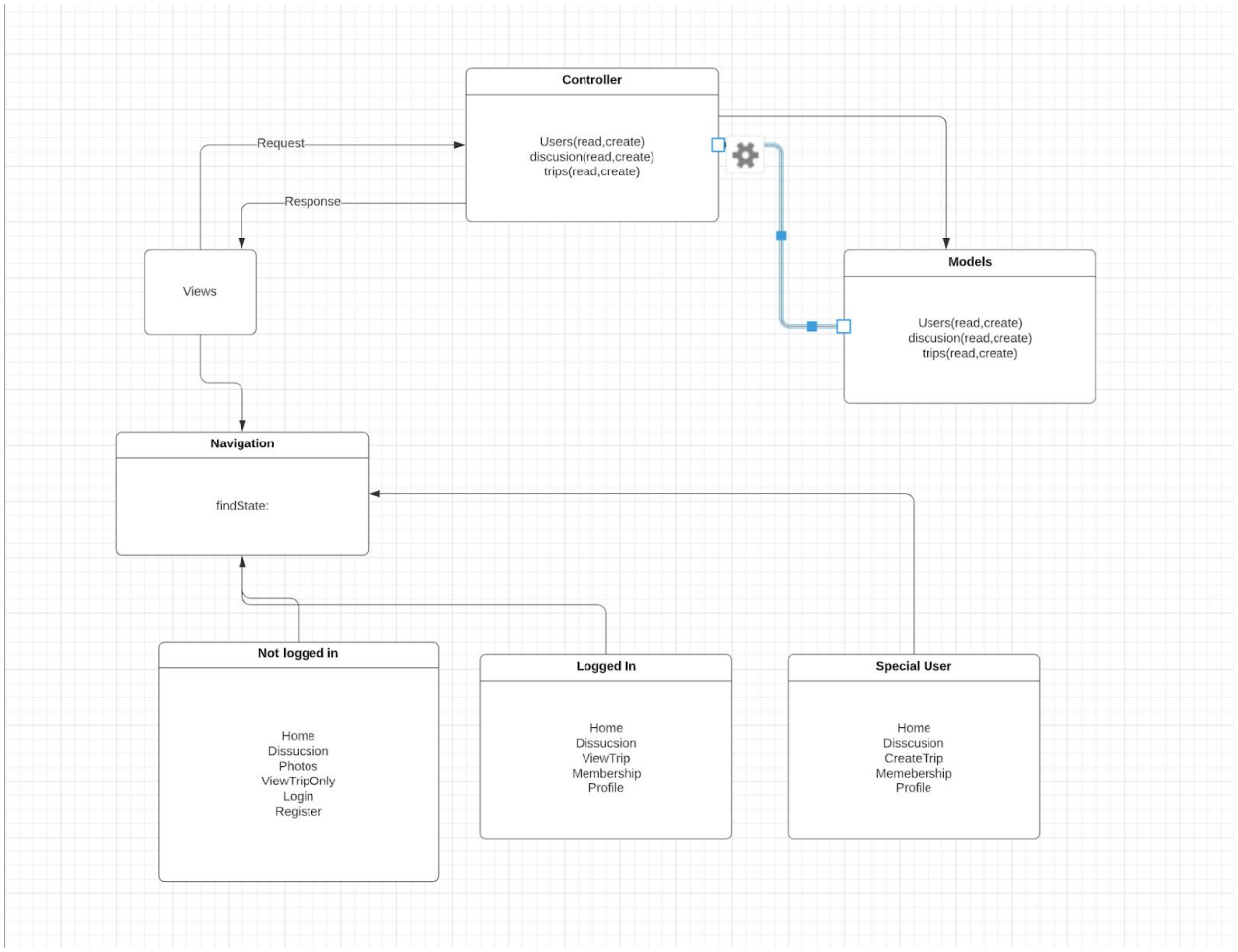
Final Class Diagram:
Planned diagram/project 4


Visual Paradigm Online Diagrams Express Edition
Member
+emailAddr: String
+expireDate : Date
+eventList : List<Event>
+eventsOwned : List<Event>
+name : String
+addEvent()
+removeEvent()
+organizeEvent() : Event
MemberView
+update()
MemberController
+getMemberInfo()
+verifyPayment()
+addMember()
+deleteMember()
<<interface>> Model
+registerView()
+unregisterView()
+notifyViews()
<<interface>> View
+update()
+add(View)
+getChild(int)
+remove(View)
<<interface>> Controller
CompositeApp
+add(View)
+remove(View)
+getChild(int)
+update()
UI Views
Home
...
Event
+eventOwner : List<Member>
+eventAttendees : List<Member>
+date : Date
+cost : Int
+desc : String
+maxAttendees : Int
+name : String
+addAttendee()
+removeAttendee()
EventView
+update()
EventController
+getEventInfo()
+createEvent()
+deleteEvent()
Visual Paradigm Online Diagrams Express Edition

Project 5:


Spring
views(pages)
Login
discussion
Home
Photos
Trips
Mmembership
request
response
controllers
Users(read)
discussion(read,create)
models
Users
discussion
mongodb

Project 6:



The key relationships of the diagram are the view controller and model. They each do their parts individually. The view is made up of different pages that a user might see. Behind the scenes it displays the correct views by checking the login status and then dynamically chooses a set of links that direct the user to the correct page that they should go to based on login status. The controller gets information from the views when a user submits a login and sends it to the database to process and gets the info it needs from the database and then sends it back to the view.

The patterns that were useds include the MVC which can be seen As the views, controller, and model.
Another pattern that was used in the views was a state pattern. Instead of checking the navigation going to a trip page and the trip page deciding what to display based on the login status the navigation has decides what state it is in and then provides the correct links. This means that when a user logs in the trips link is still displayed but it will go to /viewtrips instead of /viewtripsPleaselogin and these different pages are different

components that can do different things or can be combinations of each other. This isn't exactly the standard java implementation, but it does allow a single source of truth in the sense that every view won't have to be changed when a new user is added, just the new views for the new type of user. This follows the idea of the state pattern.

The main difference between project 5 and six is that trips were added everywhere, that the photos view was implemented, user login and registration was updated, and a profile page was added. . In the project 5 submission discussion was the only working element and in project 6 trips and photos were added. Photos only to the view. Also login, registration, and profile were fully implemented.
Compared to project 4 project 6 is missing membership and photos are not connected to the db.

Third-Party code vs. Original code Statement:

We used a lot of sources as we were all learning new technologies. Some of the logic used is from sources that help with developing with React and Spring, although no code in our project is taken directly from any sources.

Frontend:

- https://www.youtube.com/watch?v=97A3uRS7_YA&t=727s

This was used to help create a navigation bar. It was changed to use a different router, and display different pages. The styling remanied.

- https://reactjs.org/

The documentation of reactjs wa used a lot for many basic elements elements like how to create a form in react

- https://stackoverflow.com/questions/11078913/how-to-set-max-width-of-an-image-in-css

  Stack overflow was also used to help with bugs and various things like getting images to be at max a certain size, don't have all of the links but was only used for small bugs

References:

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
https://spring.io/guides/gs/accessing-mongodb-data-rest/
https://dev.to/bouhm/thinking-in-react-component-composition-fp5
https://docs.spring.io/spring-data/mongodb/docs/1.2.0.RELEASE/reference/html/mongo.repositories.html
https://mkyong.com/mongodb/spring-data-mongodb-insert-document/
https://reactjs.org/docs/handling-events.html
https://github.com/dylanlrrb/Please-Contain-Yourself

https://medium.com/@gtommee97/rest-authentication-with-spring-security-and-mongodb-28c06da25fb1
https://stackoverflow.com/questions/40366958/creating-mongodb-database-user-in-java-using-spring-data-mongodb
https://stackoverflow.com/questions/30228672/cannot-authenticate-user-in-mongodb-3-0-2-using-java-connection
https://github.com/Hygieia/Hygieia/issues/999
https://stackoverflow.com/questions/49475177/failed-to-auto-configure-a-datasource-spring-datasource-url-is-not-specified
https://stackoverflow.com/questions/40366958/creating-mongodb-database-user-in-java-using-spring-data-mongodb
https://www.petrikainulainen.net/programming/spring-framework/creating-a-rest-api-with-spring-boot-and-mongodb/
https://stackoverflow.com/questions/32538123/spring-boot-and-mongodb-configuration
https://stackoverflow.com/questions/23619018/mongodb-not-authorized-for-query-code-13/31487774
https://medium.com/mongoaudit/how-to-enable-authentication-on-mongodb-b9e8a924efac
https://www.youtube.com/watch?v=97A3uRS7_YA&t=727s
https://reactjs.org/
https://mkyong.com/mongodb/spring-data-mongodb-update-document/
https://stackoverflow.com/questions/43773822/why-postman-not-send-data
https://stackoverflow.com/questions/42173127/mongodb-i-want-to-store-array-object-in-collection
https://developer.okta.com/code/java/
https://mkyong.com/java8/java-8-streams-map-examples/

Statement on the OOAD process for your overall Semester Project:

One of the key design elements of our project was the use of state for user login status. This was overall a positive because it made it easy to display different things to different users without having massive if/else statements in every page to render the correct things. It was also nice because once the navigation was working the individual pages could work without having to check for user state and changes were easier to make. This did however couple the pages to the navigation because they relied on navigation to have the user state correct and they don't have many checks to see if the user is actually logged in. This could be fixed by adding a check for the state an element expects. It was also not great because the frontend was started before a coherent plan

for handling state was thought of so not every element was made with this idea, however this could be remedied and maintained going forward.

Another key design was the choice to use a MVC. This worked great, which is unsurprising since it is such a common pattern to use in website design. It allowed every person to focus on a specific part. Will took the view, Dmitri primarily did the controller, and Max primarily did the model. We created templates for how information would be passed between, i.e. there would be usernames. Then using the pattern Will was able to work on the views without having to worry about the how information was being handled between the db and react just that it would get certain information.

A negative for our design was that our scope of project was rather large and it would have been better to focus on fewer elements and really make them work well. Given the limited web development knowledge it was hard to get a good grasp of what was a good goal for the amount of work. This in addition with going remote and dealing with the new situation caused it hard to complete everything that we set out to do in project 4, however we got a lot done and it will be easy to continue fleshing out the project to be usable in if desired because of the strong OOAD elements that were implemented.