Project Part 4: Final Report

1. What features were implemented?

We implemented:

- Mentor display: On the main page, every mentor in the database shows up, with basic information displayed (This is the key information we learned we needed to display from talking to customers).
- UC-010: Mentor Search: You can search for mentors by name or keyword (marketing), and results will show up on main page.
- UC-009: Searching for which mentors have mentored a specific company.
- UC-017: You can search for a mentor then add feedback to a mentor profile (1-5 stars, and comments).
- UC-009: Searching for which mentors have mentored a specific company. Though mentee (company) profiles were not implemented, a user can still add them to the mentor's profile page, and then search for that company as a keyword
- FR-004: A mentor's mentoring history stored: though it was not as thorough as we hoped, a user can add a company that a mentor has mentored as a list, and is searchable, which accomplishes the user goal of saving time by finding mentors who have mentored similar companies in the past, as a basis to inform future mentor-mentee matching.
- UR-003: Admin can edit usage rights of other (less senior) admin
- UC-011: Manually input mentor data. Users can manually create new mentor profiles and import their information. You can submit information about a mentor, and it will save it as a mentor profile in our database
- UC-007: User can log out
- UC-008: Admin can add another admin.
- UC-015: Searching for a mentor with a specific skillset. Because user can manually add skills, they are searchable in the search bar.
- UC-019 (partial): Logging in- User can log in. Partial implementation, currently experiencing role issues. Users cannot authenticate with the framework.
- UC-016: Adding Bio to mentor profile- User can add bio information manually to a mentor's profile.
- UC-018 (partial): You can add skills to mentor, **but you cannot edit skills**

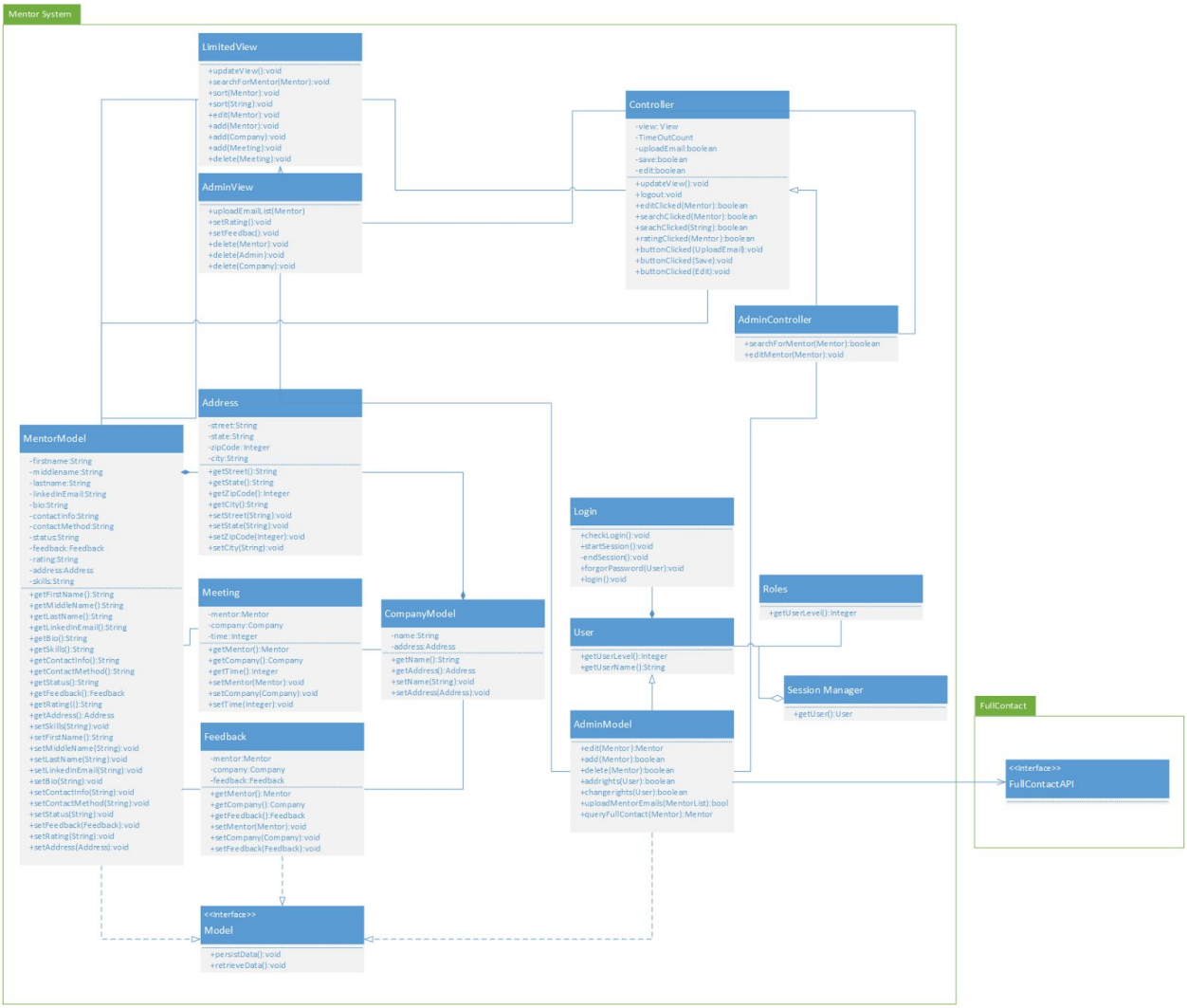
2. Which features were not implemented from Part 2?

- FR-001: Automatic mentor import using a Linkedin URL, and pinging Full contact API,we would need to obtain a license from Full Contact in order to use this feature.
- FR-002: Store a mentor's linkedin profile, linkedin forbids the storing of information scraped from the web pages.

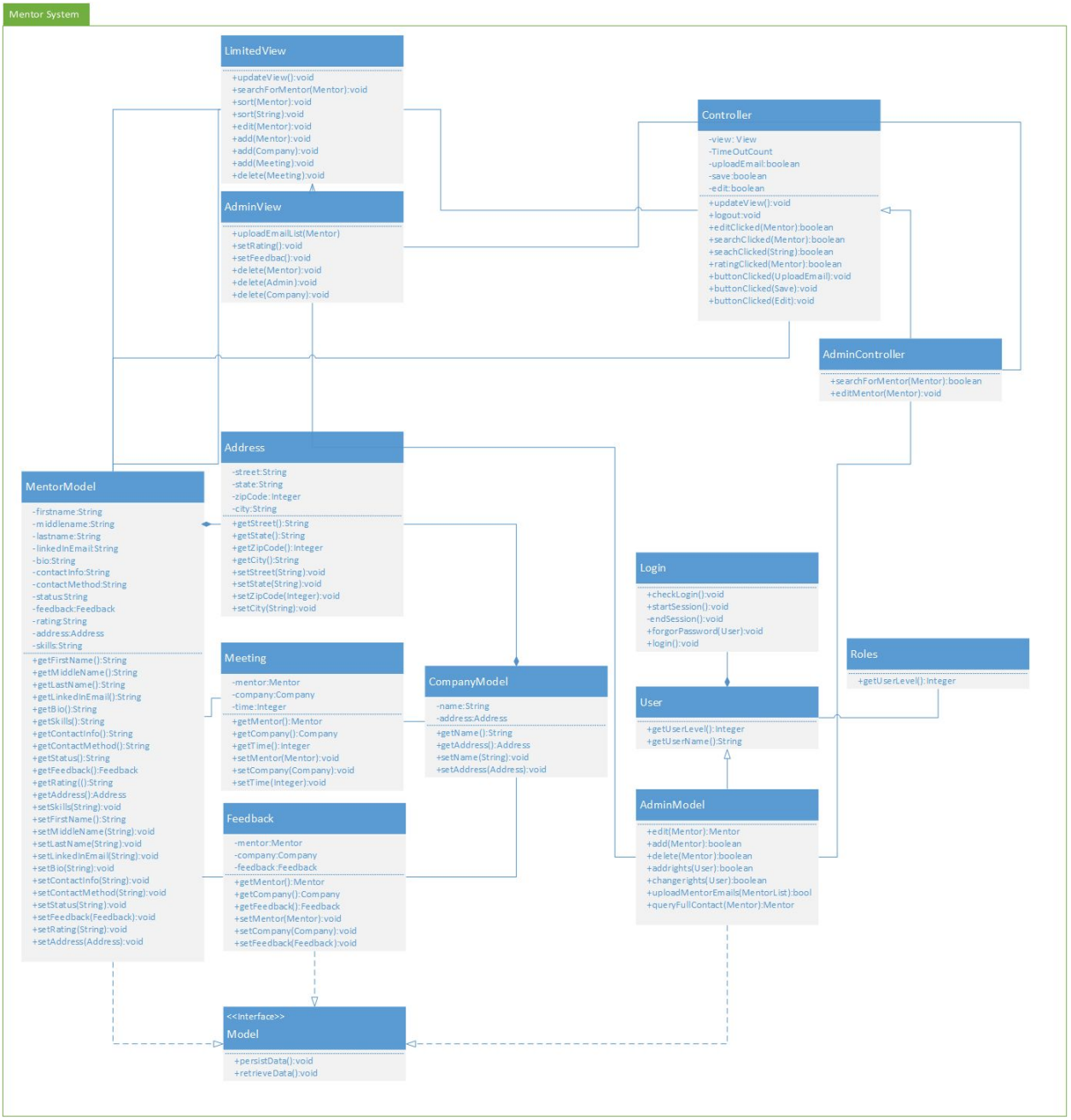
- UC-012: Mentor sorting (by average rating, newest, most active, etc).
- Mentor Profiles: in depth mentor profiles were not created, but this is something we are doing next after the class is done
- FR-003: Adding mentees and mentee profiles to associate them with a mentor.
- FR-005: Save company and mentor meeting. We learned that this was not a critical feature, from talking to customers, so we lowered priority on this.
- NFR-001 - The system should retrieve a Linkedin profile in one minute. Linkedin functionality was not implemented therefore this requirement is no longer applicable.
- UC-018 (Partial): You can add skills to mentor, **but you cannot edit skills**

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:



Final Class Diagram:



Our original and final class diagram didn't change that much. We were originally going to use Full Contact for populating mentor profiles via LinkedIn email address, but Full Contact would require a license that we are not willing to pay for. We also eliminated the Authentication class since it is offered by Django.

We knew we would be implementing MVC and we also knew we would use a MVC web framework to implement it. Consequently our MVC based design didn't change that much. However, our design allowed us to nicely map our classes from the class diagram into the Django framework. For example, modeling our database literally required us copying fields from our models and then issuing some Django specific commands.

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?

By electing to go with the Django framework we were forced to use MVC for our design pattern. Django implements MVC via the following mechanisms:

- A view is referred to as Django Template
- A model is referred to as a Model
- A controller is essentially the framework itself with connections setup in a urls.py file.

So all of our models specified in our class diagram from part 2 were implemented in models.py and our views were implemeted via Django templates and the connections were established between the view, model, and controller/Django framework with the urls.py file located in our project.

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?

- We've learned that the more time you spend carefully crafting your designs up front, the less guesswork is involved when coding your solutions.

- Design is also critical for many frameworks these days. For instance, Django can do a lot of the coding for you but you have to know where to stick your views, models, and controllers. Going through the design process helps identify what the models and views are.