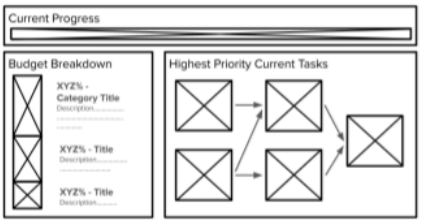
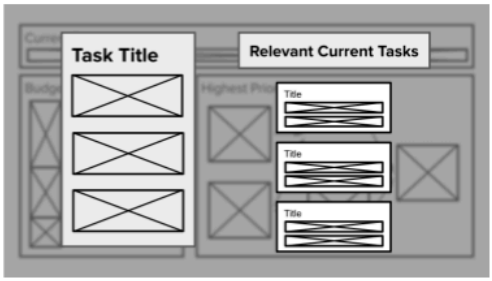
Team members: Sri Vangaru, Alex Urbina, Sean Sullivan, Yuxuan Guo

Process Deliverable II + Design Sketch

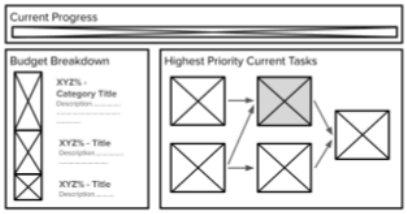
Summary: As we are going with Prototyping as our method of practice, our main process deliverable lies in our Design Sketch. For that purpose, we decided to make a low-fidelity prototype in the form of a wireframe for our task analysis. Specifically, we decided to analyze the user task of **Viewing Related Tasks** when using our service to understand the current progress of a project. We chose this task because it can be useful and convincing in showing that a particular aspect or effort in the project is important by demonstrating how many and which developers are dedicating time to it, and it also allows the user to have a more intuitive grasp of the distribution of tasks being worked on right now through this simple sort of recommender system.



Step 1. Initial screen.

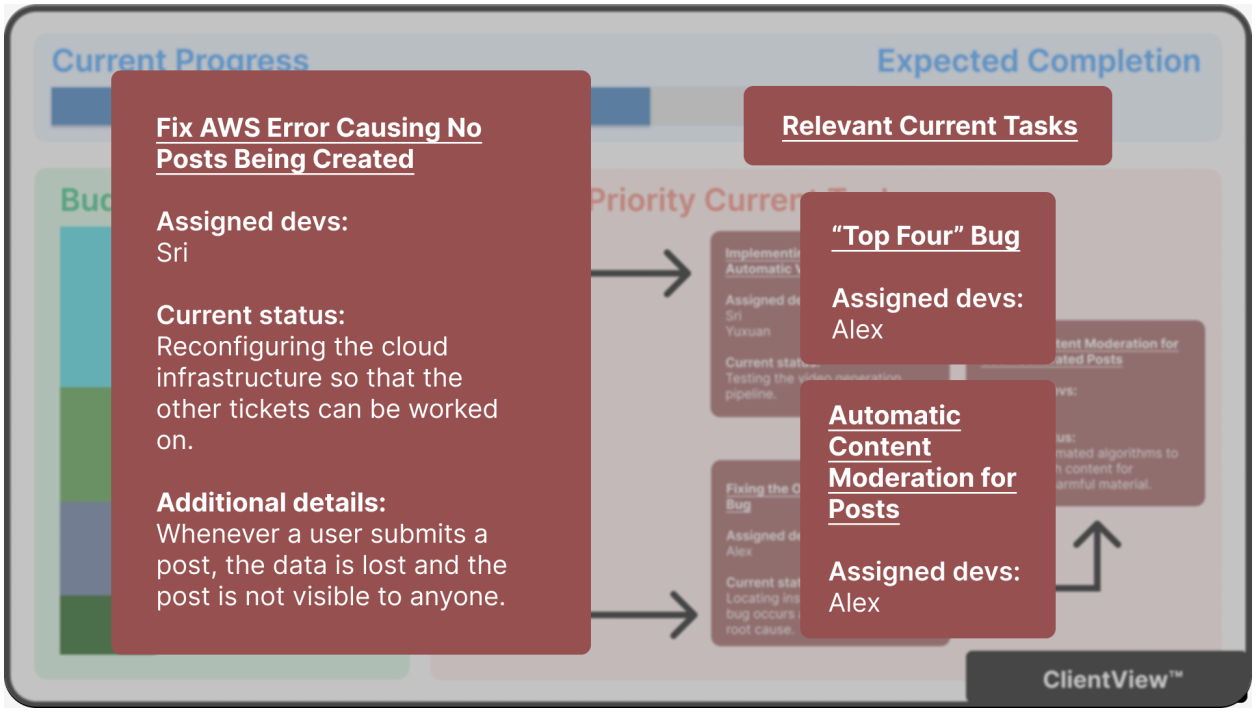


Step 3. View the task focus screen, with other relevant tasks and their information.



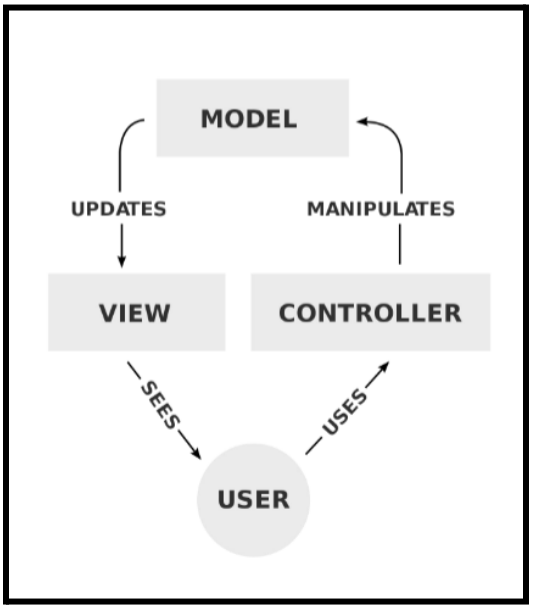
Step 2. Click on a task. It will shimmer temporarily while loading.

Design decisions rationale: A more realistic view of this feature is as shown below. One design decision is that we would allow the user to be able to click on other tasks through this screen itself, to be taken to a similar focus screen for that task, instead of having to go back to the home screen because it is more convenient. They can also go back to the default view through the standard/intuitive UI heuristic of clicking on any of the grayed out area (which is a pattern seen in many apps). This way, this feature acts like a magnifying glass, allowing a client to take an intuitive tour through the tasks that form up the current state of the team's progress.



High-Level Design

We chose a Model-View-Controller (MVC) architecture for *ClientView* because it facilitates a UI to interact with users and stores and retrieves information as needed (used by most modern web applications). The **Model** layer will handle Jira API integration and data management, allowing for real-time synchronization of project data. The **View** layer will handle creating simplified, client-friendly dashboards and visualizations, making complex project information easy to understand for non-technical stakeholders. The **Controller** layer will manage transforming technical Jira data into simplified visualizations, handling the main requirement of making project progress more accessible to clients. This clear separation makes the system highly maintainable, as changes to the Jira integration won't affect the client interface, and new visualizations can be added without modifying the backend.



Low-Level Design

Implementing the "View Task Dependencies" feature could benefit from the Behavioral Pattern family, particularly the Observer Pattern. This is because the system needs to reflect task dependencies dynamically and update them in real-time as data is synchronized from Jira. Moreover, the Observer Pattern efficiently propagates updates to all subscribed components whenever the task data changes.

```
# Example Code
task_data = TaskData()
dependency_graph = DependencyGraph()
critical_path_highlighter = CriticalPathHighlighter()

task_data.add_observer(dependency_graph)
task_data.add_observer(critical_path_highlighter)

# Update task dependencies
task_data.update_dependencies({"Task A": ["Task B", "Task C"], "Task
B": ["Task D"]})
```