3 Project Plan

3.1 Project Management/Tracking Procedures

What project management style are you adopting (e.g., agile, waterfall, or waterfall+agile)? Justify your project management style concerning the project goals and contexts.

Also, describe how your team will track progress throughout the course of this and the next semester. This could include Git, GitHub, Trello, Slack or any other tools helpful in project management.

For GridAI, a predominantly agile methodology will be at the forefront of management. In terms of our code, we are using agile, with minor, faster updates and additions; we will deploy new code very frequently. Our sprints are weekly meetings and close-to-daily discussions where we have discussed the nature of what we've worked on, the progress we've made and preparing for our other weekly client meetings with Dr. Gelli and his associates. This method allows us to have a cohesive structure and awareness of what we are all working on, making it easier for the client to understand what we have done. We believe an agile methodology is better for frequent deliverables, adaptability, and transparency with each other and our clients.

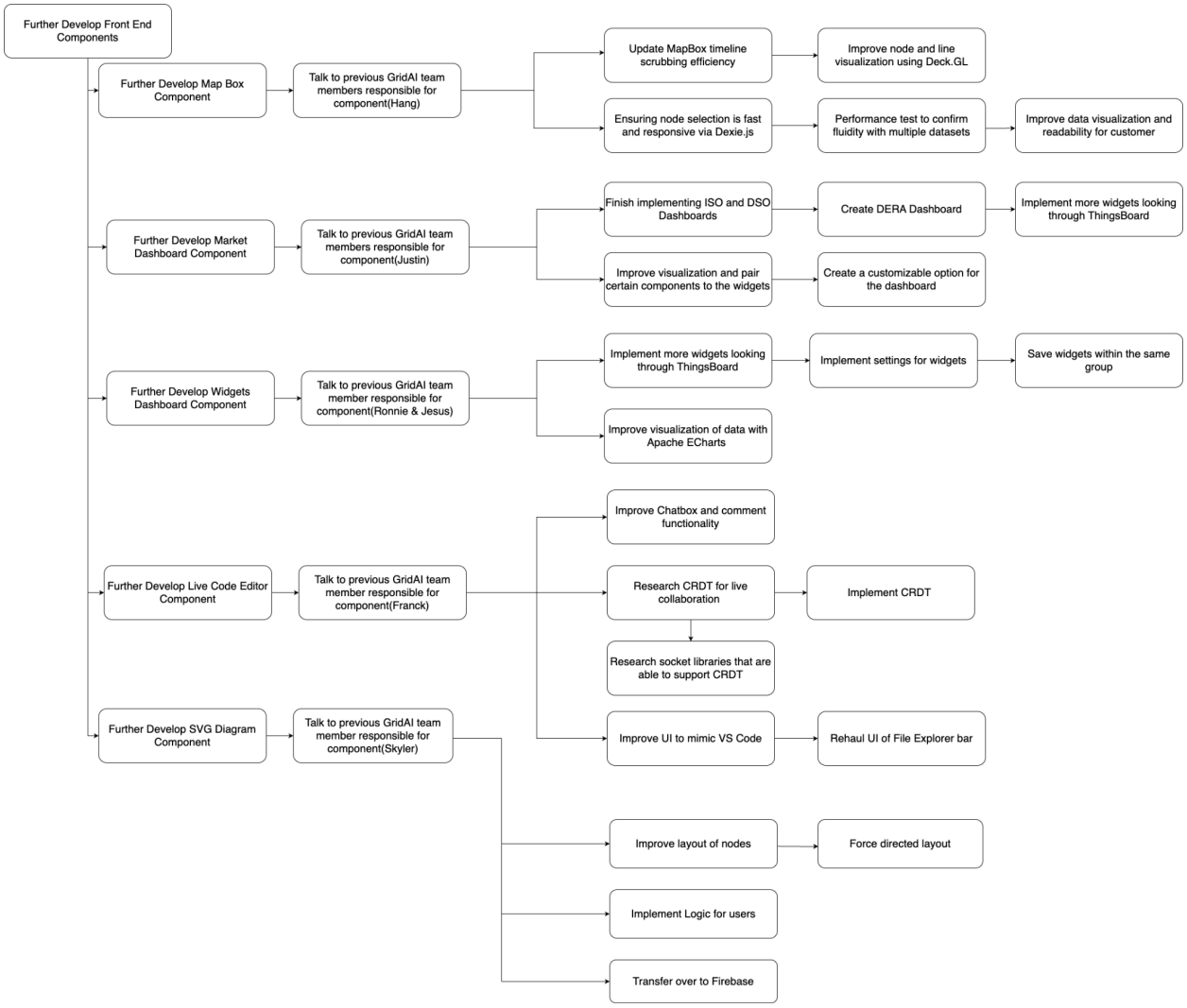
We use Gitlab as our primary project management tool, utilizing features such as the issue board to show what we are currently working on and what still needs to be completed. It also allows us to use CI/CD pipelines to automate testing and deployment and ease frequent updates. There are also simple merge requests with code review that enable all our work to be checked by another. We primarily use Discord for communication, as it allows us to have both text and voice communication channels and is easily accessible from anywhere on any device.

3.2 Task Decomposition

In order to solve the problem at hand, it helps to decompose it into multiple tasks and subtasks and to understand interdependence among tasks. This step might be useful even if you adopt agile methodology. If you are agile, you can also provide a linear progression of completed requirements aligned with your sprints for the entire project.

To effectively address the solution, the frontend improvements for GridAI have been decomposed into high-level components. These primary components are the Live Code and File Editor, Dashboard and Widgets, Map Box Visualization, Market Dashboard, and SVG Diagrams. Each component is then broken down into the specific subtasks needed to complete the component. This structured approach enables efficient task management, promotes parallel development, and ensures measurable progress through clearly defined objectives.

Below is a Task Decomposition chart to visualize our tasks for each component.



3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

What are some key milestones in your proposed project? It may be helpful to develop these milestones for each task and subtask from 3.2. How do you measure progress on a given task? These metrics, preferably quantifiable, should be developed for each task. The milestones should be stated in terms of these metrics: Machine learning algorithm XYZ will classify with 80% accuracy; the pattern recognition logic on FPGA will recognize a pattern every 1 ms (at 1K patterns/sec throughput). ML accuracy target might go up to 90% from 80%.

In an agile development process, these milestones can be refined with successive iterations/sprints (perhaps a subset of your requirements applicable to those sprint).

GridAI's workflow operates weekly, with progress tracked regularly and milestones updated consistently. Every Friday, the team holds a group meeting to discuss progress made during the week, outline objectives for the upcoming week, and address any challenges encountered. Each member provides an update on their assigned tasks, detailing accomplishments and any blockers they faced. The team meets weekly with their advisor to receive feedback, refine action items, and ensure alignment with project goals.

The team utilizes GitLab's issue board and milestone tracking system to manage development. Each milestone is tied to a significant feature, enhancement, or bug fix. Progress on tasks is measured by issue completion within GitLab, including code review, testing, and integration into the project. Once a milestone is reached, corresponding work is reviewed and merged into a staging branch. After verification and approval, changes are pushed to the master branch. This structured workflow ensures a smooth development process and helps maintain code stability while allowing incremental progress toward the final deliverables.

3.4 Project Timeline/Schedule

• A realistic, well-planned schedule is an essential component of every well-planned project

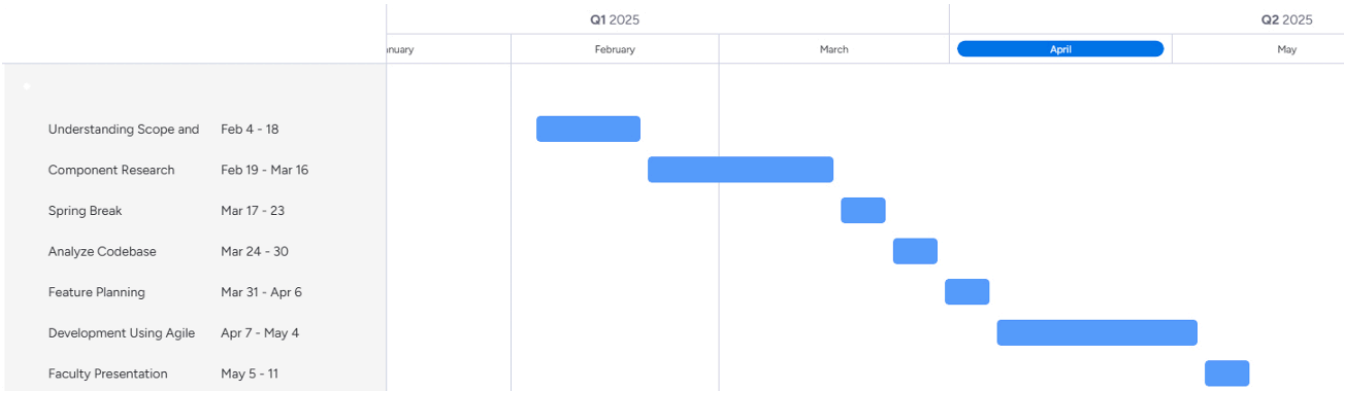
• Most scheduling errors occur as the result of either not properly identifying all of the necessary activities (tasks and/or subtasks) or not properly estimating the amount of effort required to correctly complete the activity

• A detailed schedule is needed as part of the plan:

– Start with a Gantt chart showing the tasks (that you developed in 2.2) and associated subtasks versus the proposed project calendar. The Gantt chart shall be referenced and summarized in the text.

– Annotate the Gantt chart with when each project deliverable will be delivered

• Project schedule/Gantt chart can be adapted to Agile or Waterfall development model. For agile, a sprint schedule with specific technical milestones/requirements/targets will work.



3.5 Risks And Risk Management/Mitigation

For each task, identify all salient risks (certain performance target may not be met; certain tool may not work as expected) and assign an educated guess of probability for that risk. For any risk factor with a probability exceeding 0.5 and each high severity risk, develop a risk mitigation plan. Can you eliminate that task and add another task or set of tasks that might cost more? Can you buy something off-the-shelf from the market to achieve that functionality? Can you try an alternative tool, technology, algorithm, or board?

Agile projects can associate risks and risk mitigation with each sprint.

Task	Risk	Risk Probability	Mitigation
Websocket connections	Risk of connections not being able to handle multi-user scenarios consistently	50%	1. Research into better live-time implementations such as CRDT or OT 2. Implement a socket library that implements the collaborative structure.
Limited testing with real-world users	Risk of the project not being able to handle large-scale real-world grids and may be unstable or crash	50%	1. Have more comprehensive testing with large datasets 2. Stress test with large constant data
Backend Integration with large datasets	Risk of large datasets causing delays in rendering the data onto maps and other widgets.	60%	1. Optimize operations to retrieve the project and display its nodes 2. Optimize frontend display and managing of data

3.6 Personnel Effort Requirements

Include a detailed estimate in the form of a table accompanied by a textual reference and explanation. This estimate shall be done on a task-by-task basis and should be the projected effort in total number of person-hours required to perform the task.

The GridAI team in comprised of six students, each with unique backgrounds in technologies to bring into the project. The team is structured for efficiency and to ensure critical areas of the project are covered. Each member is responsible for a major component of the project. During the embarkment of this project, the team will be going through unrelated coursework due to the academic semester. Due to external obligations, our team has agreed that each member will aim for 5-8 hours of work each week deciated to GridAI. This and the team's weekly meeting ensure consistency in progressing towards the team's deliverables.

Effort Requirements For Implementing Live Code Editor

Task	Estimated Hours
Front-end logic	**40**
CRTD Implementation	**40**
Code comments and chatbox	**30**
UI Overhaul	**50**

Collaborative features mimicking Google Suite	**20**

Effort Requirements For Implementing SVG Diagrams

Task	Estimated Hours
Improve Node Layout	**20**
Force Directed Layout	**15**
Implement Logic for users	**25**
Transfer over to Firebase	**20**

Effort Requirements For Implementing Market Dashboard

Task	Estimated Hours
Adding all components for visual metrics	**20**
Implementing real-time updating of visuals	**25**
Finishing DSO and ISO dashboard pages	**15**
Creating DERA dashboard page	**40**
Bonus (if time allows): Make the market dashboard customizable	**30**

Effort Requirements For Implementing Widgets & Dashboard

Task	Estimated Hours
Improve visualization of data	**25**
Implement more widgets	**30**
Implement widget settings	**30**
Implement bundle widgets (saves widgets in the same group)	**30**
Potentially integrate the real-time data into the widgets.	**40**

Effort Requirements For Implementing Map Box

Task	Estimated Hours

Update MapBox timeline scrubbing efficiency	**30**
Ensuring node selection is fast and responsive	**30**
Improve data visualization and readability	**30**
Bonus (if time allows): Use Deck.GL to improve visuals of node energy usage with 3D elevation	**40**

3.7 Other Resource Requirements

Software Requirements

1. Virtual Machine (7):
    a. Specifications:
        i. 8 vCPUs, 16 GB RAM, 75 GB storage each
        ii. Ubuntu 22.04.5 LTS installed
    b. Purpose:
        i. Consistent development environment between team members
        ii. Testing in a secure isolated environment
2. Firebase:
    a. Purpose:
        i. Real-time database for grid monitoring and updates
        ii. Authentication for secure access
        iii. Cloud hosting for backend operations and deployment
3. Development Tools:
    a. Visual Studio Code
        i. Extensions installed to support project tools and libraries
    b. React
        i. Node.js for package management
        ii. Jest and React Testing Library for testing
        iii. Tailwind CSS for styling
        iv. Mantine UI for components
        v. React Developer Tools for debugging
        vi. TypeScript For Developing The Frontend
    c. Git
        i. Enables collaboration by allowing all team members to work on the same codebase simultaneously
        ii. Uses branching to isolate feature development and avoid conflicts from working on similar components
        iii. Maintains a complete history of code changes for traceability and version control
    d. Docker
        i. Allows our team to build and run the same environment with consistent dependencies across different machines
        ii. Simplifies deployment by ensuring the application runs consistently across development, staging, and production environments

iii. Helps ensure all team members can run the project with minimal setup, reducing environment-related issues