GridAI: Front-end Team

Design Document

sddec25-17

Revikumar Gelli - Client/Adviser

Rolf Anderson - Mentor

Peeyush Gupta - Mentor

Developers:

Ponciano Ramirez - Widgets/Dashboard

Tristan Nono - Widgets/Dashboard

Yusef Harb - Live Code Editor

Ethan Messmer - Market Dashboard

Nicholas McCullough - Map Box

Evan Sivets - SVG Diagram

Email: sddec25-17@iastate.edu

Website: sddec25-17.sd.ece.iastate.edu

Acknowledgment

This research is partially supported by U.S. NSF Grant CNS-2105269 and U.S. DOE CESER Grant DE-CR000016.

Executive Summary

GridAI is a research-driven power grid management platform developed at Iowa State University over multiple years. This application is designed to tackle modern challenges in modern energy distribution systems. GridAI empowers operators with tools such as outage prediction, power flow optimization, and effective management of distributed energy resources (DERs). The backend system of the platform is highly developed and feature-rich, but the front-end user interface was limited in flexibility for different types of users. The team is focused on redesigning and expanding the frontend experience. These improvements aim to improve usability, interactivity, and real-time data transfer.

Problem Statement

As modern power grids become increasingly complex, ensuring accessible and intuitive interfaces is essential for effective system management. Distribution System Operators (DSOs), DER Aggregators (DERAs), and Independent System Operators (ISOs) must interact with detailed visualizations, configure control settings, and analyze historical trends, all within a user-friendly web interface. When these interfaces are buggy, fragmented, or slow to respond, it leads to operational inefficiencies and even the potential for costly mismanagement if the data is unreliable during critical grid events.

Requirements

- **User-Friendly Interface:** The interface must be easy to navigate and responsive, accommodating various user roles with a minimal learning curve.
- **Live Data Visualization:** Real-time, low-latency updates are essential for timely insights and decision-making.
- **Scalable and Maintainable Architecture:** The system should efficiently handle large datasets and support straightforward maintenance and future feature expansions.

Design

- **Modular Component Architecture:** The system is structured using reusable React components, making coding easier to develop, test, and maintain individual features across the UI.
- **Server-Component Model:** Optimizes performance by separating rendering logic and reducing front-end load, improving scalability across large datasets and user sessions.
- **Consistent System Design:** Built with TailwindCSS to ensure UI consistency across devices and resolutions, improving usability for operators in varying environments.

Technologies Used

- **Frontend:** Developed with React, TypeScript, TailwindCSS, and Next.js to create a dynamic, responsive, and maintainable UI.
- **Backend:** Utilizes Firebase for real-time data handling and authentication, InfluxDB for managing time-series data, and MongoDB for storing and customizing widget configurations.

- **Real-Time Communication:** Employs WebSocket to enable instant data synchronization and support collaborative multi-user editing.

Key Components and Progress So Far

- **Widgets and Dashboard:** Implemented multiple new widgets and started adding real-time data integration into the application.
- **Map Box:** Started on new Figma designs for Map Box node icons for EV Charging Stations, Solar Systems, Substations, Transformers, and more to use with Deck.GL, in the process of implementation into the codebase. Initiated Mantine to ShadCN UI migration.
- **Code Editor:** Enabled backend to support CRDT websocket communication and laid groundwork for frontend implementation.
- **SVG Diagram:** Focused on the layout best to resemble a real-life example of a single line diagram, and formed a plan to implement this data to Firebase.
- **Market Dashboard:** Started transfer of Mantine components to ShadCN and is building a foundation for finishing up the remaining platforms.

Next Steps

- **Integration:** Improve connectivity to the frontend system to real-time data streams from the backend.
- **Performance Enhancement:** Improve upon GridAI's current components through code refactorization, reduced time complexity and integrating performance-based technologies into the project, such as Dexie.js for the Map Box component for client-sided IndexedDB to retrieve large datasets efficiently.
- **Testing:** Conduct thorough testing as outlined in the team's testing plan, including usability evaluations and feedback from real users to guide further improvements.

Learning Summary

Development Standards & Practices Used

- **Software Development Practices:**
  - Documentation with tools like Swagger for APIs
  - CI/CD using GitLab CI to automate testing and deployment
  - Version Control with Git to collaborate through branching and pull requests
  - API verification with Postman
  - Agile Methodology to ensure flexible progress and adaptability to changes
- **Applicable Engineering Standards:**
  - ISO 50001: International Standard for Energy Management Systems
  - ISO 9241-210:2010 — Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems
  - ISO 5055 - Software Quality Standards
  - IEEE 1547.3-2023 — IEEE Guide for Cybersecurity of Distributed Energy Resources Interconnected with Electric Power Systems

Summary of Requirements

- **Functional Requirements:**
  - Must display accurate information and data for users to ensure reliability.
  - The system must support user authentication.
  - Must validate input data before processing to prevent errors.
  - Allow collaboration for developers through a code editor environment.
- **Non-Functional Requirements:**
  - Must be easy and intuitive for users.
  - The system must operate with minimal downtime when the project is deployed fully.
  - The codebase must follow clean architecture principles, enabling easy updates, debugging, and feature extensions.
- **UI/UX Requirements:**
  - Must provide customizable ways for users to display data.
  - Integrate Mapbox for interactive geospatial visualizations and SVG Single Line Diagrams to provide an overview of the electrical grid's structure and flow.
  - Design and implement a user-friendly Market Dashboard interface that allows users to submit, view, and manage bids easily.

Applicable Courses from Iowa State University Curriculum

- **SE 3090**: Software Development Practices
- **SE 3190:** Construction of User Interfaces
- **SE 3290**: Software Project Management
- **SE 3390**: Software Architecture and Design
- **COM S 3630:** Introduction to Database Management Systems
- **SE 4190**: Software Tools for Large-Scale Data Analysis
- **SE 4210:** Software Analysis and Verification for Safety and Security

New Skills/Knowledge acquired that was not taught in courses

- Advanced Docker
- Advanced Node.js
- Advanced React
- Advanced CI/CD pipelines
- Dexie.js
- Advanced Websockets
- Custom React Hooks
- Modular Design
- Radix
- Deck.GL
- Apache Kafka
- SVG Manipulation
- TailwindCSS
- Typescript
- React-Map-GL
- Next.js
- Firebase

Table of Contents

List of Figures/Tables/Symbols/Definitions

Figures

Tables

1. Introduction

1.1. Problem Statement

With an endless need for electricity growing, the efficient management of power grids is more crucial than ever. Power distribution networks are increasingly complex, integrating multiple energy sources, including distributed energy resources (DERs) such as solar and wind power. Unlike traditional power generation, these must be managed dynamically to account for the fluctuating energy inputs. Additionally, grid operators must handle multiple input streams, including real-time data from sensors, historical usage patterns, and AI-driven predictive analytics, making a well-designed user interface (UI) and front-end system essential for effective decision-making.

GridAI, a power grid management software, aims to provide an advanced solution for optimizing and managing grid performance. Currently, the backend of GridAI is fully developed. However, the application still requires development on its UI and front-end components. The team has been tasked with improving existing front-end components and UI to ensure seamless user interactions and clear and efficient data visualizations.

1.2. Intended Users

1.2.1. Commercial Users

This group of users contains different facets of the power sector. Each is responsible for overseeing and managing large-scale power grids. These can be further broken down into the following:

DSOs:

DSOs, or Distribution System Operators, are primarily responsible for local energy distribution. They likely are in charge of maintaining local energy grids and connecting them to the end user. They need the following information:

- DERA operator
- Timeline of analyses
- Grid Impact Risk
- Grid location
- Bids under review
- Grid Capacity

ISOs:

ISOs or Independent System Operators manage wholesale electricity, ensure grid reliability, operate the energy market, and coordinate power flow. An ISO requires information similar to a DSO, with a few exceptions.

- More of an emphasis on bids
- Price of electricity and the market value
- The current status of the market

DERAs:

DERAs, or Distributed Energy Resource Aggregators, combine small energy resources, manage multiple virtual power plants, and optimize the DER portfolio. A DER again needs much of the same information, but some unique data is as follows.

- Full bid timelines
- The performance and utilization of different power grids
- Service plans offered
- Real-time execution data

These three user types represent the primary professional users of the Grid AI application, and the team is committed to catering to the needs of these three users with features such as the Market Dashboard, which will be showcased later in the document. This will be where any information a commercial user requires will be available.

1.2.2. Residential Users

This group of users represents an average homeowner or small business owner with little to a lot of knowledge of their residential grid and power consumption. They want to reduce electricity costs, track usage, and integrate grid or solar panels.

User Needs:

- A responsive front-end application
- Energy tracking tools for real-time data on statistics, power consumption, cost, etc
- Easy to navigate and professional UI
- Compare renewable energy (solar) vs grid energy utilization
- Clear visuals of the data, such as Map Box, where they can view the energy consumption per bus node, relevant to their energy address or small cluster

1.2.3. Educational Users

This group of users represents users in the academic industry. They are academic researchers analyzing power grids and need a software tool to help them collect and analyze data. They will be dealing with multiple grid sources of varying sizes and collaborating with numerous other researchers. They are interested in utilizing GridAI to closely evaluate and organize large amounts of grid data from multiple input streams.

User Needs:

- Be able to handle and organize multiple input streams
- Data visualization tools to help easily understand
- Features to filter and organize datasets efficiently
- Features to collaborate and share data with other researchers
- Simple and clean navigation
- Generate automated reports based on grid simulations with data analysis

1.2.4. Developer Users

This group of users represents the developers who could expand upon GridAI by creating their own Widgets and using the platform to collaborate with other engineers working on the same project. They need an environment to work efficiently and in a team-friendly environment.

User Needs:

- Being able to deploy code and interface updates in a controlled environment for testing
- Effective communication between the team through Code Editor comments via Websockets
- Access to the codebase, system logs, and backend APIs for RESTful connections for development
- User-friendly UI that feels natural for a developer
- Be able to view Map Box functionality without limitations

1.3 User Research

1.3.1 Empathy Map

To better understand GridAI's intended users, the team collaborated to create an empathy map to help visualize user thoughts, feelings, behaviors, and challenges. This allowed the team to design a solution that aligns with the user's needs.

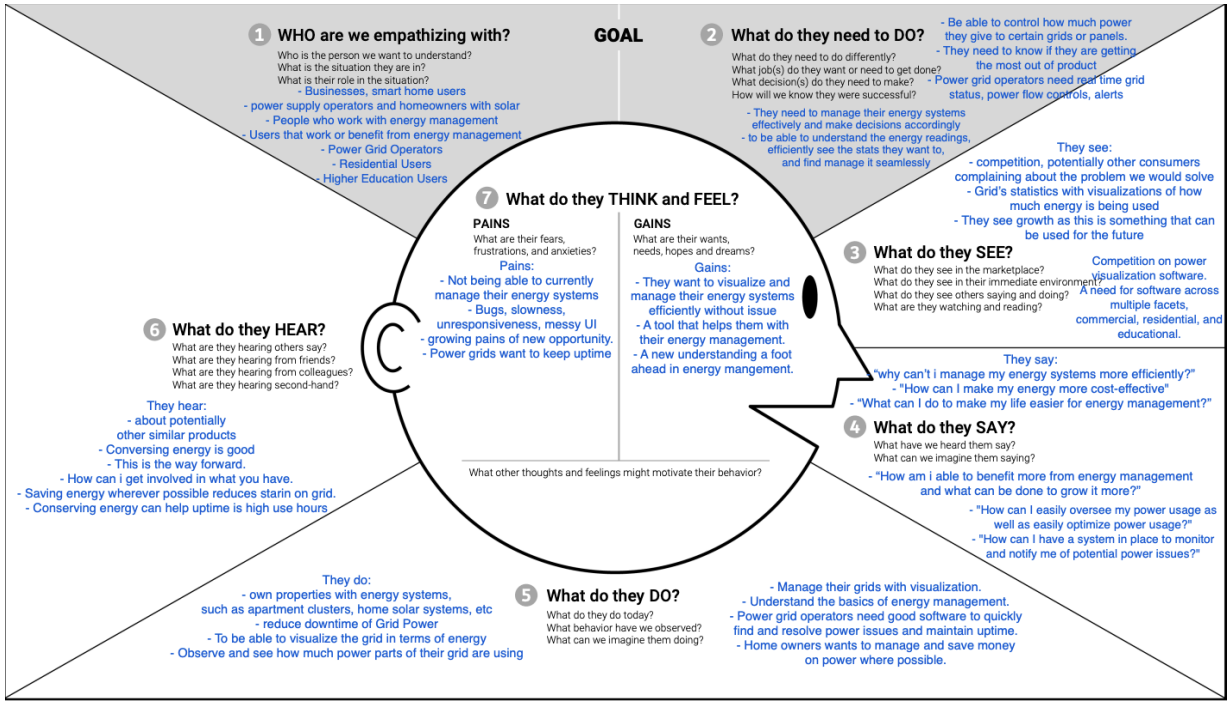

Figure 1: Empathy Map

1.4 Conclusion

This design document provides a structured approach to defining GridAI's UI development goals. By focusing on user-driven design enhancements through research and thinking about what GridAI users need, the team aims to create a highly interactive and scalable UI that empowers homeowners, researchers, developers, and grid operators to make informed energy decisions like never before.

2. Requirements, Constraints, And Standards

2.1. Requirements & Constraints

Requirements:

- Functional Requirements
  - Real-time grid monitoring - The application must display power voltage, location, speed, and other data applicable to the user.
  - Data visualization - Data must be displayed in geographic map diagrams and single-line diagrams (SLDs).
  - Code Editor - The code editor must support real-time multi-user collaboration, syntax highlighting, inline commenting, and a live chat for discussion between developers.
  - Market Dashboard - The market dashboard must display real-time data from what is monitored and provide a search, filter, and sorting for market metrics.
- Resource Requirements
  - Cybersecurity - GridAI must comply with commonly accepted cybersecurity standards such as ISO 27001 or NIST.
  - API and Backend integration - The frontend must be able to access data from the backend using REST and other APIs (e.g., the Bus and Lines APIs for the map box).
  - Database fetching - The frontend must be able to access static and dynamic data from their respective databases. (e.g., All static data is currently stored in Firebase).
- Physical Requirements
  - Multi-device functionality - The application should work across multiple devices such as tablets, laptops, phones, and more, supporting touch functionality as needed.
  - Performance optimization - GridAI must use techniques like lazy loading to ensure quick loading times and well-loaded images.
- Aesthetic Requirements
  - Consistent theme - The theme of GridAI must have uniform styling across different components, with a dark mode and high contrast option available.
  - Interactive data - Analytics such as grid status or market trends should be color-coded for ease of comprehension.

- Experiential Requirements
    - Easy navigation - Users should be able to switch between dashboards seamlessly and in less than two clicks.
    - Accessibility - GridAI must be usable by all, with accessibility features such as high contrast and display customization. This could be expanded further with screen readers and voice control features later in development.
- Economic/Market Requirements
    - Subscription model or API monetization - A revenue stream for users to pay for GridAI's services.
- UI Requirements
    - Widget-based Dashboard - Users should be able to resize, move, and configure widgets created with prebuilt templates or from scratch.
    - Market Dashboard - Offers historical and live data on charts with tooltip explanations for market terms available.

Constraints:

- Uptime - Data has to be available as often as possible, shooting for near 100% uptime.
- Processing speed - Must be able to handle thousands of transactions per second.
- Load time - Components such as widgets or the map box should be able to be loaded in as little time as possible, shooting for less than two to three seconds.
- API call response - API calls should be quick and efficient, shooting for less than 500 ms per API call.
- Speed of intractability - UI elements should be fully interactive shortly after the page loads.
- Keyboard shortcuts - Keyboard shortcuts should be available for many of the core features for ease and speed of functionality.

2.2. Engineering Standards

Engineering standards are in place across industries to ensure safety, reliability, and interoperability. These standards provide a common ground for designing, testing, and manufacturing products to help maintain consistency and quality across a common industry. Companies could cut corners without engineering standards, potentially compromising safety and reliability. With engineering standards, consumers can be confident in the purchases they make. For example, various standards regulate data sharing and security in personal health devices. Without these safeguards, consumers might hesitate to trust such devices, uncertain how their sensitive health information is handled.

The first standard, ISO 50001, is about making the developer's life more manageable for organizations to integrate energy management into their work efforts. This standard wants to use energy efficiently and use the data collected to be able to be used more efficiently. This standard intends to continue off the efficiency to improve the energy management and measure the results of all the data collected.

The second standard, 1547.9-2022, applies the IEEE Std 1547-2018 to interconnect energy storage of distributed energy resources. This is set out to bring energy resources to export active power to the electric power systems. Also, the standard sets out to address energy storage topics not covered thoroughly in the IEEE Std it went to apply, laying out the groundwork for future practices.

The third standard, 2030.2-2015, uses the reference of IEEE Std 2030-2011 to define the layout of energy storage systems interoperability with energy power systems. This is to cover the energy storage systems and how to integrate them with the electric power systems properly. The standard also covers correctly applying customer premises and working with bulk storage. Another topic is the standards IT, power systems, and communications methodology, all based on the IEEE 2030 definitions.

The fourth standard, ISO 5055, is a code quality measurement system that provides rules to ensure that the code of the software is effective in security, reliability, efficiency, and maintainability.

All four standards are relevant to this project because GridAI specifically integrates home power systems/solar/clusters into the existing power grid and to ensure when developing this project, the code is structured to be as maintainable and effective. This is precisely what IEEE 2030.2-2015 refers to regarding the innovative grid interoperability reference model (SGIRM) process regarding energy storage systems (ESS) that interface with the electrical power grid infrastructure.

The second standard, IEEE 1547.9-2022, refers to a guide on how to apply the interconnection of energy storage and distributed energy resources to electric power systems. This standard/guide also gives instructions on approaching these power system interconnections. This is relevant to GridAI because the application interfaces with electric power systems for the end user.

The third standard, ISO 50001 (software), discusses and provides a framework for developing an energy management system. This standard discusses developing a policy for efficient uses of energy, setting objectives, using data to make better decisions, measuring results, and continuous improvement. These are potentially important to the project because even though the infrastructure/backend is already established, knowing the engineering standards behind how they were developed and good practices will help us create the frontend for the user.

The fourth standard, ISO 5055, is used for evaluating how effective the software's code is in those four areas to help control risks and improve trustworthiness.

ISO 50005:2021:

This software standard discusses a phased implementation of developing an energy management system for small to medium-sized organizations.

ISO 50009:2021:

This software standard discusses guidelines for implementing a common energy management system for multiple organizations.

The project modifications planned incorporate these standards by making sure when it comes to energy management, the project is efficient, the data provided to the user is understandable, and the user can make decisions based on the provided data and use the application to improve energy management. ISO-50001 details the requirements of managing energy effectively, so the team has plans to use that in the GridAI application when developing.

The team can implement IEEE 1547.9-2018 standards in the project by developing the frontend to allow users to transfer specific amounts of power to their grid and keep track of the energy they are monitoring. This standard will also determine how the frontend will be structured and the direction

that the application will take, because a significant portion of GridAI's platform is based on energy storage and electric power systems.

IEEE 2030.2-2015 will ensure the team prioritizes development to make sure the data provided is understandable for GridAI's users. When data is displayed, the team must ensure the energy statistics are precise to facilitate the data comprehension and understanding by all people involved, such as residents or companies.

ISO 5055:

The team can utilize ISO 5055 when developing by ensuring code is maintainable for future developers to take over this project through writing clean code, documentation, and reducing unnecessary code. The team must ensure the project performs effectively. This can be done by making sure that calls are fast, efficient and there are little to no hiccups. The code will also have to be secure, ensuring data does not get leaked and shows correct data to the correct user.

3 Project Plan

3.1 Project Management/Tracking Procedures

For GridAI, a predominantly agile methodology will be at the forefront of the project. In terms of software project management, agile was used, with minor, faster updates and additions; the team will deploy new code very frequently. The team's sprints will be weekly meetings and close-to-daily discussions which will include discussions of work being done, progress made, and preparing for weekly client meetings with Dr. Gelli and his associates. This method allows the team to have a cohesive structure and awareness of what each member is working on, making development easier for the client to understand what the team has completed. The team believes an agile methodology is better for frequent deliverables, adaptability, and transparency with each other and the client.

GitLab will be used as the primary project management tool, utilizing features such as the issue board to show what is currently being worked 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 work to be checked by others. Discord is the team's primary communication tool, as it allows the team to have both text and voice communication channels and is easily accessible from anywhere on any device.

3.2 Task Decomposition

To effectively address the solution, the frontend improvements for GridAI have been decomposed into high-level components. These primary components are the Live Code Editor, Dashboard and Widgets, Map Box/GIS Visualization, Market Dashboard, and SVG Diagrams. Each element 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 the team's tasks for each component.

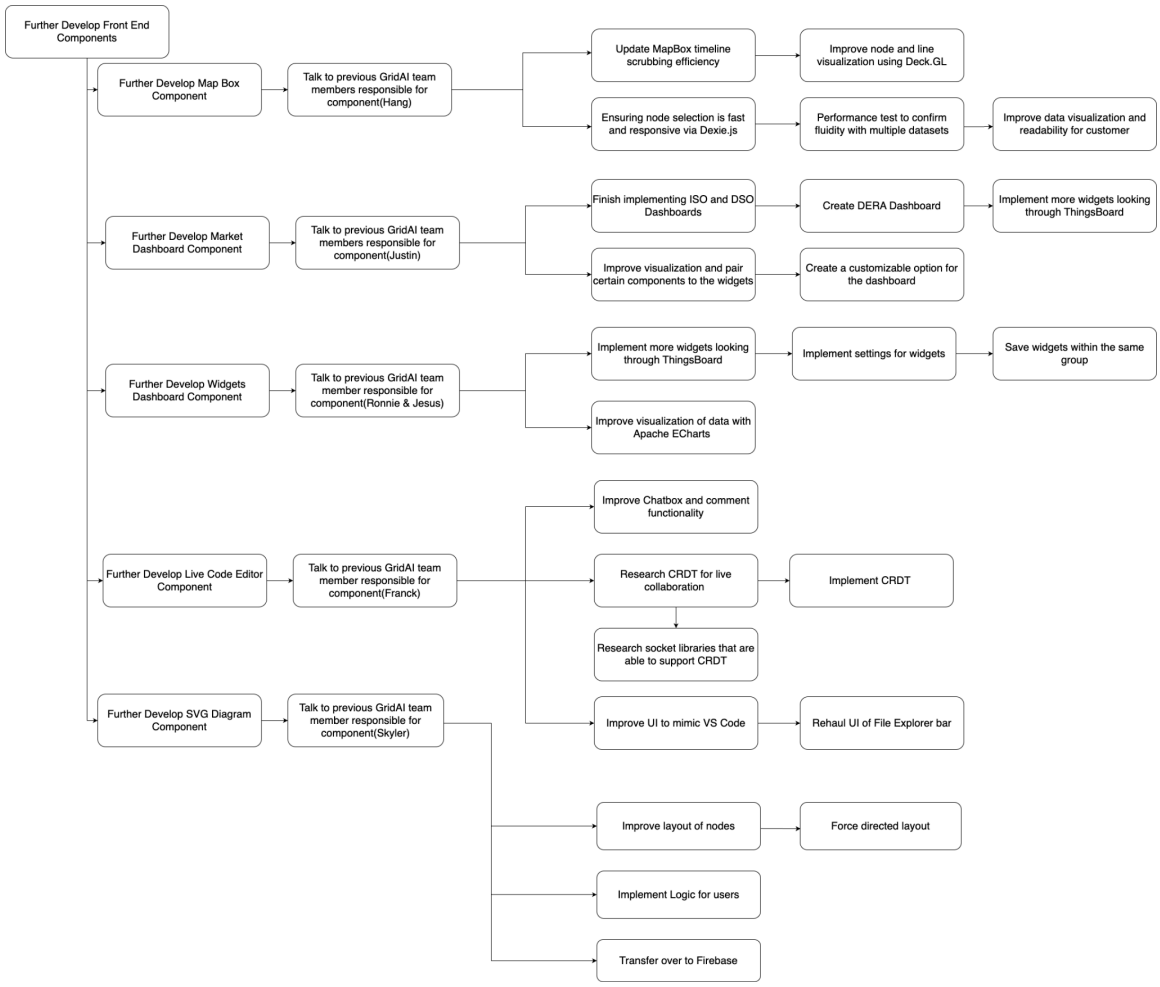

Figure 2: Task Decomposition Chart

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

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 manages development and uses GitLab's issue board and milestone tracking system. 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 staging branch. This allows the team to

merge all individual component code into a staging environment before merging to our protected main 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

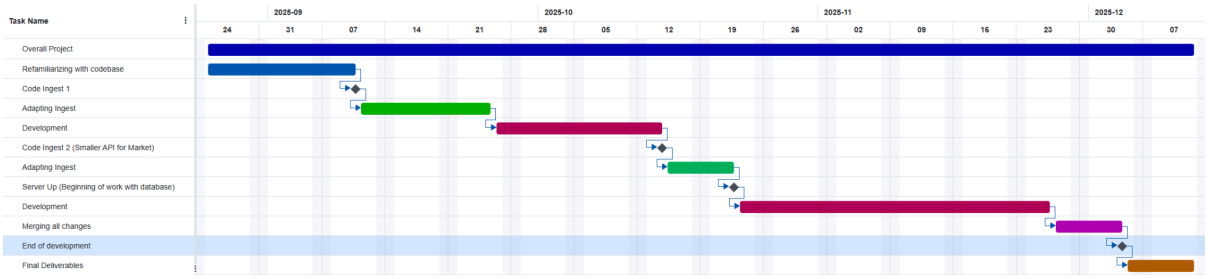

Figure 3: Gantt Chart

3.5 Risks and Risk Management/Mitigation

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 extensive constant data
Backend Integration with large datasets	The 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

The GridAI team is 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 significant component of the project. During the implementation of this project, the team will be going through unrelated coursework due to the academic semester. Due to external obligations, the team has agreed that each member will aim for 5-8 hours of work each week dedicated 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**

Task	Actual Hours
Front-end logic	**35**
Adapting Ingest	**20**
Code comments and chatbox	**40**
UI Overhaul	**60**
Attempted CRDT implementation	**20**

During the second semester, the team was also tasked with taking on the ingestion from the previous GridAI team. During this ingest, it was discovered that features intended to be completed for the code editor were not completed. Such as the chat and comments. This was a major setback that caused me to shift focus to that component that was otherwise unplanned. Although the Yjs implementation was scaffolded, it was incomplete as there were difficulties getting syncing to work and it was causing websockets to fail. This resulted in its removal and a delay for the next team.

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**

Task	Actual Hours
Improve Node Layout	**25**
Force Graph Layout	**20**
Implement Logic for users	**25**
Transfer over to Firebase	**30**

Due to unforeseen medical circumstances this semester, less time was spent on developing the SVG diagram component. In addition, the previous team's code for this component was less than anticipated in terms of both quality and quantity. Therefore, the amount of time that was available was spent trying to fix the previous team's code.

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 the DERA dashboard page	**40**
Bonus (if time allows): Make the market dashboard customizable	**30**

Task	Actual Hours

Adding all components for visual metrics	**20**
Mantine to Shadcn migration	**20**
Finishing DSO and ISO dashboard pages	**15**
Rewriting the tenant_service API	**40**
Connecting frontend to tenant_service	**20**

This semester went very differently than previously planned, with the introduction of the tenant_service API about halfway through, that became most of the time spent. The ingest of the previous team's code (sdmay25-43), and the dashboards (DERA was already done), were much further along than expected. As such, much of the time was spent on that API for nice-to-haves.

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**
Integrate the real-time data into the widgets.	**40**

Task	Actual Hours
Improve visualization of data	**25**
Implement more widgets	**15**
Implement widget settings	**10**
Implement bundle widgets (saves widgets in the same group)	**0**
Integrate the real-time data into the widgets.	**50**

This semester most of the development was spent on the widgets specifically integrating the real time data to the widgets. There were other implementations for the widgets such as settings and adding in more widget types for the user to choose from. Due to this and working on the dashboards, there was no time to develop the widget bundles as there were different priorities taking over development.

Effort Requirements For Implementing MapBox

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**

Task	Actual Hours
Update MapBox timeline scrubbing efficiency	**0**
Ensuring node selection is fast and responsive	**30**
Improve data visualization and readability	**70**
Bonus (if time allows): Use Deck.GL to improve visuals of node energy usage with 3D elevation	**30**
Developing and Preparing Figma Icons	**40**

During the second semester, Rolf advised to remove the Timeline functionality from the Map Box page altogether as it would be used for a future widget project instead. With this, a majority of time was spent developing implementation API endpoints for Loads and Transformers to be used in the project in the future. Additionally, a significant block of time was spent developing icons in Figma for expandability purposes for nodes on the map. In total, 71 high quality Figma icons were added to the project for a wider selection of node options. Instructions on how to correctly add them onto the map when the time comes were provided and also included in this design document below.

3.7 Other Resource 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. A 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 the 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

4 Design

4.1 Design Context

4.1.1 Broader Context

Area	Description	Examples
Public health, safety, and welfare	Many different user types will be positively impacted by GridAI, with little downside for using the platform.	With the potential increase in solar panels, there would be a natural reduction in traditional, pollutant-emitting power plant production, with no inherent safety risks. The knowledge from the platform poses no risk except a positive change for any of its users.
Global, cultural, and social	Energy literacy is critical in the global community. This has become more popular in Australia and some European countries. GridAI's use of AI to make this extremely understandable to users of all types allows for a more informed culture to develop.	Using AI could lead to a bias for or against the platform. Many users are potentially scared of using it, and others are more drawn to it being "future technology".
Environmental	GridAI will likely not have much of an environmental impact. There will likely be some beneficial aspects, with increasing energy efficiency and incentivising more people to use alternative energy sources.	Decreasing energy usage is one thing that many residential users will be using the platform for. They will understand their current usage better and potentially resort to alternative energy sources, such as Solar panels, to lower energy costs. This, in turn, would benefit the environment from a long-term perspective.
Economic	GridAI will be highly economically viable. Without much upfront cost from development, the main cost will be server maintenance and a small team. The many features make it highly competitive in today's market, especially when hosting a DERA platform on top of all the other features.	Currently, it is unknown what GridAI's cost will be. Hopefully, it is manageable, as the aim

4.1.2 Prior Work/Solutions

Previous Products:

1. Siemens Spectrum Power - Industry-standard SCADA/Grid monitoring system that has been used for many years professionally.
    - Pros: Reliable, highly regulated, and trusted by many companies.
    - Cons: Expensive, steep learning curve, proprietary, outdated UI, and features.
2. EcoStruxure - Schneider's interoperable smart energy management platform focuses primarily on buildings, data centers, and industrial grids.
    - Pros: Compatible with many legacy systems, energy efficiency tools, and a comprehensive ecosystem.
    - Cons: Expensive, static user interface, with latency in edge-cloud syncing.
3. Thingsboard - IoT data efficiency tool, primarily used for collecting, visualizing, and managing data and widely used for metering, asset tracking, and some industrial uses.
    - Pros: Open Source, lightweight IoT integration, highly scalable, and provides an automation rule engine.
    - Cons: Limited grid-specific features, real-time latency issues, steep learning curve.

4.1.3 Technical Complexity

GridAI is a very technologically advanced project, with many components working in parallel to each other that have been worked on for several years. The software combines some platforms that already exist, and some that don't yet.

Multi-Component Architecture:

Component	**Technical Challenges**	**Scientific/Engineering Principles Applied**
Widget Dashboard	Real-time data streaming and visualization with easy-to-develop/use widgets.	GPU-accelerated widget rendering with Apache Echarts and Reactive state management.
Live Code editor	CRDT-based collaborative editing with conflict resolution.	WebSockets, Operational transform fallbacks, and CRDTs
Map-Box	Deck.GL rendering thousands of grid bus nodes and lines.	IndexedDB caching, WebGL optimization, and Spatial indexing for node clustering.
Single Line Diagrams	SVG-based dynamic diagrams with drag-and-drop node customization	SVG performance optimization and Graph theory
Market Dashboard	Role-based UIs (ISO/DSO/DERA) with bid workflows, feasibility reports, and live/historical market monitoring.	Role-based access control and data aggregation pipelines.

Challenging Industry Standards:

Legacy systems like Siemens' Spectrum Power and Schneider Electric's EcoStruxure feel like ancient relics, clunky interfaces trapped in the early 2000s. GridAI brings grid management into the modern era with a responsive interface that works seamlessly across control room displays and tablets. While old systems crash under pressure, GridAI's Deck.GL and Apache ECharts technology handle thousands of grid nodes seamlessly. With features branching beyond that, the Live Code Editor and Market Dashboard are chief examples of advancements in their own right. The Live Code Editor is a Google Docs-like editor with a live chat and comments, and the Market Dashboard is the first of its kind in a blossoming industry, on top of the grid management already included. In today's complex energy environment, GridAI provides the modern, reliable tools that power professionals have been waiting for.

4.2 Design Exploration

4.2.1 Design Decisions

To ensure a smooth and responsive UI/UX for GridAI users, the team has implemented a front-end enhancement that stems from several design decisions after analysis of users' needs and requirements. Through this analysis, design decisions were narrowed down to three primary areas listed below. These changes aim to enhance user experience and guide the project to success.

1. Rehaul WebSockets to utilize Conflict-free Replicated Data Type (CRDT) friendly sockets

Currently, the project is built on socket.io. However, these sockets are not friendly to multiple users making active changes, and can lead to issues. To have an effective live collaboration environment, it must be able to deliver all live time changes instantly to all online users.

Key Requirements:

- Must transmit CRDT operations instead of raw text changes
- Multiple users should be able to edit concurrently without conflict
- Reliable real-time syncing
- Awareness of other users (such as track and display cursors, selections, and presence)

2. Migrate away from Mantine UI

Currently, the project is built on the Mantine UI for components and visuals. However, through analysis, Mantine UI does not perform as well under high load and can take time to bundle. It is critical for the project's live data displays that there is an efficient and well-established UI.

Key Requirements:

- Lightweight bundle size to reduce initial load times

- Compatibility with TailwindCSS for design consistency
- High customizability to adapt UI components
- Support for real-time dynamic content

3. Optimize Front-end for Large Scale Data Visualization

GridAI's core purpose is to visualize and manage electrical grids. The application must efficiently handle and visualize large datasets of over 50,000 grid nodes without lag. The front-end design must ensure operators receive quick, accurate information under a heavy system load.

Key Requirements:

- Support real-time updates to the grid without degrading performance
- Clear visibility and controls for grid operators
- Improved visualization capabilities for complex grids
- Efficient handling of data streams

4.2.2 Ideation

The lotus blossom technique was used to explore other UI libraries outside of MantineUI that better suit the project and users' needs. Various UI libraries were considered that align with the team's objectives of project performance, flexibility, and design consistency.

Five potential options were identified:

1. Material UI

Widely used component library offers:

- Designed for React
- Easy integration
- Pre-built components
- Customization
- Optimized performance

2. ShadCN UI

A modern UI toolkit built on top of Radix and UI offering:

- Lightweight components emphasizing performance
- Styled with TailwindCSS with complete design freedom
- Ideal for maximum customization
- Larger UI won't be bogged down

3. Headless UI

A utility-driven library of components designed for:

- Design control through TailwindCSS or custom CSS
- Accessible-by-default interactive elements

- Clean separation of logic and presentation

4. Radix UI

A low-level UI primitive library that emphasizes:

- Strong community and regular updates
- Unstyled accessible components for building design systems
- Advanced interactions out of the box
- High integration with React hooks

5. Subframe UI

A visual-first UI builder offering:

- Drag and drop building for prototyping
- Figma-like interface
- Code export for seamless integration into React projects

4.2.3 Decision-Making and Trade-Off

To ensure the best UI for GridAI's users and project requirements, a weighted decision matrix was developed incorporating key criteria categories important to the project. Each UI was analyzed against the following factors:

Evaluation Criteria:

- Performance (25% weight)
- Development Speed (25% weight)
- Customizability (20% weight)
- Maintainability (15% weight)
- Out-of-box features (15% weight)

Criteria	Weight	Material	ShadCN	Headless	Radix	Subframe
Performance	0.25	3 (0.75)	5(1.25)	3(0.75)	5(1.25)	2(0.5)
Customizability	0.20	2(0.4)	5(1)	5(1)	4(0.8)	2(0.4)
Maintainability	0.15	4(0.6)	4(0.6)	3(0.45)	3(0.45)	1(0.15)
Out-of-box features	0.15	5(0.75)	4(0.6)	1(0.15)	2(0.3)	4(0.6)
Development Speed	0.25	5(1.25)	3(0.75)	2(0.5)	1(0.25)	5(1.25)
Total Score	1.00	3.75	4.2	2.85	3.05	2.9

After evaluating the decision matrix, ShadCN UI was selected as GridAI's new UI library, which achieved a score of 4.2 on the decision matrix. ShadCN offers many advantages compared to the other libraries:

Development Advantages:

- Seamless integration with TailwindCSS
- Easier developer experience with clean, modular component structure
- Improved design consistency across the application

Performance Advantages:

- Only includes what's necessary for the program to run.
- Smaller bundle sizes lead to faster run times
- No runtime styling overhead
- Well-suited for dynamic content

Long Term Advantages:

- Strong industry adoption
- Full ownership of UI components
- Adapts well to project scaling

This UI choice provides a well-established library for the foundation of the UI that ensures long-term support and better performance gains compared to other libraries. It is a popular and well-liked library used by larger companies such as Vercel. It doesn't just meet the team's immediate technical needs; it allows a foundation for more robust features to be built and maintained.

4.3 Final Design

4.3.1 Overview

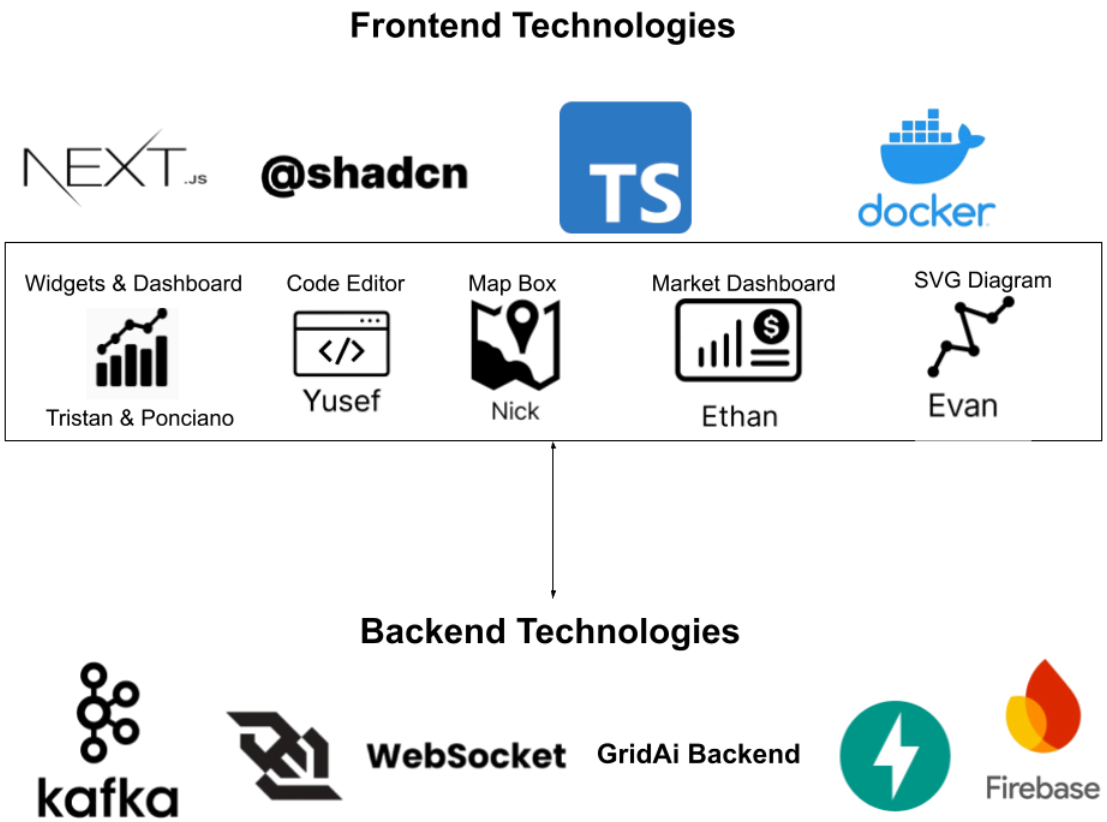

Figure 4: Full-Stack Component Technology Diagram

Widgets/Dashboard Component:

The Widgets component is one of the core services in the application, allowing users to create and customize widgets based on predefined templates. These widgets serve as visual tools for data monitoring and analysis. This component enables users to:

- Create custom widgets using a variety of templates
- Choose from different chart types, including line, bar, and pie charts
- Organize and save widgets within a personalized dashboard
- Display only selected widgets for a streamlined, user-specific view
- View specific information tailored to each widget

Market Dashboard Component:

The Market Dashboard Component is a way for Distributed Energy Resource Aggregators (DERA), Independent System Operators (ISO), and Distribution System Operators (DSO) to participate in the wholesale electricity market. This component allows:

- DSOs to manage distribution between grids and ensure reliability among them
- ISOs to manage power markets to balance supply and demand
- DERAs to optimize small-scale energy resources such as solar panels or grids

SVG Diagram Component:

The SVG Diagram Component is a visual representation of the electrical grid that the Grid is running. It is designed to make complex systems and designs easy to understand. This component provides:

- A clear layout of the components with straightforward icons
- Visualization of power flow, connections, and nodes on the grid
- Visual of electrical control and protection within the grid
- Comprehensive view for planning and analysis of the grid design
- Real-time and Past time system analysis to see grid performance over time

Map Box Component:

The Map Box component is the geospatial interface with layered maps (ReactGL for the mapping, Deck.GL for the nodes and lines) that allows the user to monitor their specific use-case energy system for GridAI. This component will enable users to have:

- An interactive visual interface showcasing the layout of their GridAI nodes
- Real-time and historical timeline of energy usage
- Energy consumption visual indicators, such as red coloring for high energy usage
- Responsive zooming and panning within the mapping feature
- Highlighting and selecting an area of nodes for real-time energy usage data of the selected nodes
- Data retrieval from the usage of Dexie.js and IndexedDB node data storage

Code Editor Component:

The Live Code Editor component aims to provide a live collaboration workspace for operators and users to edit files without leaving the GridAI interface. This component strives to:

- Support smooth, conflict-free collaboration
- Enable real-time multiple live time changes by various users
- Allow users to comment on specific lines
- Ensure changes are automatically saved to a back-end database
- Maintain user-specific context, such as cursor position and file state, across sessions

4.3.2 Detailed Design and Visual(s)

System Architecture:

The front-end user interface has a structured layering architecture design with Front-end, Integration (Middleware), Back-end services, and a Widget System to ensure a responsive and robust user interface.

Front-end Layer

- Purpose: Responsive, real-time UI
- Mechanics: Handles UI with React (TypeScript), Server Components for efficient rendering, WebSockets for live data updates, TailwindCSS, and ShadCN for consistent and responsive design
- Capabilities: Dynamic component updates, real-time data display, interactive controls

Integration Layer

- Purpose: Efficient data exchange and system optimization between front-end and back-end
- Mechanics: Manages real-time data stream pipelines, sophisticated caching, oversees server-side rendering, and handles robust security authentication
- Capabilities: High-throughput data handling, performance tuning, secure operations

Back-end Services

- Purpose: Anchors the entire system with computational power and reliable data management
- Mechanics: Leverages Firebase to ensure secure authentication and real-time database operations, InfluxDB for high-performance storage, OpenDSS for power system simulation details, and MongoDB for structured widget data. All services are used via asynchronous API calls
- Capabilities: Real-time analytics, persistent state tracking, power flow modeling

Widget System

- Purpose: Provides modular, user-configurable interface elements
- Mechanics:
    - Database Tier: MongoDB + React hooks for efficient and persistent data management
    - Back-end Tier: Provides RESTful APIs to streamline data routing and configuration logic
    - Front-end Tier: TailwindCSS + ShadCN for responsive, interactive widget rendering

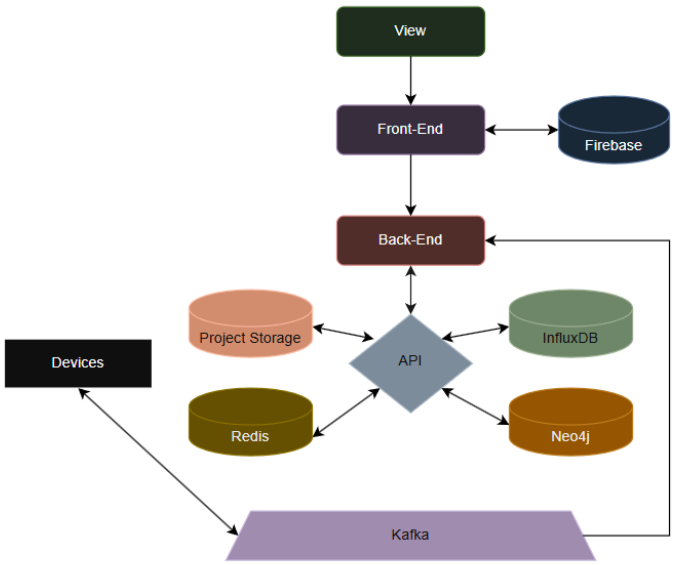

Figure 5: System Architecture Diagram

4.3.3 Functionality

The team's design is intended to operate in the real world by allowing users to visually represent their energy, whether it is for research, solar panel systems, or energy companies with thousands of energy nodes. For example, users can create their own visuals of the energy through customizable widgets and see how much energy is consumed for their specific grid system.

GridAI has many features that support its users regarding energy management. In addition to the Widget System, users can look at their data through a geospatial Map Box visual to display data based on power voltage, location, and speed. This data can also be real-time or historical. This allows for more flexibility in monitoring their grid system.

Another feature available to users is the Single Line Diagram, which displays electric systems and provides a simplified version of the map box visualization, allowing for greater flexibility in visualizing their GridAI system. This can also help users get information about specific connections between multiple nodes. Other features cater to various user types, including developers and businesses. Developers can use this application to create widgets within a code editor, giving them greater customization freedom.

GridAI's Live Code Editor allows users to write, edit, and collaborate with code. This is perfect for developers, researchers, and even tech-savvy business owners. The Live Code Editor helps simplify the process by allowing developers to address issues quickly, ensure a smoother workflow, and allow for real-time collaboration. Businesses can use this application to manage costs when it comes to energy with the market dashboard, and it will enable Distributed Energy Resource Aggregators (DERAs) to participate in the wholesale electricity market.

4.3.4 Areas of Concern and Development

The design will meet users' needs for ease of understanding and interaction, resulting in a minimal learning curve when engaging with the product. Simplicity will be incorporated into all aspects of the project to facilitate easy understanding and functionality. The primary focus is on ensuring that the product is responsive and functionally sound, with design considerations coming second to functionality.

The integration process of the design has been examined to promote seamless communication between the user and the product. The user interface must be polished and thoroughly tested to guarantee a seamless experience and clear understanding, even for newcomers or those unfamiliar with the product. User testing will be conducted to gather valuable feedback, which will inform an iterative development process. This approach will allow for growth and refinement based on the feedback received from GridAI testing. Further optimization of all components will include integration and stress testing to enhance overall performance.

4.4 Technology Considerations

Technologies being used directly (Front-end):

- React
  - Strengths
    - Component-based architecture
    - Extensive library with lots of tools and support
    - Virtual DOM for improved performance
    - Synergizes well with other technologies such as Next.js and Node.js
  - Weaknesses
    - It requires a lot of optimization
    - Steeper learning curve
    - CSR(Client Side Rendering) can hurt SEO (Search Engine Optimization)
  - Alternatives
    - Svelte - Used in simple web applications with few resources, with no virtual DOM
    - Vue.js - Great for single-page applications and easier to learn

- Mantine (In the previous version)
  - Strengths
    - Pre-built accessible components
    - Theming and customization support
    - Extensive documentation
    - Built on React
  - Weaknesses
    - Larger bundle size than headless libraries
    - Not easy to customize
    - Locked into a vendor
  - Alternatives
    - Shadcn - Expanded more below

        - Chakra UI - Very similar to Mantine

- Shadcn (In the new version)
    - Strengths
        - Headless UI + Tailwind for full control over components
        - Lightweight and modular
        - High accessibility standards
        - No vendor lock-in
    - Weaknesses
        - Steeper learning curve
        - Less out-of-the-box features
    - Alternatives
        - N/A (Same as Mantine's)

- Next.js
    - Strengths
        - Hybrid rendering
        - Built-in API routes, routing, and optimizations
        - Strong SEO
    - Weaknesses
        - Complexity increased with the NodeJS backend. (Which is present)
        - Cold Starts in serverless deployments (Delay while starting)
        - Overkill for static sites
    - Alternatives
        - Remix - Nested layouts with progressive enhancement
        - Astro - lighter island-based architecture

Other technologies in GridAI (Backend):

- Node.js
    - Strengths
        - Non-blocking I/O
        - NPM ecosystem with lots of support available
        - Unified JS
    - Weaknesses
        - Callbacks can make development cumbersome
        - Not great for CPU-based tasks
        - Lots of dependencies
    - Alternatives
        - Go - Better for performance
        - Deno - A TypeScript and JavaScript run platform with a secure runtime
- Kafka
    - Strengths
        - High-throughput event streaming
        - Highly scalable for microservices
        - Durable
    - Weaknesses
        - Complex maintenance and setup
        - Very steep learning curve
    - Alternatives
        - RabbitMQ - Better performance for simpler queues

        - AWS Kinesis - Managed streaming
- Firebase
    - Strengths
        - Real-time database updating
        - Easy to manage
        - Serverless architecture
        - Scalable for mid-sized applications
    - Weaknesses
        - It can get expensive based on usage
        - Lacking query options
        - Vendor lock-in
    - Alternatives
        - AWS Amplify - More flexibility but harder to learn
        - PocketBase - Self-hosted and lighter
- Neo4j
    - Strengths
        - Cypher language queries
        - ACID(Atomicity, Consistency, Isolation, Durability) compliant
        - Optimized for graph-based queries
    - Weaknesses
        - Less support than something like SQL
        - Hard/expensive to scale
        - Not the best for relational data
    - Alternatives
        - Apache AGE - Graph extension for Postgres
        - Dgraph - High-performance alternative to Neo4j
- InfluxDB
    - Strengths
        - Built for time-series data
        - High performance for temporal data
        - Supports SQL-like queries
    - Weaknesses
        - Not practical for more complex queries
        - It can be hard to tune
    - Alternatives
        - Prometheus - Good for full monitoring
        - TimescaleDB - PostgreSQL extension for temporal data

4.5 Design Analysis

So far, the team's focus has been on evaluating and testing the components handed down from the previous senior design team. We've worked to understand what functions correctly and what don't, as the codebase was inherited with limited documentation. One of the team's primary objectives has been transitioning the entire UI from Mantine to Shadcn UI, a change planned to be completed across all five components by the end of the semester.

The design aims to meet users' needs for straightforward understanding and interaction, facilitating a minimal learning curve when engaging with the product. Simplicity will be integrated into all aspects of the project to ensure easy comprehension and functionality.

The primary focus is to ensure that the product is responsive and functionally sound, with aesthetic design considerations following functionality. The integration process has been reviewed to facilitate seamless communication between users and the product. The user interface must be polished and rigorously tested to ensure a smooth experience and a clear understanding, even for those new to the product.

Conducting user testing will gather valuable feedback that informs an iterative development process. This method will enable growth and refinement based on the insights obtained from GridAI testing. Further optimization of all components will involve integration and stress testing to enhance overall performance.

5 Testing

5.1 Introduction

Testing is critical to GridAI's development lifecycle due to its role in real-time power grid operations. The team's testing strategy is closely tied to the project requirements, emphasizing early, continuous, and comprehensive validation across all components and interfaces. Tests are designed to reflect both technical correctness and real-world usage.

Challenges include:

- Verifying synchronization of real-time collaboration features across multiple users
- Verifying visual accuracy for components like single-line diagrams and energy dashboards
- Simulating and validating diverse user workflows, such as residential users, energy providers, and educational users
- Testing performance and scalability during high-traffic or data-intensive scenarios

5.2 Unit Testing

The frontend of GridAI is built using React, with many components forming the user interface. Unit testing focuses on validating these components in isolation to ensure they behave as expected under various conditions. Jest and React Testing Library were the main tools used to test the components.

These tools allow us to simulate real-world usage, validate DOM behavior, and ensure UI correctness without requiring a live backend.

Focus areas include:

- Validating component props, ensuring they render the expected output
- Verifying internal state management, including user interactions and context updates
- Ensuring expected UI behavior across different user roles and data inputs
- Testing custom hooks for connection logic and event handling

Tools:

- React Testing Library for rendering components, simulating interactions, and asserting DOM output
- Jest for running test suites, mocking functions, and validating logic and hooks

5.3 Interface Testing

Interface testing focuses on verifying that communication between individual system components is reliable, consistent, and behaves as expected during user interaction and data exchange.

Interfaces Tested:

- Dashboard and Widget Communication
    - Verify that widgets are placed in their correct position within the dashboard layout
    - Ensure real-time updates between the dashboard and widgets are consistent
    - Check the handling of events for widgets, such as dragging, resizing, and deletion of widgets
    - Test if widget states are appropriately updated in the dashboard UI
- Frontend and Backend Interaction
    - Validate API requests and responses for creating, updating, and deleting users, dashboards, and widgets
    - Test proper handling of edge cases, including missing or invalid data
    - Verify status codes and error messages for failed operations
    - Ensure real-time updates are received from the backend when using WebSocket connections.
- Code Editor and Widget Connection
    - Confirm that changes made in the code editor are applied live to the associated widgets
    - Verify that syntax handling and error feedback within the editor interface
    - Test widget behavior in response to user logic injected via the editor
- Frontend and Firebase Authentication
    - Validate sign-up, login, and logout functionality
    - Test token handling and expiration logic from the client side
    - Simulate invalid tokens and unauthorized access to ensure proper redirection
    - Ensure user role affects access to certain pages and components
- Tools
    - Cypress
    - Postman
    - Websocket Test Clients

5.4 Integration Testing

The team's integration testing focuses on the interactions between the system components to ensure seamless functionality and compliance. The following are the critical integration paths and the testing strategies:

- Real-time data updates are central to the user experience since GridAI's features rely on live data, such as widgets or live code editing. The system must ensure the correct data is displayed. The testing strategy includes:
    - The system must be able to handle high traffic scenarios to ensure WebSocket connections remain stable and reconnect properly after disruptions.
    - Data integrity verifies that incoming and outgoing messages carry accurate, timely data.
    - Widgets need to ensure that they update correctly in response to real-time events.
- Authentication needs to be secure, and this governs user access. The testing strategy includes:
    - Utilizing the Firebase authentication integration by testing the sign-up, logins, logouts, and password recovery flows.
    - Verification for users with different roles, such as user or guest, can only access the appropriate data and features.
    - Ensure sessions persist appropriately and are invalidated on logout or expiry.
- The dashboard and backend integration rely on the API backends for data display and control logic. It must ensure that the endpoints are working efficiently for the best user experience. The testing strategy includes:
    - Validating the API endpoints to ensure the correct status codes and responses.
    - Handling errors through simulating failures to verify proper user feedback and fallback mechanisms.
    - Performance monitoring will measure response times and throughput under typical and peak loads.
- The tools utilized for these critical paths include:
    - Real-time Data:
        - WebSocket test clients
    - Authentication:
        - Cypress
        - Firebase Authentication Emulator
        - Postman
    - Dashboard and backend integration:
        - Postman
        - Monitoring tools like Firebase Performance

5.5 System Testing

The team's system testing strategy focuses on overall system performance and different user workflows:

User Workflow Testing (Automated using Cypress): This ensures every user interaction within the GridAI platform works as intended. The focus is on end-to-end scenarios that would mirror real-world use cases:

- DSO (Distribution System Operator) Scenarios:

    - Test workflows for grid operators managing power distribution, such as voltage stability or balancing load across regions, ensuring the UI updates correctly and data remains accurate during the test.
    - This is important because DSOs rely on precise, real-time data to make decisions and maintain grid reliability.

- DERA (Distributed Energy Resource Aggregator) Management Workflows:

    - Validating workflows to aggregate and manage distributed energy resources (i.e, battery storage, solar panels, etc) to optimize energy sales or grid contributions.
    - This is important because DERAs need streamlined processes to manage multiple energy sources, and the team's testing will help ensure the reliability for users trying to profit from energy.

- ISO (Independent System Operator) Monitoring Procedures:

    - Testing to ensure ISOs can monitor grid performance, track energy consumption, and respond to responsive energy demand spikes or other signals through the GridAI platform.
    - This is important because ISOs oversee larger-scale grid stability, so GridAI must provide reliable, responsive, and accurate information.

Performance Testing (Automated using Lighthouse): Provided the data-intensive nature of GridAI, performance testing is critical to ensure the platform remains responsive and stable during heavy usage.

- Load testing with multiple data points (tens, hundreds, thousands, etc):

    - Simulate thousands of data points (real-time energy readings, historical) testing for multiple nodes being processed on the system (MapBox highlighting functionality)
    - This is important because the entire platform was built and relies heavily on the functionality of processing large datasets quickly; slowdowns could frustrate users or delay critical decisions for essential users.

- Resource Utilization Tracking:

    - Monitoring server CPU, memory, and database usage during situations of scalability, ensuring GridAI scales accurately and efficiently.
    - This is important because efficient resource use helps keep GridAI's platform cost-effective and reliable for each user.

- Response Time Monitoring:

    - Measuring how quickly the platform responds to multiple user actions/interactions, such as loading a dashboard/widget or calculating real-time node power usage.

    - This is important because GridAI's users expect the application to work with instant or near-instant information and feedback from their requests, especially during peak energy usage or time-sensitive tasks.

Multi-device and Multi-browser Testing (Automated using BrowserStack): The system must work consistently and efficiently across devices and browsers that GridAI users could be on.

- Mobile Responsiveness:
    - Testing that the platform is mobile-friendly, adapting to smaller and vertical screens, with proper layouts, and displaying data optimally.
    - Ensure touch-based interactions (swiping through different screens, dashboards, and button interactions) are smooth and intuitive for the user.
    - This is important because many users could access the GridAI platform while on the go and expect a desktop-like experience without compromise.

- Chrome/Firefox/Safari compatibility:
    - Verify all features (node data visualizations, transactions, etc) render correctly and function identically across major internet browsers.
    - This is important because users may have their own browser preferences,and mitigating inconsistencies is a high priority for user experience.

5.6 Regression Testing

To ensure additions do not detract from the functionality currently found in GridAI and that new functionality is implemented correctly. This involves continuous regression testing and re-running previously validated components that are changing. This must be through test cases whenever updates are made to these components. Many tests are similar, if not identical, to previous testing formats, but are still essential to do after changes.
Critical features to test:

- Data visualization tools - Includes the live and historical data in the Map Box, Widgets, and Market Dashboards.
- API integration stability - Validating all new and existing uses of the backend API in the frontend and ensuring they are seamlessly integrated.
- Responsive Design - Testing all UI components to ensure they are easy to use, understand, and perform without error.
- User validation - Validating that user types get sent to the right page. This is especially important in the market dashboard and map, where separate user types see entirely different information.

Testing Approach:

- Requirement-driven testing -

- Automated regression suites - Cypress and other test suites can automate different aspects of regression testing.
- Manual verification - Mostly for API validation, directly using the backend with test calls using systems like Postman.
- CI/CD integration - Integrating CI/CD pipelines could give us automated regression testing, an elegant way to lighten the load of manual testing.

Tools and Metrics:

- Postman - Manual backend API testing allows us to see the intended outcome of an API call and see if the frontend uses it appropriately.
- Cypress - A testing service that can be manually used in a browser or automated into a CI/CD pipeline.
- Lighthouse - One of the most common UI responsiveness trackers on the market. Allows the tester to see a rating based on responsiveness, accessibility, SEO, and best practices.
- Web vitals - Another of the more common UI responsiveness trackers, reporting on how well the software performs.
- Jest - An automated testing service widely used in JavaScript and TypeScript projects.
- Copado - Another automated testing platform using CI/CD to test after every change. However, it has a potentially high cost associated with it.

5.7 Acceptance Testing

The team's acceptance testing strategy will ensure that both functional and non-functional requirements outlined in the design document are fully met. The focus is on verifying that the system from the end-users' needs to confirm the product's readiness for deployment.

Client Collaboration:

- Demonstration Sessions - Hosting walkthroughs of major testing features using real-time datasets, allowing the client to observe workflows and provide immediate feedback.
- Client Test Cases - Incorporating test cases provided by clients to validate the real-world cases
- Feedback Loops- Collecting client feedback from test client test cases and addressing concerns and growth that can be applied

Functional Requirements:

- Scenario Testing - Using real-world scenarios for testing cases across user types
- Checklist Confirmation - Using a structured checklist based on requirements to keep track of confirmed scenarios
- Hierarchy Access - To confirm that every type of user has the proper permissions for the application

Non-Functional Requirements:

- Performance - Confirm that the application is reaching performance goals and benchmarks while it is running
- Accessibility - Making sure that the application is usable across multiple platforms and devices
- Reliability - Test the application for extended periods to ensure that it can remain stable

Tools:

- Postman - Validate workflows and data through API testing

- BrowserStack - Verify that the application runs on multiple platforms of devices
- Client Feedback - Take in client feedback and apply it to testing and application
- Cypress - Use this for functional testing

5.8 Results

The team is still in active development, but several testing efforts are already underway. For unit testing, the team is finalizing a CI/CD pipeline and creating test cases for each component so that code cannot be merged unless it meets the team's mutual baseline criteria. On the interface side, the frontend and backend communicate correctly, and the frontend authenticates with Firebase as expected. Work is ongoing to ensure the code editor updates its associated widgets reliably. For integration testing, Postman is used after each major change to confirm data flows smoothly and remains consistent between the frontend and backend. System-level testing has not started yet, but it is planned to capture the performance metrics needed to confirm the application is fast and reliable. Regression is covered by shared Postman collections that are run after every sprint, and those tests are being expanded to guarantee new updates coexist with existing code. Finally, full acceptance testing will happen once the application is more mature and real data is available.

6 Implementation

Widgets:
The Widget component utilizes a modular React architecture, enabling flexibility and customization of monitoring data. The widgets operate independently of one another, accessing real-time data and managing their own states. Widgets can access data by adding a node key from Kafka; they can be subscribed to or switched out. The key provides the data for the widget visually, and measurements can be changed from voltage or power, with fields such as kV for voltage and p and q for power. There is a live data inspector that users can access to view the data, including the measurement time and value.

Key Features:

- Dynamic Visuals: Real-time data and customization.
- Live Data Inspector: Displays live information for the widget.
- Responsive Design: User-friendly UI and fast responsiveness.

Real-time data updates are powered by a WebSocket, ensuring a seamless backend connection, efficient data caching, and optimized performance. This allows for the display of data through the data inspector, which shows all live data for the widget in a table.

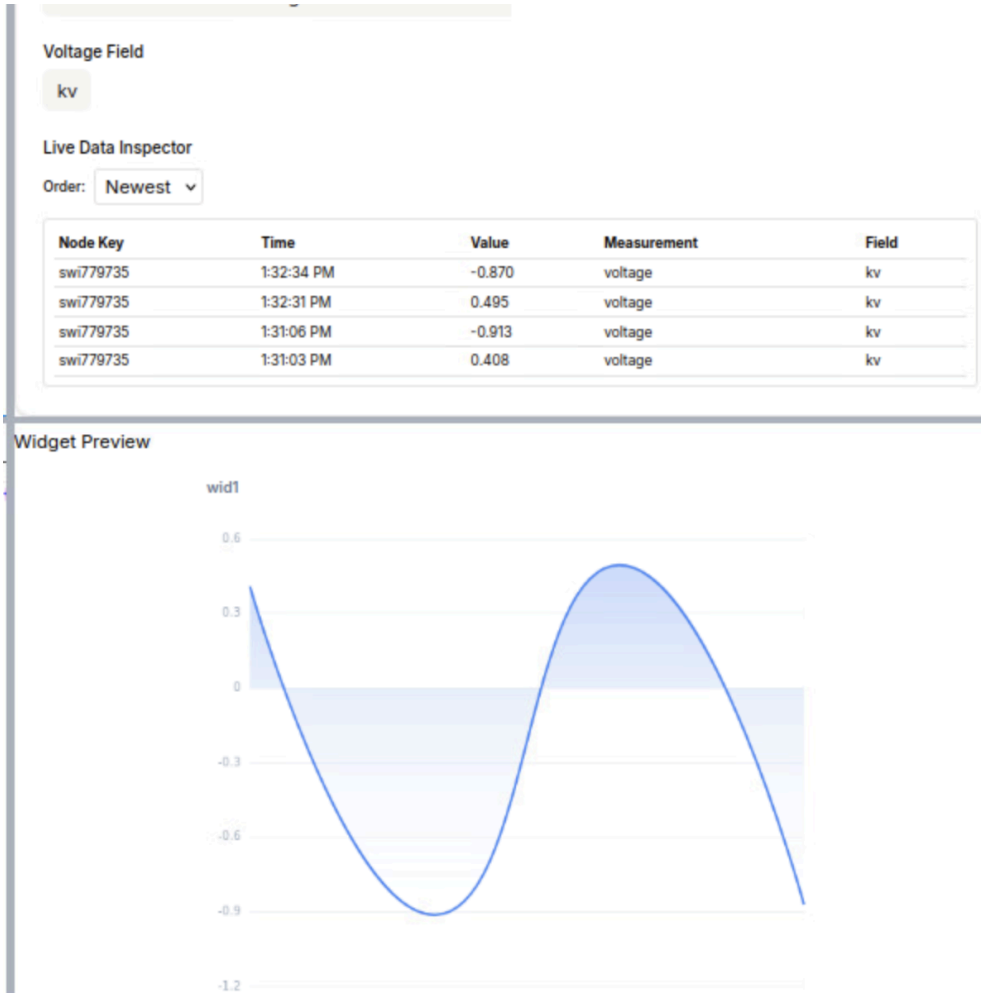

Figure 6: Widget Editor Showcasing Data

Widget Dashboard:
Users interact with a fully customizable dashboard with widgets tailored to their needs. The system remembers their preferences and loads them upon login.

The Dashboard component utilizes a React architecture similar to the widgets. This allows for flexibility for the dashboard. The dashboard is where users can display multiple widgets in a grid, each showing different types of data. The widgets can be resized and moved around to tailor them for the user's needs. Widgets can be edited and removed from the dashboard.

Key Features:

- Dynamic Visuals: Real-time data for multiple widgets.
- Customization: Users can select different widgets and position or resize them
- Responsive Design: User-friendly UI and fast responsiveness.

Figure 7: Dashboard Showcasing Multiple Widgets

The layout of the dashboards is automatically saved and persists with each page visit, and is specific to each user.

Live Code Editor:
The Live Code Editor component uses a modular, VS Code–inspired architecture that enables real-time collaboration, file management, and project editing within GridAI. The editor provides users with an intuitive environment for writing, organizing, and updating project logic. Users can interact with project files, create comments, view chat history, and build their project directly inside the editor.

Key Features:

- File Management: Create, delete, rename, and edit files within the project.
- Built-in Chat: Displays session-wide chat for real-time communication between users. Only displays the previous 24 hours of message history.
- Comments System: Users can leave comments tied to specific lines or selections for future reference or to share with other users.

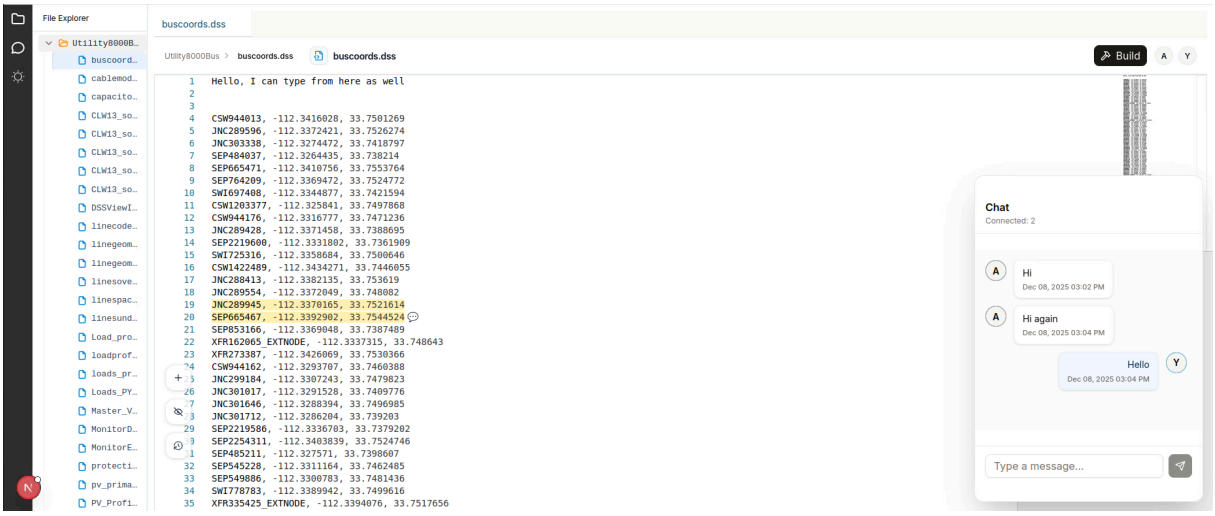

Figure 8: Live Code Editor

Market Dashboard:

The Market Dashboard utilizes a lightweight tri-platform setup, featuring a separate dashboard for each of the main three user types (ISO, DSO, and DERA), referred to as tenants. This enables a seamless workflow from bid creation to completion, with everyone involved able to access accurate analytics. Tenants are directly associated with each other to ensure a seamless and consistent flow of bids.

Key Features:

- Tenants: Hierarchical tenant structure with association between tenant types
- Bids: Bids created by DERAs can be reviewed by DSOs and ISOs, with either a rejection or a counter, along with all necessary analytics to make informed decisions.
- Analytics: Each platform has a profile of analytics that is most relevant to that user type. This can include capacity, number of tenants, and feasibility.

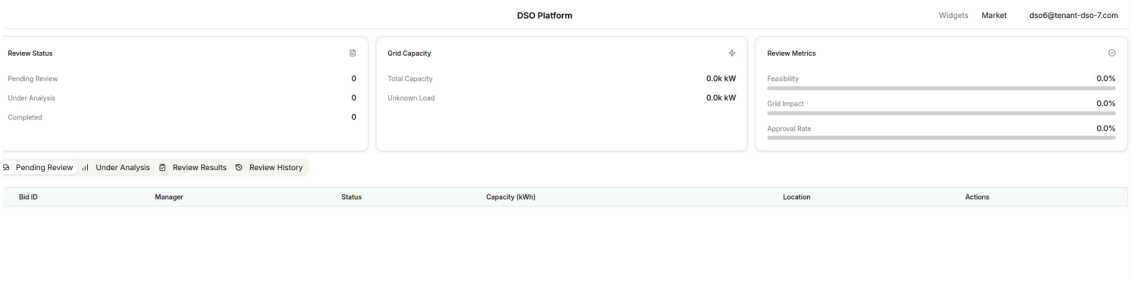

Figure 9: DSO Platform Page

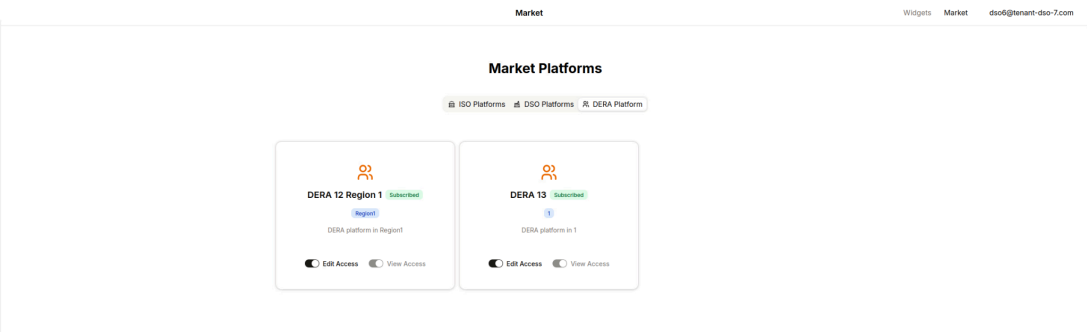

Figure 10: Market Platforms Page

Single Line Diagram:

The Single Line Diagram is a tool used to visualize the electrical system layout of the grid. It enables dynamic circuit rendering, real-time status, and customizable diagram figurations. It allows the user to save and switch between multiple display modes, supporting system diagnostics.

Key Features:

- Dynamic Visuals: Specialized icons for each symbol of the graph, making it easy to distinguish what is on the grid
- Customization: Allows users to customize icons and add to the grid when something new is implemented

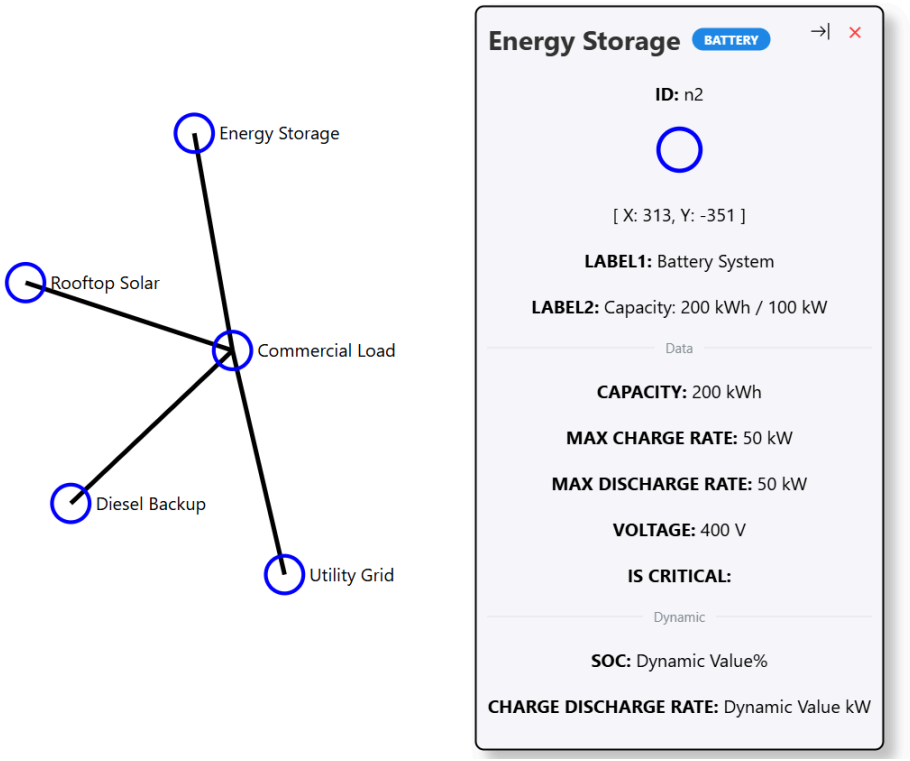

Figure 11: Single Line Diagram

Map Box:
The Map Box component is a grid visualization tool developed to allow multiple types of users to view multiple types of bus nodes on the map box grid view.

This Map Box grid view page was designed to provide scalable grip map visualization for customers with potentially hundreds of thousands of different bus nodes. The Map Box grid view has a panel on the left, which allows users to choose which types of nodes to view, such as Node (Bus), Lines, Solar, EV, Transformers and Loads. The Map Box grid view has a System Overview panel on the right with a collapsable and expandable feature, that displays electrical grid data for specific types of nodes and allows users search for specific nodes in their inventory, and display grid data for that one specific node. The collapsing System Overview panel provides users more map viewing options.

Key Features:

- Updated Figma icons for multiple types of nodes, useful for future expandability.
- Transformers and Loads API endpoints developed, tested and ready to go for future use.
- Collapsable and expandable System Overview panel for more user viewing options.

Figure 12: Map Box Component Grid View

7 Ethics and Professional Responsibility

Developing the frontend of GridAI requires a high level of ethical and professional responsibility due to its role in power grid operations and its broader impact on utility providers, the environment, and communities. The team takes these responsibilities seriously, ensuring the design and development choices consider the needs and well-being of GridAI's users. The team recognizes the importance of building software that is secure, reliable, accessible, and aligned with sustainable energy principles. The following sections will provide further elaboration on how the team incorporates these responsibilities into the software development process.

7.1 Areas of Professional Responsibility/Codes of Ethics

Area of Responsibility	Definition	Relevant IEEE Code of Ethics Principle	Team Application
Work Competence	Completing a job with skill, honesty, and timeliness, maintaining a high standard of professionalism.	"To improve the understanding of technology, its appropriate application, and potential consequences."	Each member is assigned a component to seek expertise in and share with the team. The team conducts weekly sessions to share information and status.
Financial Responsibility	Provide valuable products and services at a fair and affordable price.	"To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, and to properly credit the contributions of others."	The team ensures to disclose any uses of monetary cloud resources and continues to cut down on any unwanted costs by optimizing processes.
Communication Honesty	Share information clearly and truthfully, without misleading others, and make sure it's easy for everyone involved to understand.	"To be honest and realistic in stating claims or estimates based on available data."	Provide clients and stakeholders with accurate project updates, address any challenges faced, and communicate updated timelines.
Health, Safety, Well-Being	Take steps to protect the health and safety of all who are affected by your work. Including users and developers.	"To avoid injuring others, their property, reputation, or employment by false or malicious action."	Consider developing around the safety of the operators and recipients of grid energy. Test edge cases to prevent harmful outcomes and comply with IEEE standards.
Property Ownership	Respect the rights, ideas, and data belonging to clients and other users.	"To avoid unlawful conduct in professional activities, and to reject bribery in all its forms."	Any libraries not created by the team are referenced and follow open-source licenses.
Sustainability	Use resources wisely	"To improve the	Ensures safety

	and work to protect the environment on both local and global levels.	understanding of technology, its appropriate application, and potential consequences."	measures are in place to not overconsume electricity and to optimize usage to help the environment.
Social Responsibility	Create solutions that help improve communities and make a positive difference in society.	"To treat all persons fairly and not to engage in acts of discrimination."	Designed to benefit users and communities by saving electricity and optimizing usage in times of high demand.

The team consistently demonstrates strong communication and honesty throughout the development of the software. Especially for setbacks, which there were many. Through these setbacks, it was imperative that the team update the project timeline with realistic expectations of what can be completed. The team meets regularly, once a week, for two hours, excluding meetings with the client. During this meeting, each member shares their progress, setbacks, and updated timeline for that week. Any general objectives are addressed during this meeting, and updated goals for the upcoming week, along with action items for each member, are provided. This has fostered an environment of honesty, helping the team accurately set milestone dates for features.

The team could use improvement in property ownership. Given that the software is still in development and new features are being added, some references to the software may not be accurate. This is something the team will improve upon and include within the development process to ensure that proper documentation and credit are given to any project maintainers of these libraries.

7.2 Four Principles

	Beneficence	**Nonmaleficence**	**Respect for Autonomy**	**Justice**
Public Health	Enhances grid reliability, contributing to more stable communities	Lowers risks for grid operators through timely and precise data	Offers flexible, operator-driven data interfaces for better control	Ensures all users have fair access to a dependable energy infrastructure
Global/Cultural	Accommodates a wide range of users and device environments	Promotes interoperability across platforms and geographical	Values the choices made by the operator and maintains	Delivers equitable benefits to users across various

		regions	data confidentiality	locations and roles
Environmental	Supports sustainable practices and the integration of green energy sources	Reduces environmental impact through a resource-conscious design	Provides environmentally considerate settings	Encourages long-term sustainability through responsible monitoring and oversight
Economic	Minimizes downtime, improving efficiency and output for energy providers	Prevents unnecessary economic strain on users and operators	Delivers affordable, scalable solutions tailored to different user needs	Enables balanced distribution of market data

One context-principle pair that is particularly important to the team is public health and justice. GridAI plays a crucial role in ensuring reliable access to electricity, which has a direct impact on public safety, healthcare services, and overall community stability. By building a well accessible platform that is responsive, the team aims to deliver equal access to dependable grid management tools to all users. To ensure this, a role-based dashboard and features will be developed to provide tailored support for each user's class.

One context-principle pair that is lacking within the team is global/cultural respect for autonomy. The goal is to support a wide range of users. However, GridAI's current design isn't fully accommodating to regional UI preferences, language preferences, and deeper cultural accessibility considerations. Moving forward, as certain UI components finish the bulk of base implementation, it is important to integrate tools to support a broader user base.

7.3 Virtues

- **Team virtues are important to the team**
    - **Commitment to quality**
        - The team has committed to a high standard of work through a research-driven implementation. Ensuring rigorous testing is done, including user-driven testing, is crucial to maintaining good quality with respect to the user
    - **Frugality**
        - GridAI emphasizes conserving and optimizing users' energy; the team is mindful of resource use, as well as generating UI by using lightweight libraries and optimizing data fetching.
    - **Justice**

        - When developing GridAI, importance of all users was extremely important, regardless of technical role, being able to have equal access to the platform
- **Individual Virtues**
    - **Yusef**
        - **Virtue Demonstrated: Honesty**
            - Honesty is a critical virtue to Yusef because it helps build trust and ensures that communication is clear, respectful, and productive.
            - Being transparent with the team about challenges he faces and providing honest feedback during meetings is how he demonstrates this. Yusef also makes sure to keep consistent updates on my progress
        - **Virtue Not Demonstrated: Moderation**
            - Moderation is important because it allows for a more balanced workflow and prevents burnout. In a team setting, it ensures that no one is overextending themselves.
            - Moving forward, Yusef plans to manage his time more intentionally, set boundaries for each task, and adhere to project priorities.
    - **Nick**
        - **Virtue Demonstrated: Leadership**
            - Leadership is important to Nick because he enjoys getting things done. He led multiple team projects in the past and this senior design project is a culmination of those projects throughout his undergraduate career at Iowa State.
            - Priding himself on being a good leader is something he strives for and has helped the team through consistent planning, communication and implementation. His efforts in helping the team align towards one goal, or multiple goals (per component) the team has successfully completed each weekly and monthly task of the project and has done a good job staying on track.
        - **Virtue Not Demonstrated: Wonder**
            - Wondering is important because you need to be curious about why you are doing something and have a desire to understand it, especially in engineering.
            - Diving deeper into the component would have been helpful, but there were various tasks such as weekly deliverables, planning tasks and staying on track that it was hard to wonder about how the component works and how to improve it.
    - **Evan**
        - **Virtue Demonstrated: Simplicity**
            - Simplicity is beneficial to working on a complicated project as it helps focus and efficiency. Keeping the work organized, breaking it into smaller tasks to not over complicate what had to be done.

            - When work was kept simple, it improved time efficiency and management, made it easier to work towards goals that were laid out, and made good progress on the work that was done
        - **Virtue Not Demonstrated: Endurance**
            - Endurance is important as it involves maintaining a consistent effort of the course of the semester and that was not demonstrated, as the work was done more in bursts.
            - Endurance throughout the semester would improve progress in the project and make it easier to adjust to what needs to be done on a week to week basis
    - **Ponciano**
        - **Virtue Demonstrated: Determination**
            - Determination is important because it shows how hard someone is willing to work to get something done and make sure their end of the work is completed. It also shows how persistent you are in not getting discouraged and letting an obstacle get in the way of completing your goals.
            - Ponciano showed determination all semester, despite all the roadblocks in making sure that his work was done to the best of his ability. He was always willing to make sure that he gave his best effort when faced with any task and didn't let the major obstacles the team faced stop him from working hard.
        - **Virtue Not Demonstrated: Leadership**
            - Leadership is important because it shows who is willing to go a step further and make sure that the team is accomplishing their goals. A good leader also makes sure that their team is on the right track and is someone that will make sure to help their team if any blockers come their way.
            - Leadership was not shown as much on Ponciano's end because it's a skill that he always struggled with showing in the past. However, throughout the semester, more of his abilities to stay on top of tasks for his team and willingness to help others was better demonstrated.
    - **Tristan**
        - **Virtue Demonstrated: Respect**
            - Respect is an important virtue since working on a team. Ensuring that each team member is treated with respect and that the team gets along with each other to ensure the work is done effectively with little to no arguments.
            - Respect was demonstrated by getting along with teammates and respecting their time and efforts while working on the project. Ideas would be talked and there would be agreements or agree to disagree with not shut downs.
        - **Virtue Not Demonstrated: Determination**
            - Determination is important because it shows the will to work on a project or task and shows that the team wants to complete it and get to the end with good results.

            - Determination during this semester was not easy due to the end and issues with receiving the codebase. It did not help that there was an internship and other classes while this project was ongoing which caused issues on finding determination. Having determination would be useful when working on this project as it would have helped in completing the tasks and development.
    - **Ethan**
        - **Virtue Demonstrated: Endurance**
            - Throughout the semester, Ethan was pretty consistent with the amount of time he was putting into the project, attempting to average about 5 hours a week, trying to be as thorough as possible with my understanding of everything related to my component.
        - **Virtue Not Demonstrated: Industry**
            - Industry as a virtue mostly represents not losing time or being efficient with said time. It involves not getting hung up on irrelevant issues and getting distracted.
            - Efficiency was not the easiest thing to achieve this semester, whether it was the late-arriving ingest waves or the chaos brought on by other classes, it was hard to find the motivation to work on the assignments efficiently. Having an industry would have helped significantly when continuously working on the project, potentially enabling the market dashboard to reach a release-ready state. However, hindsight is 20/20, and achieving this goal would have been very difficult.

8. Conclusions

8.1. Summary of Progress

The GridAi Frontend represents an advancement in the power grid management interfaces. By addressing the user needs and utilizing modern web technologies, the team has managed to add more to the project by implementing new features to the frontend portion of the application. These improvements included an efficient user interface to help enable grid operators to utilize the tools to monitor and respond to the real time conditions with precise accuracy. As development concludes, the project has made significant progress and has left a foundation for future teams to build and continue.

8.2. Value Provided

The team's contributions greatly assisted in enhancing GridAI's accessibility, functionality, and extensibility. Key contributions include:

- Real time data streaming for widgets using Kafka
- Dynamic dashboards to showcase multiple sources of data using widgets
- Developed a code editor with real time collaboration
- Market dashboard that supports multiple users (ISO, DSO, DERA) to participate in the wholesale electricity market
- Implemented a dynamic mapbox interface with React-Map-GL and Deck.GL for visualization of bus nodes and lines
- Developed a diagram for showing visualization of distribution grids
- Refined the user interface with ShadCN and Tailwind

8.3. Next Steps for Future Developers

To continue building upon and enhancing GridAI, it is recommended to any following developers to address the below in the continued implementation of GridAI:

1. Live Code Editor:
    a. Implement Yjs Websocket. This adds a CRDT implementation to live time editing, which is a more efficient way to sync changes. Even if they are offline and then coming online and syncing.
    b. Introduce richer user awareness features such as live cursors, selections, and presence indicators.
    c. Improve scalability of multi user editing beyond the current basic implementation.
2. Map Box:
    a. Implement Loads and Transformer map functionality through the use of the API data endpoints developed and tested this semester. When that data becomes ready from the backend, it will be ready to use and displayed on the map using the APIs.
    b. Improve scalability of large dataset retrieval through Dexie.js optimizations.
    c. Enhance the UI viewing of the System Overview panel via shadCN/Tailwind CSS.
3. Market Dashboard:
    a. Revamp bid logic to compensate for tenant changes.
    b. Discard old files from the MantineUI versions.
    c. Enhance authentication token storage to be more secure.
    d. Improve modularity to be able to add new tenant_types as needed.
4. Widgets:
    a. Add in the historical data inspector
    b. Showcase between multiple streams of data
    c. Implement in customization settings
5. Dashboard:

    a. Make the sizing friendly for multiple resolutions
    b. Add in more customization of widgets
    c. Implement default grids
    d. Allow dashboards to be shared
6. Single Line Diagrams:
    a. Implement more Map customizations
    b. Implement diagram to main branch
    c. More customization for adding onto a growing diagram

9 References

[1]"Grid Management Solutions with Spectrum Power," *Siemens USA*, 2025. https://www.siemens.com/us/en/products/energy/grid-software/operation/grid-control.html (accessed May 05, 2025).

[2]"Enel's Ricardo Pérez Sánchez on Actions for the Planet | Schneider Electric," *@SchneiderElec*, 2019. https://www.se.com/us/en/work/campaign/innovation/energy.jsp (accessed May 05, 2025).

[3]thingsboard, "ThingsBoard - Open-source IoT Platform," *ThingsBoard*, 2017. https://thingsboard.io/

10 Appendices

10.1 Appendix 1 – Operation Manual

Important

- These commands are also in the README inside the root of the project.
- Please run this command every couple of weeks, Docker does not have automatic garbage collection, this can result in your disk storage depleting if not cleaned of old containers and images.
    - `docker system prune` (normal clean, won't remove latest containers or built images)
    - `docker system prune -a` (for full, deep clean including latest images and containers)

Project Setup

- Execute presetup script passing the full path to your project root and follow prompts:
    - `./scripts/presetup.sh [proj path]`
- Make sure you have Docker and it's related packages installed:
    - `sudo ./scripts/setup.sh`
- Restart your system, this is necessary to add your user to the Docker group and allow you to use it without root:
    - `sudo restart now`
- Test that Docker is installed correctly and can be run without root:
    - `docker -v`
- Additionally, there are more Docker Compose commands in the README for the project:
    - **Docker Compose**
        - To build and start docker system:
            - `docker-compose up --build`
        - To build and start docker system in detached mode:
            - `docker-compose up --build -d`
        - To stop docker system and remove orphaned images:
            - `docker-compose down --remove-orphans`
        - To build docker image:
            - `docker-compose build`
        - To start docker containers:
            - `docker-compose up -d`
        - To shut down docker image:
            - `docker-compose down -v`
    - Note: if you see a Container Config error after building, a quick fix is to shut down with `docker-compose down` and restart with `docker-compose build`

Using the Live Code Editor

Use the following as a guide to assist in using the collaborative code editor component.

1. Entering the live code editor

- Upon application startup and log in, you'll be greeted with the homepage.
- On the home page, find the project you'd like to bring into the code editor and select "Show Files."
- You are now in the collaborative environment; any other users who join this shared project will automatically be loaded into the same room.

2. How to use comments

- Select the "+" icon on the left-hand side to create a comment on the currently highlighted text.
- On the left side, select the history icon to view all comments.
- To delete or resolve a comment, you can select it from the history menu or by selecting the comment icon by the highlighted text.

3. How to use chat

- Upon opening the application, click on the chat bubble icon on the left-hand side.
- If 2 or more people are in the session, the chat will open, and you can start typing, as well as see the previous 24 hours of chat history.

4. **Other basic functions**

- Select a file and begin editing it as you would a file in any IDE.
- Click the “Build” button in the top right to build the project.
- Right-click on a file to delete it or right-click a folder to make a new file.
- All changes are auto-saved as you write.

Using the Map Box

Use the following as a guide to assist in using the Map Box component.

1. Ensuring the Master DSS File is Built

- If the project is correctly compiled, you are currently on the home2 menu.
- Before you can use the Map Box, you must ensure the Master DSS file is built.
- Navigate to the Code Editor (“Show Files”) page under the Project/Grid Name you choose.
- Under File Explorer on the Code Editor page, select the Utility8000 folder.
- Locate the Master DSS File and ensure it is selected.
- Click “Build” on the upper right of the Code Editor page.

2. Switching to Map Box / Grid View

- Once the Master DSS file is built, you will be able to view bus nodes and lines on the map.
- Revert back to the home2 menu, click the “back” button (<-).
- On the same project you built the Master DSS file, click on the “Show Grid” button (map).
- This will take you to the main Map Box / Show Grid page.
- If your Master DSS file was built successfully, you will see green success indicators for Bus nodes, Lines, Transformers and Loads (Success 200 messages for fetching data).

3. Viewing Bus Nodes and Lines on the Map

- The user should now be successfully loaded onto the Grid View.
- To view Bus Nodes, click the “Nodes” checkbox.
- To view Lines, click the “Lines” checkbox.
- You can zoom in and out accordingly to view bus nodes in greater detail.

4. System Overview Panel

- On the right hand side of the Map Box / Grid View page, locate the System Overview panel.
- The user can search for a specific bus node with the name or ID of the node.
- Upon searching, the System Overview panel will display grid data details for that node.
- For convenience, if the user wants to see more of the map, click “Collapse” at the top right.
- Likewise, if the user wants to see the System Overview panel again, click “Expand.”

Using the Market Dashboard

1. Creating a tenant profile

- A user must contact an administrator to become a tenant (ISO, DSO, or DERA), to access the market dashboard
- The user will be associated with other tenants for the bid workflow.

2. How to create and submit a bid (DERA)

- Select the “Create bid” button underneath the region’s metrics.
- Fill out the form that appears regarding the DER wanted. Including Manager, capacity, service plan and more.
- Press “Create Bid” at the bottom of the form to complete bid draft creation.
- Review the specifications of the bid and press “Submit bid” just to the right of “Edit Bid” in the “Day ahead bids” table.

3. How to approve bids (DSO, ISO)

- Select bid from the table of submitted bids underneath the region metrics.
- Open the specifications from the bid and review everything necessary to validate the bid.
- At the bottom of the form, select “Approve build” or “Counter bid”.
    - If a bid is approved as a DSO, the bid gets sent to ISO for them to approve.
    - If a bid is approved as an ISO, the bid goes back to DERA for final changes or full approval.
    - If a bid is counter, new specifications are given to the DERA to either approve or counter back.

Using Widgets

1. How to create a widget

- Go into the widgets page and select the widget library in navigation.
- Click on the add widget button.
- A form will then allow the user to create their widget.

2. How to edit a widget

- Click on the edit button for a widget.
- The edit page will allow the user to edit the following:
    - Template code.
    - Controller script.
    - Widget title and description.
    - Node key, measurement type, data field, and data order.
- Can update changes by clicking the update changes button.
- Save changes by clicking the save changes button.

3. Subscribing to live data

- Input the valid node key in the node key field.
- Subscribe in the widget editor.
    - This updates the widget with the real time data.

4. Changing node key

- Input the new valid node key in the node key field.
- Click the switch key button.
    - This disconnects the previous connection.
    - Reconnects to the new node key.
    - The live data updates.

5. Unsubscribing from the node key

- Click the unsubscribe button.
    - This disconnects the connection.
    - The history gets cleared.
    - Widget stops receiving live updates.

6. How to delete a widget

- To delete a widget, make sure the user is on the widget library page.
- There is a delete button that the user can use to delete the widget.
- Deletion removes the widget from the database and the UI.

Using the Widget Dashboard

1. How to create a dashboard

- Go into the widgets page and select the dashboard in navigation.
- Click on the add dashboard button.
    - Users can create one from scratch which they have a form to fill out to create their dashboard.
    - Users can import an already existing dashboard.'
- To access the dashboard, the name of the dashboard is a clickable link that goes into the dashboard.

2. Using the dashboard

- There is a grid where all of the widgets can be placed and viewed.
- On the top of the grid, there are four buttons.
    - A back button that returns to the dashboard page.
    - An export button to export the dashboard as a JSON file.
    - A drop down menu to switch between dashboards.
    - An edit mode to toggle between view and edit modes.

3. Editing the dashboard

- Make sure there are already existing widgets.
- Toggle edit mode in the dashboard.

    - There is now an add button to add from a list of widgets.
    - Widgets can be placed down, moved around, and changed sizing.
    - Widgets if hovered can be edited or removed from the dashboard.
- Edits are automatically saved and loaded upon each visit.

Using the Single Line Diagrams

1. How to load a diagram

- Go to Single line diagram page
- Click on select graph, and choose what graph you want to visualize
- Press run and it will load the graph

2. How to add an Icon

- Create your customized Icon that you want to be added to the graph
- Upload it to the nodeicon page and make sure it is an SVG file
- This will save it to the system where the user will be able to add that icon to the grid and that into the diagram

10.2 Appendix 2 – Other considerations

Map Box - New Figma Icon Mapping Directions

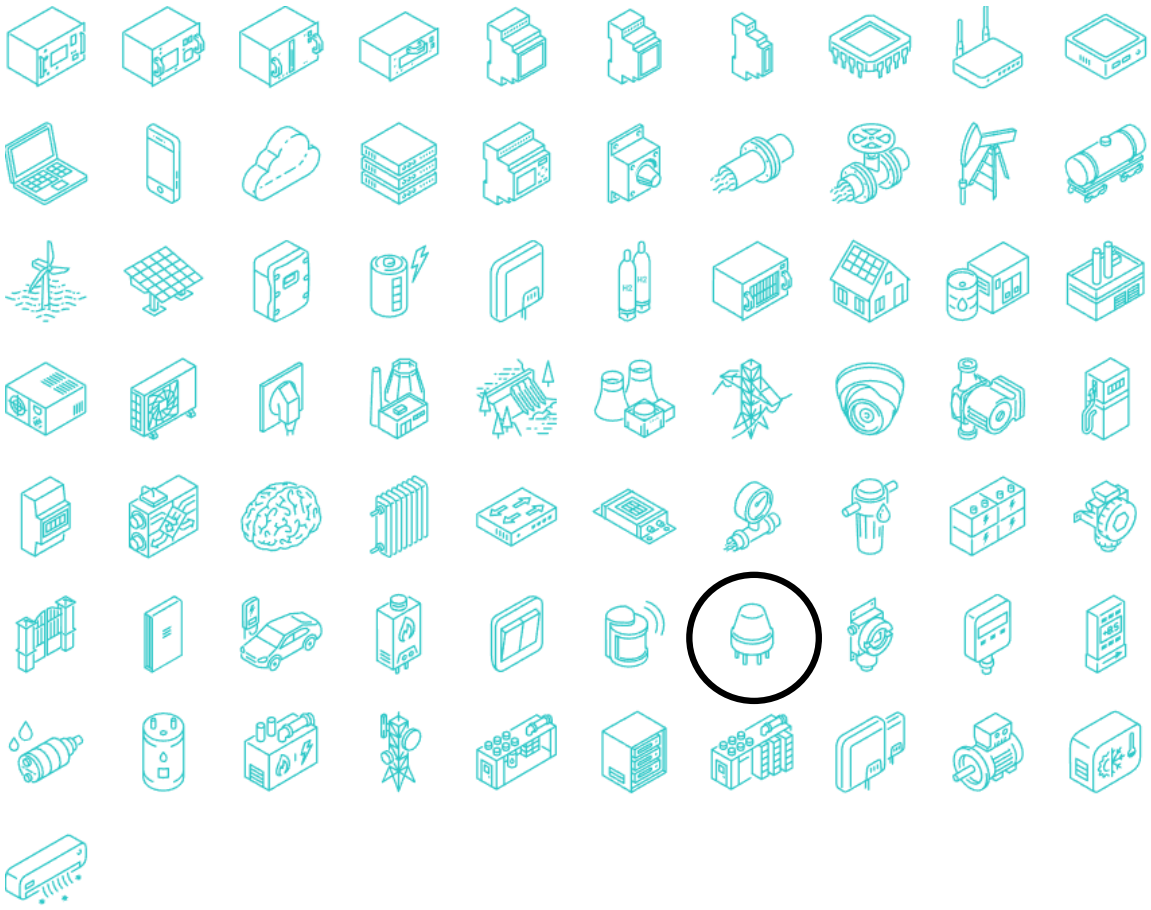

Figure 13: 71 High Quality Figma Icons for Map Box Expandability

If a future developer wants to render (with Deck.GL) a new Figma icon for a new type of bus node on the map using the SVG image above, here are the instructions with an example.

- The SVG image is a rendering atlas in the `iconLayer` const within DynamicMap2.tsx:
    - `iconAtlas: "https://raw.githubusercontent.com/mccnick/grid_ai/refs/heads/main/Frame%201.svg",`

For convenience and future development purposes, below is an outline on how to create a new icon for use on the Map Box.

Important Figma info:

- Each Figma icon is 100 x 100 pixels.
- Horizontal pixel spacing is 40 pixels.
- Vertical pixel spacing is 39 pixels.
- Deck.GL renders icons at the top left corner (0,0).
- See images from Figma below confirming the pixel spacing details.

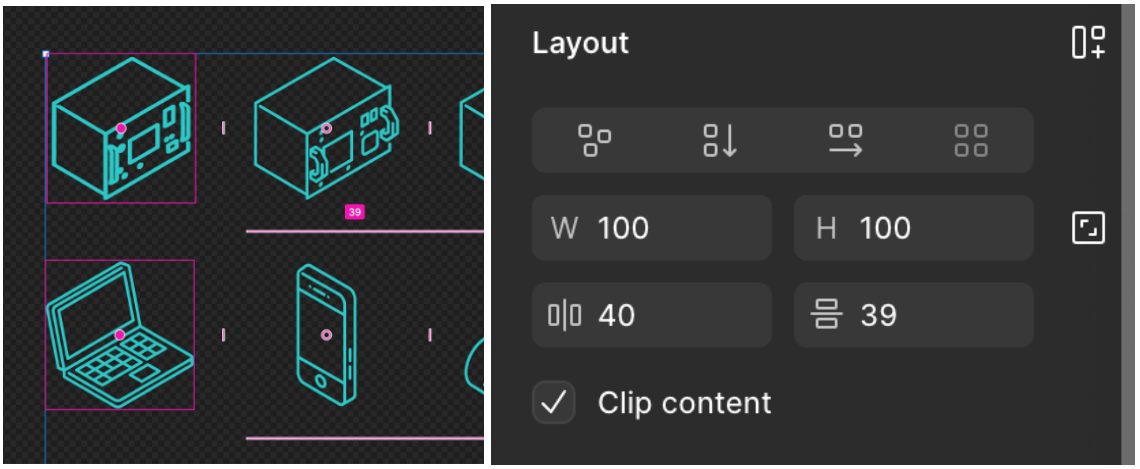

Figure 14 and 15: Figma Icon Spacing

Example:

- The current “Bus Node” in use (Figma icon circled in the SVG image above containing all 71 icons), has a location of 7 icons horizontal (across) and 6 icons vertical (down).
- X coordinate math: (100 * 6) + (40 * 6) because starting at the left of each icon is desired.
    - X coordinate = 840
- Y coordinate math: (100 * 5) + (39 * 5) because starting at the top of each icon is desired.
    - Y coordinate = 695
- This creates an X and Y coordinate value for the top left of that icon, use this same logic for a new icon of your choice and Deck.GL will render it accordingly.
- The code for this example lives in DynamicMap2.tsx (see folder layout below):
    - ```
      const ICON_MAPPING = {
            marker2: { x: 840, y: 695, width: 100, height: 100,
      mask: false }
      }
      ```

10.3 Appendix 3 – Code

GitLab Repository: https://git.ece.iastate.edu/sd/sddec25-17

Code Outline Per Component and Description of Main Code Files:

- Map Box
    - A majority of the Map Box component code lives in this folder:
        - sddec25-17/ui/frontend/src/app/showGrid2
            - ChartsComponents.jsx
                - Contains data fetchers and System Overview panel logic.
            - database.config.ts
                - Contains Dexie.js tables, constants and constructors.
            - MapContext.tsx
                - Contains important contexts and API endpoints.
      ```

                - MapProvider.jsx
                    - Contains MapProvider module for getters and setters.
                - MapTypes.jsx
                    - Contains important data formats for buses, lines, etc.
                - Page.tsx
                    - Contains important full-page functionality logic.
            - sddec25-17/ui/frontend/src/app/showGrid2/Map (subfolder)
                - ControlPanel.tsx
                    - Contains core React-Map-GL map functionality.
                - DynamicMap2.tsx
                    - Contains the React-Map-GL and Deck.GL map rendering.
                    - Contains most of the bus nodes and line viewing logic.
                - Icon-cluster-layer.tsx
                    - Contains state-selected icon layer logic.
            - sddec25-17/ui/frontend/src/app/showGrid2/SideNavigation (subfolder)
                - SideNavGrid.tsx
                    - Contains the left-hand side of the Map Box page logic.
- Live Code Editor
    - Frontend (/editPD2)
        - EditorContainer/: Core UI (Monaco + socket handling)
        - SideNavigation/: File tree + navigation
        - SideBar/: Live chat sidebar toggle
    - Backend (/fileSocket)
        - fileSocket.ts: WebSocket server logic
        - SOCKET_TYPES.ts: Shared socket event constants
- Dashboard
    - The dashboard is in this folder:
        - sddec25-17/ui/frontend/src/app/derms/Dashboard
- Widgets
    - The widgets is in this folder:
        - sddec25-17/ui/frontend/src/app/derms/widgetLibrary
- Market Dashboard
    - Most of market dashboard is in this folder:
        - sddec25-17/ui/frontend/src/app/market
    - It also has a dedicated API at:
        - sddec25-17/tenant_service/src/app.py
- Single Line Diagram
    - The single line diagram is in this folder
        - sdmay25-43/backend/src/controllers
        - ./src/endpoints
        - ./src/schemas
        - sdmay25-43/frontend/src/Components
        - ./src/Graph
        - ./src/Icons

10.4 Appendix 4 – Team Contract

10.4.1. Team Members

- Ethan Messmer
- Evan Sivets
- Nick McCullough
- Ponciano Ramirez
- Tristan Nono
- Yusef Harb

10.4.2. Required Skill Sets for Your Project

- **React & Front-End Development**: React, HTML, CSS, JavaScript/TypeScript.
- **Responsive UI/UX**: Develop intuitive, efficient, device-agnostic UIs
- **API Integration**: Familiarity with REST APIs, JSONs, and backend integration
- **Collaboration and Version Control**: Experience with Git for code management
- **Optimization and Testing**: Experience with testing tools to optimize performance

10.4.3. Skill Sets covered by the Team

- **React & Front-End Development**: React, HTML, CSS, JavaScript/TypeScript.
- **Responsive UI/UX**: Designed user interfaces for different platforms.
- **API Integration**: Connected backend systems to a frontend system
- **Collaboration and Version Control**: Utilized GitLab for the previous course, and Github for personal use
- **Optimization and Testing**: Utilized tools to test React components

10.4.4. Project Management Style Adopted by the Team

Agile Methodology

10.4.5. Initial Project Management Roles

- Team Organization: Nick McCullough
- Client Interaction: Everyone
- Test Lead: Evan Sivets
- Design: Ethan Messmer
- Widgets: Ponciano Ramirez & Tristan Nono

- Record Keeper: Yusef Harb

10.4.6. Team Contract

Team Members

- Ethan Messmer
- Evan Sivets
- Nick McCullough
- Ponciano Ramirez
- Tristan Nono
- Yusef Harb

Team Procedures

- Day, time, and location (face-to-face or virtual) for regular team meetings:
    - In person or virtual, if conflicts are present. Fridays, 1-3 pm and 4-5 pm with the client.
- Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    - Discord
- Decision-making policy (e.g., consensus, majority vote):
    - Majority vote
- Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    - Yusef will take meeting notes and record the time per meeting.

Participation Expectations

- Expected individual attendance, punctuality, and participation at all team meetings:
    - Every team member is expected to attend regular weekly meetings in person. They should try to join the Discord voice channel if they cannot attend in person.
- Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    - Every member will complete any assigned tasks promptly, not missing deadlines.
- Expected level of communication with other team members:
    - Every team member should share opinions and suggestions on the project during team meetings and/or outside of meetings. Regular communication is expected.
- Expected level of commitment to team decisions and tasks:
    - Every team member is expected to contribute to team decisions and complete tasks to the best of their ability.

Leadership

- Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
    - Team Organization/Developer: Nick McCullough
    - Record Keeper/Developer: Yusef Harb
    - Component Design/Developer: Ethan Messmer
    - Test Lead/Developer: Evan Sivets
    - API Developer/Developer: Ponciano Ramirez
    - React Specialist/Developer: Tristan Nono
- Strategies for supporting and guiding the work of all team members:
    - Frequently check in with other members weekly, ensuring no one has encountered issues/roadblocks.
    - Be willing to assist other members if they are struggling
- Strategies for recognizing the contributions of all team members:
    - In weekly team meetings, each member discusses what they contributed that week if time allows
    - Recognize and value different contributions, such as coding, creating ideas, taking on responsibilities, etc.

Collaboration and Inclusion

- Describe the skills, expertise, and unique perspectives each team member brings to the team.
    - **Yusef:** Software engineering major with hands-on experience in full-stack development across multiple programming languages.
    - **Tristan:** Software engineering major with experience working on full-stack personal projects with React (JavaScript & TypeScript), MongoDB, and Firebase. Launched an MVP for a web application and was commissioned to develop a website for a client. Other skills acquired include PHP, MySQL, Java, Python, Git, and game development with C#, Unity, and GoDot.
    - **Ethan:** Software Engineering major with experience in Java, C, HTML, CSS, JS, Git, MySQL, Spring Boot, Azure DevOps, CI/CD, and React. Has previously worked primarily from a backend and DevSecOps perspective and will be interested in learning more about frontend development beyond what was learned in 319.
    - **Nick:** Software Engineering major proficient in programming languages like Java, Python, C, JavaScript/React, and databases like SQL and MongoDB. Nick has worked on embedded software (Ada) at Collins Aerospace for over a year. Nick received a commendation on his web dev final project from my COMS319 professor and received 100% in the course. Nick won two awards in COMS309 for building a full-stack app as Best Manager and Best Coder, and received 100% in that course.
    - **Ponciano:** Software Engineering major experienced in JavaScript, C, Java, and Python. Ponciano has experience in frameworks and libraries such as React, NodeJS, and Flask. Ponciano has made various full-stack projects, such as a mobile app in COMS 309 and an application during my internship, where he and his team were nominated with Best Project for both applications.
    - **Evan:** Computer Engineering major with experience in working with hardware programs and circuits. Evan enjoys working with data and finding patterns within

it. Evan is proficient in working with Python and Java, with less experience in R, C, HTML, and MIPS.

- Strategies for encouraging and supporting contributions and ideas from all team members:
    - Give quick updates during SE 4910 on any tasks that need attention or any concerns that need to be brought up
    - Encourage an environment where members feel free to express their ideas and provide feedback
    - Meet virtually or in person to collaborate on more intensive tasks
    - Don't shut down ideas right away
- Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
    - Communicate with the team members individually or as a whole group (scenario dependent)
    - Talk to professors or group advisors if an issue cannot be resolved

Goal-Setting, Planning, and Execution

- Team goals for this semester:
    - Have a well-organized project structure with specific team roles and responsibilities
    - Meet all requirements and deadlines for the project
    - Begin the development of components
    - Become better developers and learn more about the industry.
    - Finished project design by the end of the semester
- Strategies for planning and assigning individual and team work:
    - Distribute assignments based on members' experience and personal interest
    - Each member should post and solve a git issue each week
    - Keep others updated in Discord and person on what you're working on and if you need help.
- Strategies for keeping on task:
    - Posting git issues
    - Attend weekly meetings
    - Check Discord messages and respond on time
    - Weekly team progress checks

Consequences for Not Adhering to the Team Contract

- How will you handle infractions of any of the obligations of this team contract?
    - Problems will be dealt with on a case-by-case basis, with increasing consequences. With each infraction, the team will lay out how the team can help the member and how that member can get back on track.
- What will your team do if the infractions continue?
    - Contact the team's advisor

****

a) I formulated the standards, roles, and procedures as stated in this contract.

b) I understand that I must abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Yusef Harb	**DATE** 5/4/25
2) Evan Sivets	**DATE** 12/8/25
3) Tristan Nono	**DATE** 5/4/25
4) Ponciano Ramirez	**DATE** 5/4/25
5) Nick McCullough	**DATE** 5/4/25
6) Ethan Messmer	**DATE** 12/8/25