4.2 Design Exploration

4.2.1 Design Decisions

To ensure a smooth and responsive UI/UX experience for users, we will implement a front-end enhancement that stems from several design decisions from our analysis of users' needs and requirements. Through this analysis, we narrowed down our design decisions to three primary areas. These changes aim to enhance user experience and guide the project to success.

1. Rehaul web sockets 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 UI 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 our live data displays that we have 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 10,000 grid nodes without lag. Our 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

To address our need to migrate UIs, we utilized the lotus blossom technique to explore other UI libraries that better suit the project and users' needs. This approach allowed us to view various UI libraries that align with our objectives of project performance, flexibility, and design consistency.

We identified five potential options:

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 full 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 our users and project requirements, we developed a weighted decision matrix 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)
Development Speed	0.25	5(1.25)	3(0.75)	2(0.5)	1(0.25)	5(1.25)
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)
Total Score	1.00	3.75	4.2	2.85	3.05	2.9

After evaluating our decision matrix, we selected ShadCN UI as GridAI's new UI library, which achieved a score of 4.2 on our 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

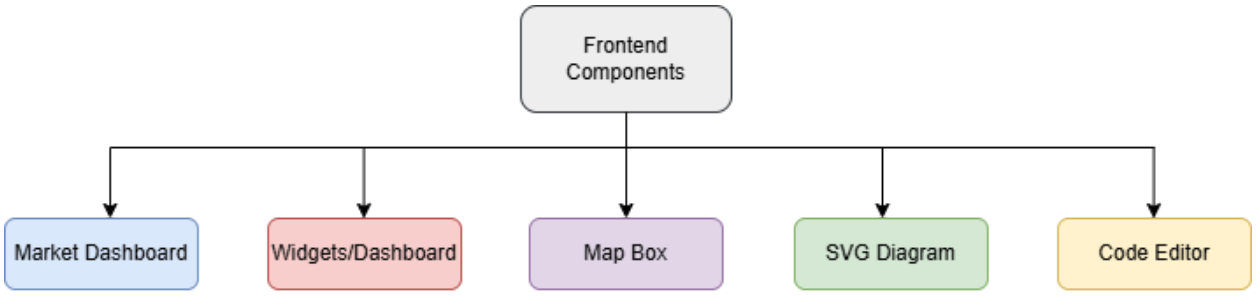
Long Term Advantages:

- 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 our 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 our immediate technical needs; it allows a foundation for more robust features to be built and maintained.

4.3 Proposed Design

4.3.1 Overview



Widgets/Dashboard Component

The Widgets component is one of the core services in our 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 allows 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
- Instant 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 multiple 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

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

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

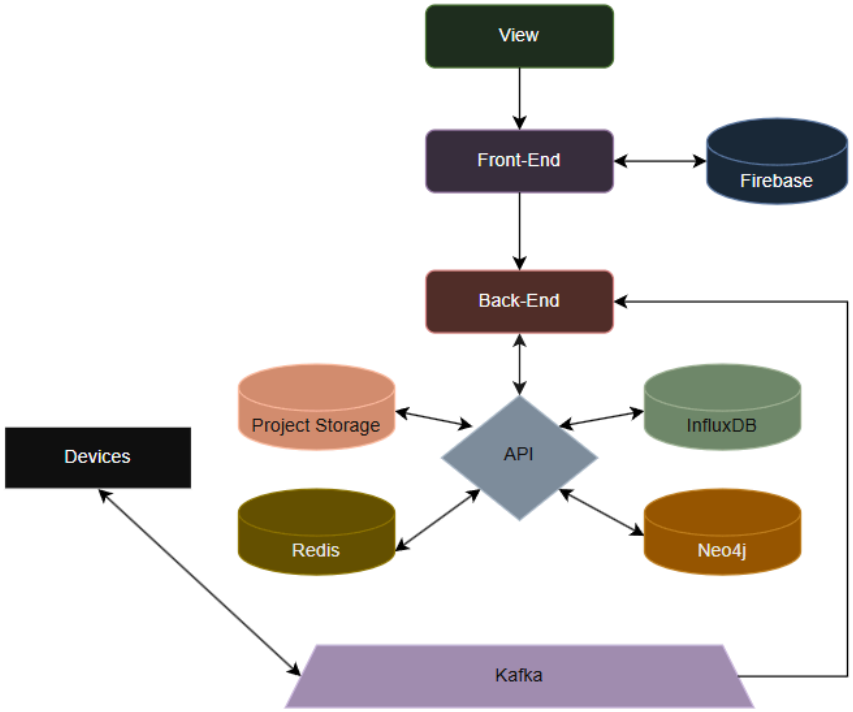
- Purpose: Efficient data exchange and system optimization between front-end and back-end
- Mechanics: Manages real-time data stream pipelines, sophisticated caching, oversees server-side rendering, and handles robust security authentication
- Capabilities: High-throughput data handling, performance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



4.3.3 Functionality

Our design is intended to operate in the real world by allowing users to have a visual representation of their energy, whether it is for research, solar panel systems, or energy companies with thousands of energy nodes. For example, a user can create their visuals of the energy through customizable widgets and can see how much energy is being consumed for their specific grid system.

GridAI has many features that support its users when it comes to 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 of monitoring their grid system.

Another feature available to users is the Single Line Diagram for displaying electric systems and provides a simplified version of the map box visualization, allowing for more flexibility for users to visualize their GridAI system. This can also help users get information about specific connections between multiple nodes. Other features cater to other types of users, such as developers and businesses. Developers can use this application for creating widgets within a code editor, and this gives them more freedom with customization.

GridAI's Code Editor allows users to write, edit, and collaborate with code, this is perfect for developers, researchers and even tech-savvy business owners. The code editor helps with simplifying 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 allows Distributed Energy Resource Aggregators (DERAs) to participate in the wholesale electricity market.

4.3.4 Areas of Concern and Development

Our design will meet users' needs to be easy to understand and interact with. That way, there is a minimal learning curve when working with our product. Overall aspects of our project we plan to incorporate simplicity in design for easy understanding and functionality. Our primary concern is for the product to be responsive and functionally sound first and foremost; design can come second after functionality.

We also look at the integration process of our design; with this, we want seamless communication between the user and the product. Our UI must be polished and tested to ensure that users have a good experience and understanding, even if they are new users or unfamiliar with the product. To fully understand what users like and dislike with our UI, we must perform user testing and gain valuable feedback to iterate upon. This way, we can grow and build upon positive or negative GridAI testing feedback. For the best optimization between all the components, more testing will be done with components, such as integration and stress testing.

4.4 Technology Considerations

Technologies we are using 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)

- NextJS
    - 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 technology in GridAI (Backend):

- NodeJS
    - 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, our 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 doesn't, as we inherited a codebase with limited documentation. One of our primary objectives has been transitioning the entire UI from Mantine to Shadcn UI—a change we plan to complete across all five components by the end of the semester.

Looking ahead, our future efforts center around building upon the progress made by the previous team. For the widgets component, we are currently in the process of integrating real live data, a major milestone in SE 492. We expect this to be a key focus moving into next semester, as it plays a central role in our application.

For the Map Box component, we aim to enhance timeline scrubbing and real-time data visualization by improving node and line rendering using Deck.GL and Dexie.js. Our goals for the market dashboard component include finalizing both the ISO and DSO dashboards and continuing where 492 left off.

In terms of collaboration, we are exploring CRDT-based socket libraries to enable real-time functionality within our Live Code editor. Lastly, for the SVG diagram component, we plan to migrate a significant portion of the data to Firebase to improve scalability and manageability.