Elevator Challenge Architecture Overview

The project simulates a building with multiple floors and elevators using React and TypeScript. It follows a Model-View architecture with these components:

Models Layer

1. **ElevatorSystem Class:** Central controller managing all elevators and floors. Handles calls, finds closest elevator, and orchestrates movements.

2. **Elevator Class:** Represents individual elevator with state (current floor, target floor, movement status, door status, queue). Calculates wait times and handles movement.

3. **Floor Class:** Represents building floor with state about calls, presence, and estimated wait times.

4. **AbstractFactory Interface:** Defines an interface for creating families of related objects (elevators, floors, elevator systems) without specifying their concrete classes.

5. **Factory Class:** Concrete implementation of the AbstractFactory interface. Creates standard elevators, floors, and the elevator system with specified configuration.

View Layer (React Components)

1. **App Component:** Sets up building parameters and initializes the system.

2. **Building Component:** Renders visual representation of building with floors and elevators.

3. **Floor Component:** Displays floor with call buttons and wait time indicators.

4. **Elevator Component:** Visualizes elevator movement using CSS transitions based on state.

Connection Layer

- **useElevatorSystem Hook:** Custom hook connecting React components to elevator system model, providing state updates and handling user interactions.

Main Algorithm Description

The elevator management algorithm includes:

Handling Elevator Calls

When a button is pressed on a floor:

1. System finds closest elevator based on wait time calculations

2. Floor state updates to reflect active call and estimated wait time

3. Selected elevator adds floor to its queue

4. Queue processing begins if elevator can move immediately

Finding Closest Elevator

For each elevator, the system:

1. Calculates total wait time including current movement, door operations, and queued stops
2. Selects elevator with minimum total wait time

Elevator Movement Process

1. Elevator updates state with movement details and wait time
2. Upon arrival, plays sound, updates state, removes floor from queue
3. Doors open for configured time period
4. After wait time, doors close and next floor in queue is processed

State Synchronization

- Periodic state sync ensures UI components reflect current state of models
- This separation allows models to operate independently of UI

The system includes audio feedback with a "ding" sound on arrival and handles edge cases like preventing multiple calls from same floor and managing elevator availability.

Implementation uses TypeScript for type safety with defined interfaces, follows object-oriented design principles including Abstract Factory pattern for flexibility, and maintains separation between business logic and presentation layers.