MICHIGAN STATE UNIVERSITY

Project Plan Presentation

Real-Time Seating Chart API, Builder, & Viewer

The Capstone Experience

Team Ludus

Jake Aquino
Yashika Desai
Noah Gaw
Navya Gupta
Bach Nguyen
Grayson Viener

Department of Computer Science and Engineering
Michigan State University

Spring 2026



From Students…
…to Professionals

Project Sponsor Overview

- Ludus is a software platform
- They power many arts organizations across the country
- A major pillar of their platform is online & box office ticketing
- Ben St. John is the CTO and our contact at Ludus
  - He oversees high level application development
  - He works with senior developers to ensure scalability, security, compliance, and stability.





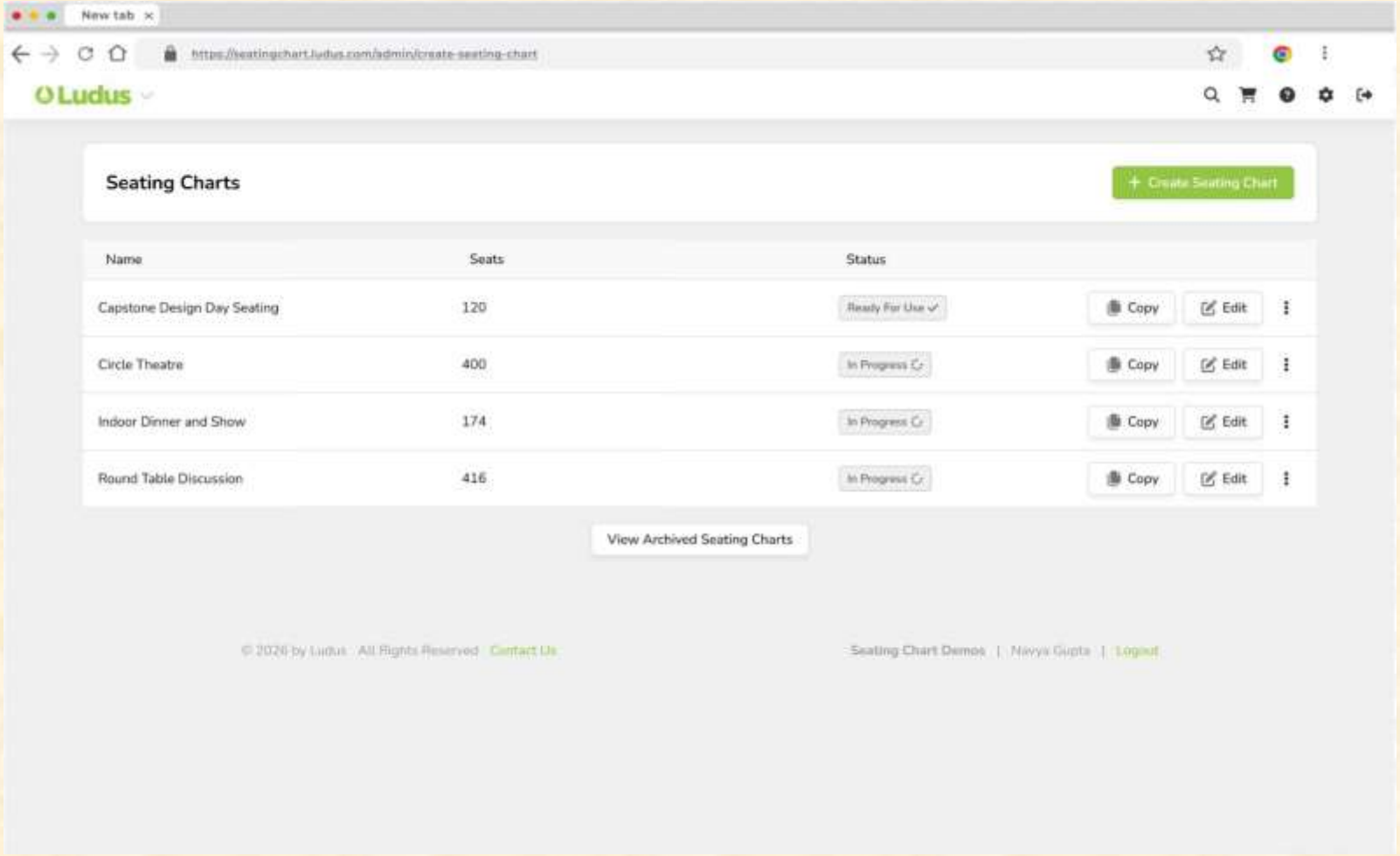
Project Functional Specifications

- Update and Modernize Seat Charting System
- By creating a Robust Charting Manager
- Deployed On a Web-Platform
- Venues across the country will use this for ticketing
- Deliverable that Ludus can sell to venue owners

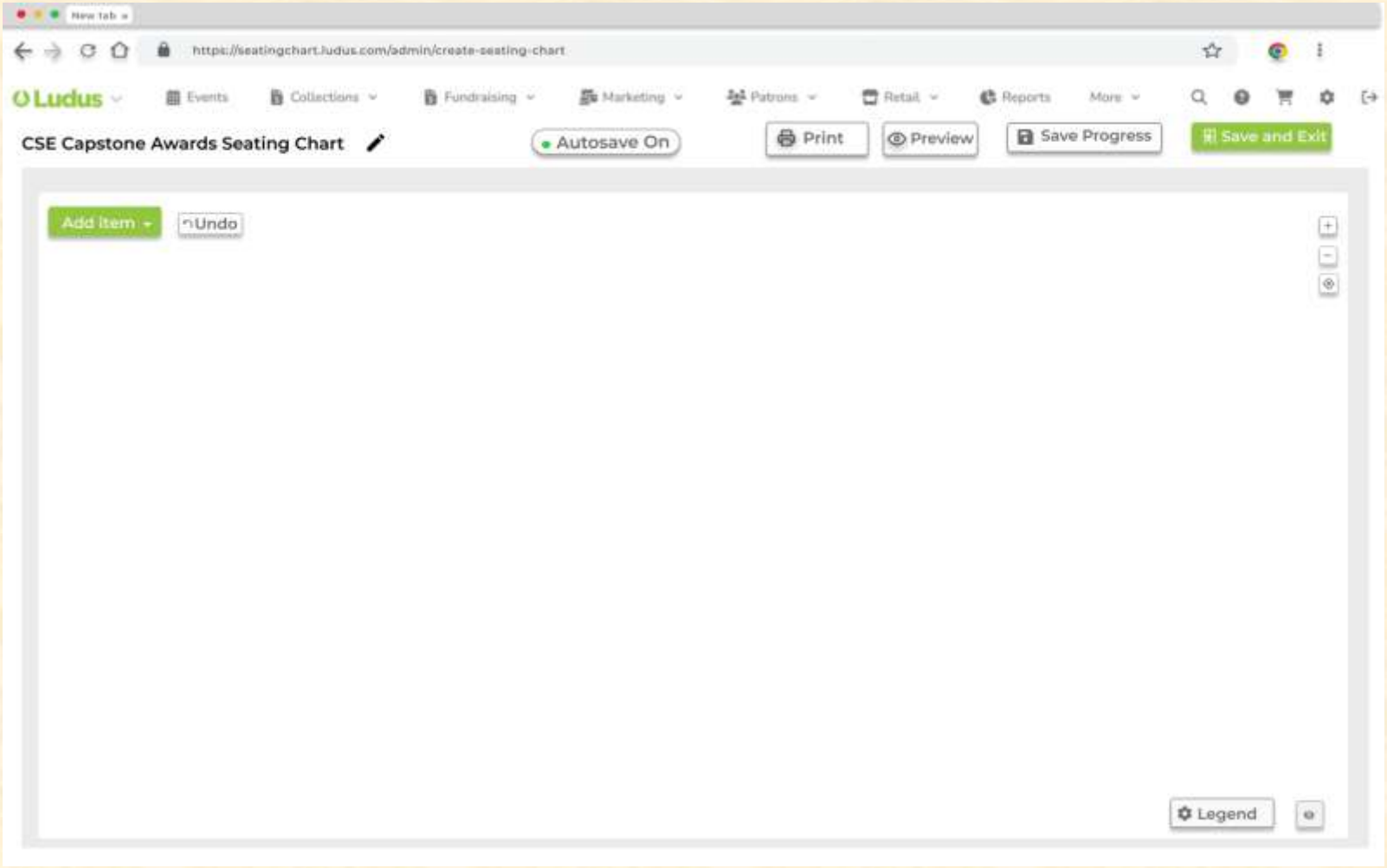
Project Design Specifications

- Implement Seating Chart Renderer and Viewer
- Robust & Adaptable Charting API
- Implement Compatibility with other Ticketing Platforms
- Develop an Embeddable Front-End Widget

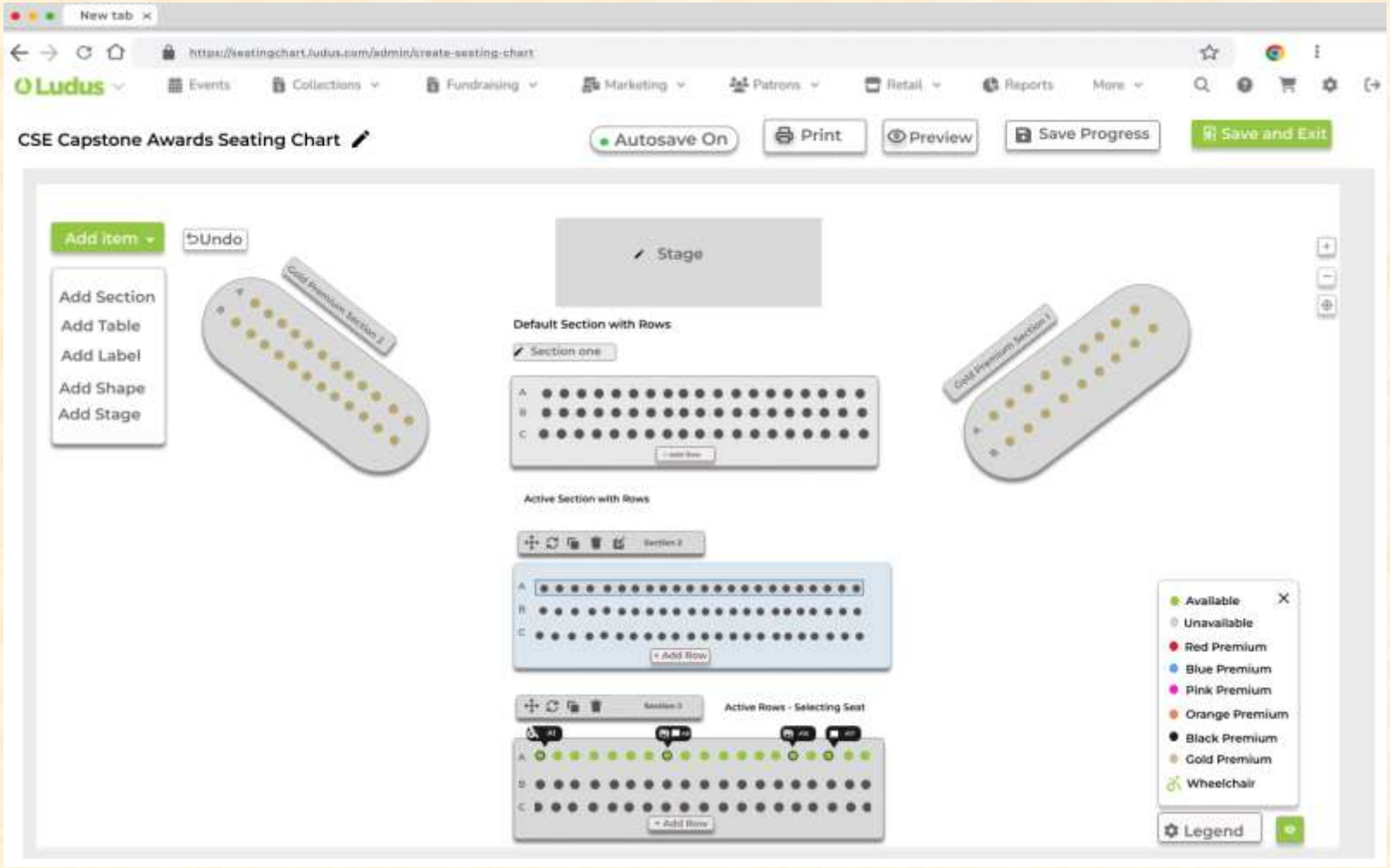
Screen Mockup: Admin Dashboard



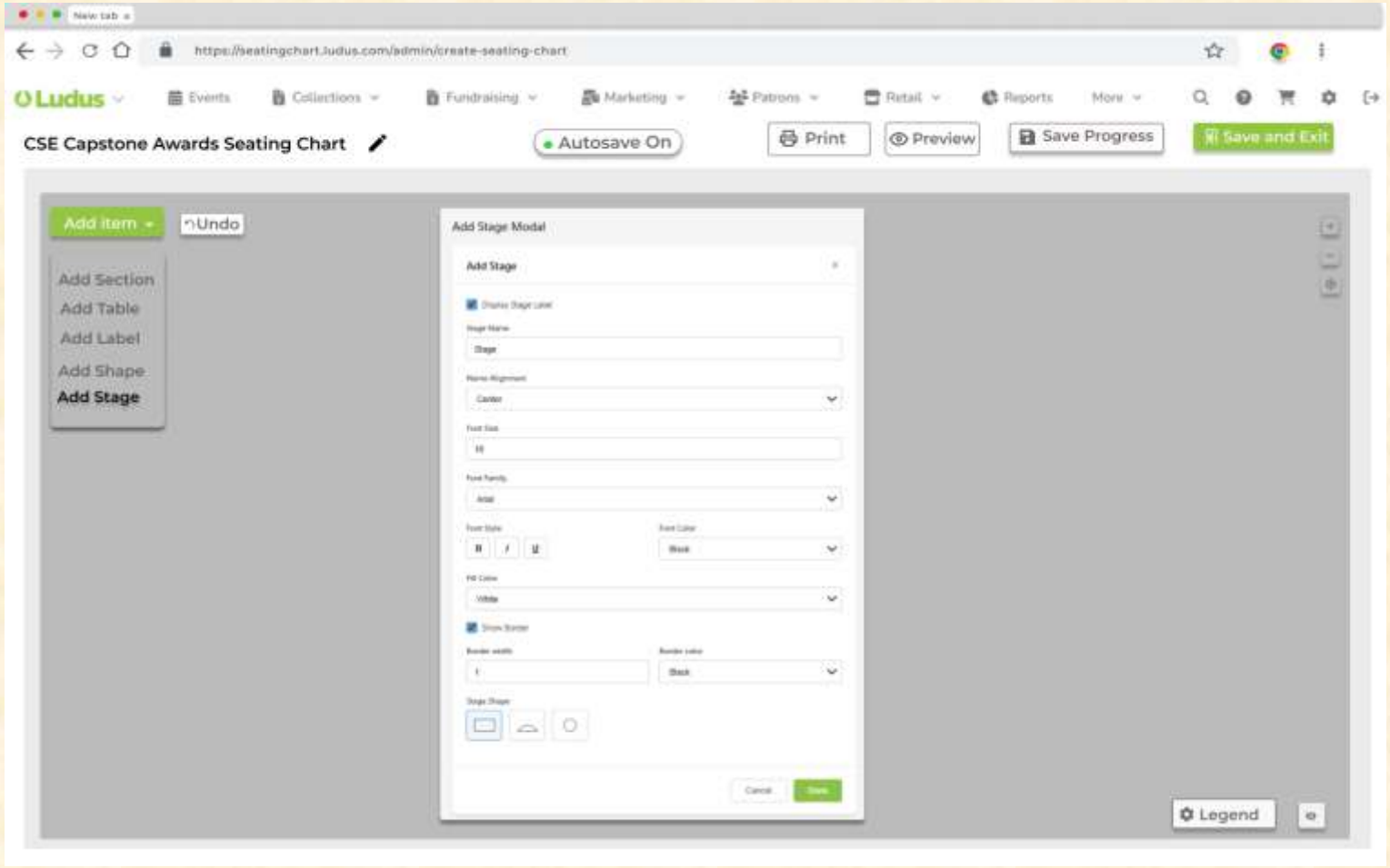
Screen Mockup:
Canvas Design Layout



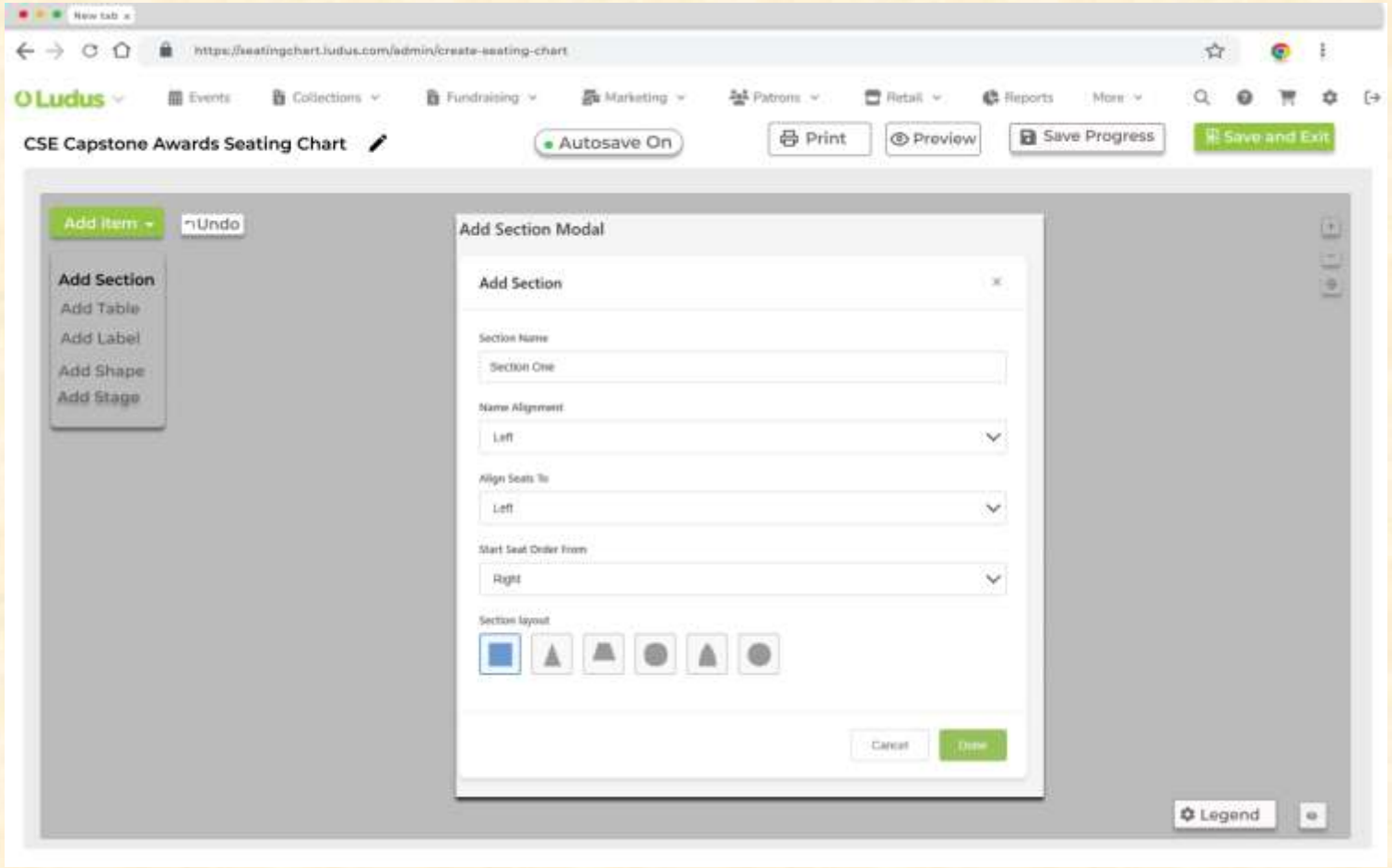
Screen Mockup:
Example Venue Seating Chart



Screen Mockup:
Add Stage



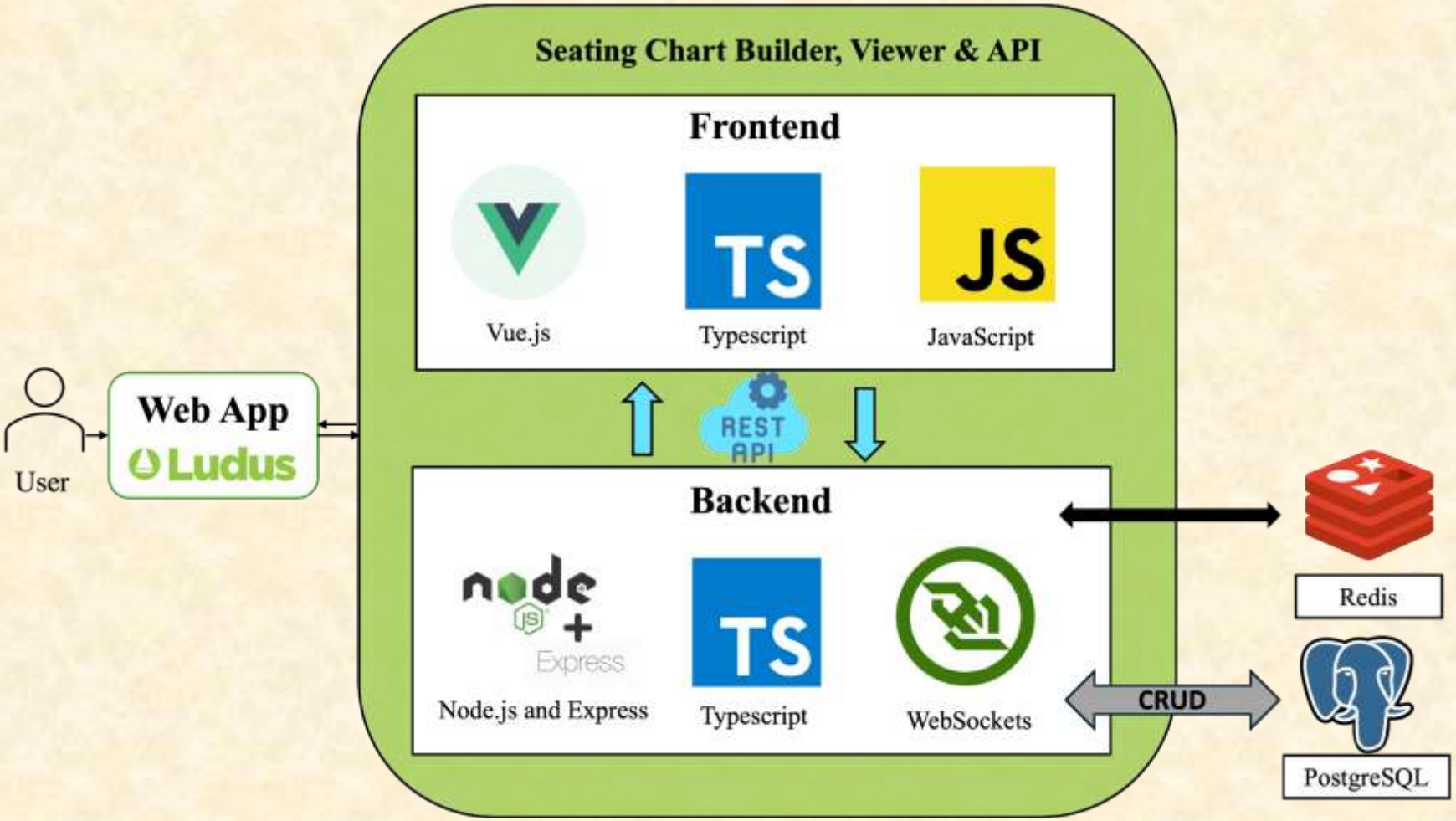
Screen Mockup: Add Section



Project Technical Specifications

- Frontend uses Vue.js and TypeScript/JavaScript
- Vue.js uses SVG to render components
- Backend built with Node.js, Express, and TypeScript
- JavaScript SDK to wrap API calls
- PostgreSQL database to store seating chart data
- WebSockets for real time seat interactions
- Using Redis for caching data

Project System Architecture



Project System Components

Software Platforms / Technologies

- Vue integrated with SVG
- Initialized Google Cloud CI/CD pipeline
- Using Docker to containerize and deploy
- Node.js API with end-points
- Frontend Ludus Webserver
- Backend SQL database
- Backend Redis cache

Project Risks

Risk 1: Temporary Holds and Locking

- We intend to use Redis in combination with our PostgreSQL server to process temporary holds and locking seats that are being purchased. We don't know if Redis can be used in a lock-and-key way to prevent race conditions between multiple servers when scaled. Having multiple servers would allow the memory load to be distributed amongst many servers and is a near pivotal requirement for having a global product..
- Stick to already proven ways to use Redis, such as optimizing one central server. We can also consider using event location based Redis distribution. I.e. if a venue in London is created, the Redis server used is the one closest to London, regardless of client location.

Risk 2: Render Time Optimization

- SVG and JSON both can see issues when used on a massive scale. SVG rendering on a single thread can slow down render time when used with thousands of objects, such as seats in a large stadium. JSON encounters a similar issue but for network traffic. We don't know if these assets can be optimized enough to not pose an issue at scale.
- We can use culling to reduce the amount of SVGs drawn on the canvas depending on the viewport, should they slow down the render time.

Risk 3: Embedding

- Ludus wants an embeddable widget that can be delivered to other organizations to easily add the seating chart into their own site for both the builder and viewer of the seating charts. We don't know if a seating chart widget can contain all elements present in the builder, as there are many key features in the builder that rely on having a large screen real estate.
- We will base our builder on the current widget that Ludus has, and if possible, include as many key elements as possible in the builder.

Risk 4: Canvas Modularity

- In order to have every element necessary in the builder, we need to ensure that the canvas is an event-driven separate object that can process all of the seats and non-interactable objects. We don't know if Vue supports the data structure we are trying to make to ensure that the canvas remains its own object. This is the most pivotal part of the project, as the entirety of the builder and viewer rely on the canvas being able to be a self-populating object. This part of the code will inevitably be very fragile, and we will need to test it.
- We will attempt to create the data structure how we envision it. So far we know that a Vue object can import other objects onto its "template", or html code. Given this, we can manipulate enough of the draw logic within the canvas itself, but this can quickly become an error-prone solution.

Questions?

