PROJECT ARCHITECTURE DOCUMENT
WEDDING SEATER PRO

OVERVIEW

1. Project Background and Description

This project was initiated to facilitate the bride and groom in preparing for their wedding by organizing and arranging seating. The target clients are wedding planners. The project provides an initial solution for arranging guests at tables based on categories such as bride's side, groom's side, family, friends, and professional contacts.

2. Project Scope

The scope of this project includes the development of a web application that allows the event host to enter guest lists, define the number of tables and specify seats at each table. The app will use an algorithm to automatically assign guests to tables based on predefined categories such as bride's side, groom's side, work, family or friends. The output will be a map of the venue with interactive tables showing the list of guests at each table.

The system allows users to make manual adjustments to the seating arrangement provided by the algorithm, display the changes made, and give the user the option to save or cancel the modifications. Additionally, an Excel file can be downloaded, showing the guests seated at each table.

Out of bounds are functionality such as consideration of guest attendance confirmations (assuming the list provided to the algorithm includes only those who are certain to attend), accuracy about table placement, and considerations for table allocation based on age categories or health conditions.

3. High-Level Requirements (Design)

The new system must include the following:

- A user-friendly web interface that requires no software downloads for access.
- Compatibility with various devices and browsers to ensure accessibility for all users.
- An ability to upload guest lists and table list.

- An algorithm that organizes guests into tables based on predefined categories and preferences.
- A graphical representation of the arrangement of the tables.

4. Who are the users?

Administrator/ event host: can enter the guest list, including details such as the number of guests, their side, and their closeness, as well as define the table details, including the number of guests per table, and oversee the entire seating arrangement process.

5. Project entities (Architecture)

Fronted – vite, vue.js

Backend, server – Node.js

Database – Mysql workbench, migration

6. Typical user flows

1) User Registration: The event host registers on the platform by providing necessary personal information such as name, email, and password.
2) User Login: After registration, the event host can log in using their credentials to access their personalized dashboard and manage their events.
3) Event Creation: The host enters specific details like the guest list, number of tables, and seating capacity per table.
4) Seating Arrangement and Adjustments: Using the uploaded or directly entered guest data, the system generates initial seating arrangements based on predefined criteria such as relationship to the bride or groom, work, family, or friends.
5) Add Tables and Chairs: If the number of guests exceeds the seating capacity, the system will prompt the host to add more tables and chairs.
6) Drag and Drop Adjustments: The host can manually adjust the seating arrangement using a drag-and-drop interface, and to ensure that the number of guests per table does not exceed the seating capacity.
7) Save Changes and Display Corrective Actions: Once changes are made, the system allows the host to save the adjustments and displays any corrective actions taken by the algorithm.
8) Download Excel File: After finalizing the seating arrangement, the host can download an Excel file that lists all guests and their assigned tables.

7. Epics and user stories and test plans scenarios

Epic 1: User Registration and Authentication

User Story: Register

Description:
As a user hosting an event, I want to register for the event seating management system to utilize the seating arrangement algorithm.

Subtasks:

- Display a registration form with fields for name, email address, and password, along with password confirmation.
- Validate that all fields are filled correctly: ensure that the email address is not already associated with another user, the password and password confirmation match, the password meets complexity requirements, and the email address is in valid format.
- Provide a link that redirects logged-in users to the login screen.
- Display a login button for users who are not yet registered.
- After successful registration, display a success message and redirect the user to the login screen.

Test Plan:

- Verify that the registration form displays all required fields.
- Check validation rules for each field (email uniqueness, password complexity and matching, email format).
- Confirm redirection functionality and success message upon successful registration.

User Story: Login

Description:
As a user hosting an event, I want to log in to the event seating management system to utilize the seating arrangement algorithm.

Subtasks:

- Display a login page prompting the user to enter their email address and password.
- Validate the email address and password format.
- After successful authentication, redirect the user to the system dashboard.
- Display an error message for incorrect login credentials.

Test Plan:

- Ensure the login page displays the correct fields.

- Confirm email and password format validations are enforced.
- Check redirection after successful login and error messaging for incorrect credentials.

User Story: User Logout

Description:
As a logged-in user, I want to log out from the system to ensure my session is securely terminated and my access to restricted features is revoked.

Subtasks:

- Implement a "Logout" button on the user interface.
- Invalidate the token on the server side.
- Redirect the user to the login screen after logging out.
- Ensure that logged-out users cannot access restricted routes without logging back in.

Test Plan:

- Verify that the "Logout" button appears when the user is logged in.
- Ensure that clicking "Logout" removes the token from the client.
- Confirm that the user is redirected to the login page after logout.
- Test that restricted routes are inaccessible without logging back in.

User Story: User Authentication and Token Management

Description:
As a user, I want to securely log in to the system and receive a token (JWT) to access restricted features and data associated with my account.

Subtasks:

- Implement user authentication using email and password.
- Hash passwords using bcrypt and store them in the database.
- Upon successful login, generate a JWT token that includes the user ID and role.
- Store the token securely on the client side.
- Ensure the token is required for accessing restricted routes on the backend.
- Validate the token on every request to ensure the user is authenticated.
- Implement token expiration logic for enhanced security.
- Add token refresh logic, if needed, to extend session without re-login.

Test Plan:

- Verify that users can log in with valid credentials and receive a token.
- Ensure the token is stored securely and accessible for future requests.
- Confirm that restricted routes are inaccessible without a valid token.
- Test that the token expires as expected, and users are logged out when it expires.
- Verify that invalid or expired tokens are rejected appropriately by the backend.

Epic 2: Guest and Table List Management

User Story: Display Event Summary Card

Description:
As a user hosting an event, I want to view a summary card displaying the number of guests for the bride and groom's side, the number of invitations, the number of seats, and the number of tables, so I can have a quick overview of the event's seating arrangements.

Subtasks:

- Design a summary card layout that displays the following information:
  - Number of guests for the bride's side.
  - Number of guests for the groom's side.
  - Total number of invitations sent.
  - Total number of tables.
  - Total number of chairs.
- Fetch data from the database to dynamically populate the card with real-time information.
- Ensure the card updates automatically if any of the data (guests, invitations, seats, tables) changes.
- Implement logic to differentiate between the bride and groom's side when counting guests.

Test Plan:

- Verify that the card displays all required information (guests, invitations, seats, tables) correctly.
- Test the card's responsiveness to changes in guest count, invitations, seat availability, and table setup.
- Ensure that the bride's and groom's guest counts are calculated correctly based on the side data.

User Story: Add Guest

Description:
As an event organizer, I want to create a guest by entering their name, quantity (number of attendees), side (bride/groom), and closeness level. I also want the ability to delete or edit guest information, and have the guest data reflected in the summary card.

Subtasks:

- Display a form with fields for guest name, quantity, side (bride/groom), and closeness.
- Implement functionality to add a guest and update the guest count in the card.
- Enable guest editing and ensure any changes are reflected in the card (updating the number of guests accordingly the side).
- Implement a delete function for removing a guest and update the guest count accordingly.

Test Plan:

- Verify that the form displays all required fields.
- Test the addition of a new guest and ensure the card updates the guest count by side.
- Ensure that editing a guest updates the card correctly.
- Ensure that deleting a guest adjusts the card data properly.

User Story: Add Table

Description:
As an event organizer, I want to create a table by entering its number and the number of seats and have the table data reflected in the summary card. I also want to be able to edit the table information.

Subtasks:

- Display a form to enter table number and seat count.
- Add functionality to create a table and update the seat and table count in the card.
- Enable table editing, ensuring any changes update the card.
- Enable table deleting, ensuring any changes update the card.
- Validate that any changes in the seat count affect the card appropriately.

Test Plan:

- Verify the form displays fields for table number and seat count.
- Test the addition of a new table and confirm the card updates with the correct number of tables and seats.

- Confirm that editing table details updates the card correctly.

Epic 3: Automated Seating Arrangement

User Story: Generate Seating Arrangement

Description:
As an event organizer, I want to generate seating arrangements for my guests, ensuring that they are seated at tables according to their side (bride/groom) and closeness, with groups filling tables efficiently. If the number of guests exceeds available seats, I want to be redirected to an error page.

Subtasks:

- Retrieve guest data from the database, grouped by side and closeness.
- Sort guest groups by size, with the largest groups prioritized for seating.
- Retrieve tables from the database, sorted by seat capacity.
- Assign guest groups to tables based on the number of available seats per table.
- Update the database to store which guest is assigned to which table.
- Ensure that if the total number of guests exceeds the available seats, return an error message.
- Handle edge cases where a guest group is too large for a table, ensuring smaller groups are assigned to remaining seats.
- Log the seating plan and verify correct assignment.
- Return the final seating plan in the response to the client.

Test Plan:

- Verify that the guest data is retrieved and grouped correctly.
- Test the sorting of guest groups by size.
- Ensure that tables are retrieved and sorted by seat capacity.
- Confirm that the seating algorithm assigns guests to tables efficiently.
- Test edge cases where the guest count exceeds seat capacity, ensuring an error message is returned.
- Verify that the correct guest-to-table assignments are stored in the database.
- Ensure that the seating plan is returned correctly in the response.

User Story: Generate Tables

Description:
As an event organizer, I want to see the tables displayed visually with chairs colored based on the

number of guests seated at each table, so I can understand how the seating arrangement was generated by the algorithm.

Subtasks:

- Implement visual representation of the tables, including chairs arranged around each table.
- Color the chairs to reflect the number of guests seated at the table.
- Display the total number of guests vs. the table's capacity.
- Ensure that the table and guest data are fetched and displayed based on the seating arrangement generated by the algorithm.

Test Plan:

- Verify that tables are displayed with the correct number of chairs.
- Ensure that the chair coloring accurately reflects the guest seating count.

User Story: Export Seating Arrangement to Excel

Description:
As an event organizer, I want to export the seating arrangement to an Excel file, showing which guests are seated at which tables.

Subtasks:

- Add an "Download Excel" button to the seating arrangement page.
- Include the guest's name and table number in the Excel file.

Test Plan:

- Verify the Excel export functionality works.
- Ensure the Excel file contains the correct guest names and table assignments.
- Test the download of the Excel file.

User Story: Manual Seating Adjustments

Description:
As an event organizer, I want to manually adjust the seating arrangement by dragging and dropping guests between tables, with the ability to save or discard changes, so I can have more flexibility in organizing the event.

Subtasks:

- Implement drag-and-drop functionality to allow moving guests between tables.
- Ensure the seating plan updates in real-time as guests are moved between tables.

- Validate that the total guest count for each table updates when guests are added or removed.
- Prevent invalid moves (exceeding table capacity or attempting to move guests into unavailable chairs).

Test Plan:

- Verify that guests can be moved between tables using drag-and-drop.
- Ensure that the guest count and chair status update correctly after each move.
- Test edge cases where a move exceeds the table's capacity and confirm the system prevents the move.

User Story: Save Changes to Table Arrangements

Description:
As an event organizer, I want to have a save button on the seating arrangement page so I can save changes made to the table arrangement.

Subtasks:

- Add a "Save Changes" button to the seating arrangement page.
- Implement basic functionality for the save button to trigger the saving process.
- Ensure the button is clearly visible and accessible to the user.

Test Plan:

- Test that the "Save Changes" button appears on the page.
- Ensure that clicking the "Save" button triggers the appropriate saving functionality.
- Verify that the page or user state is updated correctly after saving.

User Story: Display Corrective Actions

Description:
As an event organizer, I want to be informed about any manual changes I made to the table arrangement, so I can decide whether to save or cancel the changes.

Subtasks:

- Implement logic to detect if manual changes were made (guest moves between tables).
- Display a message indicating no changes if applicable.
- If changes were made, display a summary of which guests were moved and between which tables.
- Provide options to either save or cancel changes.
- Ensure the system reverts to the previous state if the user cancels the changes.

Test Plan:

- Test the detection logic for manual changes (e.g., check for drag-and-drop operations).
- Verify that a "no changes made" message is displayed when no manual changes occurred.
- Ensure that a correct summary of guest movements is displayed when changes are made.
- Test that the "Confirm" and "Discard" buttons function as expected, keeping or discarding changes as appropriate.

8. External tools, libraries, and third-party projects section

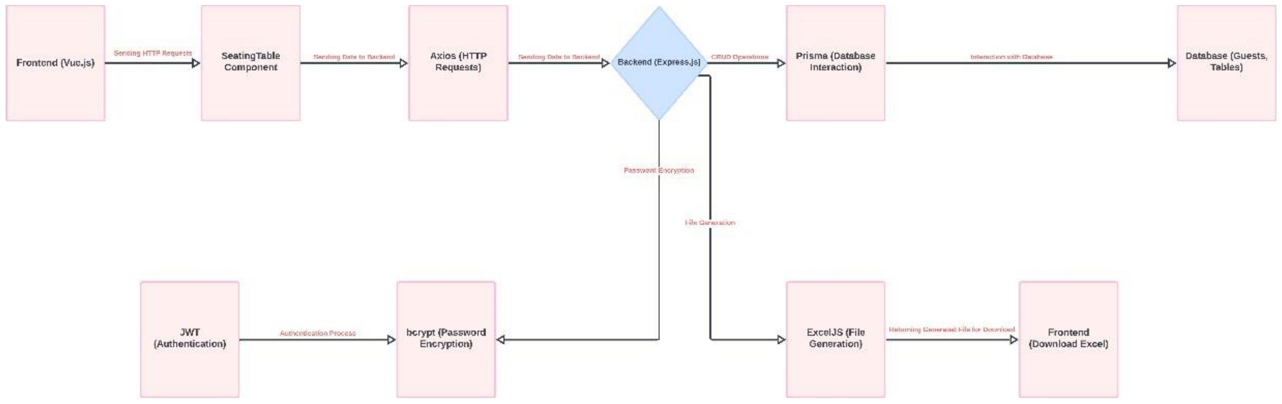
Frontend (User Interface):

- **Vue.js** :The core framework for building user interfaces.
- **VueRouter** :Manages navigation between pages in your Vue.js application.
- **Pinia** :A lightweight state management library for managing state in Vue.js.
- **Axios** :A promise-based HTTP client for making API requests to the backend.
- **VueDraggable**: A Vue component for creating drag-and-drop interfaces.
- **@vuelidate/core:** A simple, yet powerful validation library for Vue.js.
- @**vuelidate/validators**: A set of validators to use with Vuelidate for form validation.
- **ExcelJs:** Allows to create and manipulate Excel files within the app.
- **FileSaver.js:** Saves files on the client-side, enabling file downloads from the browser.

Server-side Dependencies

- **Express.js:** A fast, unopinionated web framework for building APIs and managing the server.
- **Prisma (@prisma/client):** Used for database communication and to interact with your database in a type-safe way.
- **JWT (jsonwebtoken):** Manages JSON Web Tokens for securing routes with token-based authentication.
- **bcrypt:** Handles password encryption and hashing for secure authentication.
- **ExcelJS:** Allows you to create and manipulate Excel files within your app.
- **body-parser:** Parses incoming request bodies in a middleware, making it easier to handle data.
- **CORS:** Enables Cross-Origin Resource Sharing, allowing your API to be accessed from different domains.

9. Application Flow: Frontend to Backend with Database and Excel File Generation


Frontend (Vue.js)
Sending HTTP Requests
SeatingTable Component
Sending Data to Backend
Axios (HTTP Requests)
Sending Data to Backend
Backend (Express.js)
CRUD Operations
Prisma (Database Interaction)
Interaction with Database
Database (Guests, Tables)
Password Encryption
File Generation
JWT (Authentication)
Authentication Process
bcrypt (Password Encryption)
ExcelJS (File Generation)
Returning Generated File for Download
Frontend (Download Excel)