AllEvently Report

Benjamin Bruyns, Peggy Lewis, Spenser Morey

Introduction

AllEvently is a website application that allows users to create and attend events. This application also allows people involved in the same event to chat online. AllEvently invitations can be sent via the app or by printing out an invitation poster. These functions enable the hosts to manage their events better and allow guests to respond more efficiently to invitations and interact with the other guests.

AllEvently is a versatile website application designed to simplify event management and enhance attendee engagement by addressing common challenges in event organization. It provides tools for creating events, inviting guests, and facilitating interaction among attendees, all in one platform. Integrating features like a live chat for real-time communication and multiple invitation options—digital through the app or printed posters—AllEvently solves the problem of fragmented event planning. Hosts can efficiently manage guest responses and logistics. At the same time, attendees enjoy a seamless way to RSVP, stay updated, and connect with others involved in the event, creating a more cohesive and interactive experience.

Technologies

GitHub

GitHub is one of the best tools for maintaining git repositories and tracking version control. It is an indispensable resource when working on teams such as AllEvently. It helped our team easily track updates, work on code in real-time, and keep a detailed project development history. The ability to create pull requests and new branches and maintain continuous integration through Vercel were extremely useful for improving our development productivity and easing our process.

As developers, we used GitHub on its site and through the built-in tools in our dev environments. WebStorm’s Git GUI made it easy to commit and push changes, view the commit history, and resolve merge conflicts. GitHub also has strong support for Git commands, making it easy for team members to maintain repositories from the command line. Additionally, WebStorm’s native Git button made it even easier to utilize features like the interactive Git Log, which offers an extensive graphical breakdown of the commit history. This tool allowed us to visualize the latest commits, commit

messages, author names, timestamps, and branch structure simultaneously to track commit relationships and merges easily. This log was a valuable resource for managing the repository.

Our team members were mainly used to using GitHub, so we quickly integrated it into our workflow. That familiarity allowed us to focus on using advanced capabilities to improve our repository. For instance, we crafted an easy-to-read ReadMe using markdown, organized files in the Documents folder with an explicit Table of Contents, and provided interactive elements like copyable code snippets to make the project experience easier for visitors. These improvements benefited our development process and allowed future viewers to use and see our project.

Vercel

Vercel is a secure hosting platform for hosting websites and databases. We chose Vercel because it promised to be very easy to upload our git repository and deploy the website continuously. Further, Vercel's capabilities to host a relational database fit our project's requirements for storing website data. The website and the database hosted on one platform helped provide a more seamless development experience. It also helps eliminate any integration challenges.

Before this project, we had never used Vercel, but the intuitive user interface made learning easy. Although there were a few learning curves to work through (from debugging deployment issues) and the inability to work together collaboratively when using the free version, Vercel proved helpful for our project. The continuous deployment meant that every update pushed to the GitHub repository was automatically pushed to deployment on the live website. This saved us time from having to perform the deployment process manually. We also used Vercel's environment variable management system to store sensitive data (Database connection strings and a Google API key) to help keep the project secure.

Using Vercel greatly impacted how we deployed and ran our project and tied right into our GitHub version control. It provided the ability to observe changes in GitHub that were immediately reflected on the site. Looking back on our tools and platforms, we can see how Vercel was essential in enabling AllEvently to be developed and launched.

PostgreSQL

PostgreSQL is a framework for databases. We chose it because we needed a relational database, and Vercel offered hosting for Postgres Database.

Before this project, we had used base SQL, so learning Postgres was fairly straightforward. There were some issues, but once they were discovered, they were easily rectified.

We had thought of NoSQL as the database framework but quickly decided we could not use NoSQL because we needed a relational database. We have no regrets about this decision, as Postgres has worked fine.

Vue.js

Using Vue.js on our AllEvently project was both a challenging and rewarding experience, offering valuable lessons and tangible benefits. Vue.js is a progressive JavaScript framework designed for building user interfaces, specifically for creating dynamic and responsive HTML. We chose Vue.js because of its reputation as an easy-to-use framework, which made it seem like an ideal choice for our project. None of us were fully competent in Vue.js, and the vast number of features and concepts initially felt overwhelming. However, by focusing on the most critical aspects of Vue.js, we learned what we needed to contribute to the project throughout the semester.

Despite the learning curve, Vue.js provided a modular structure that enabled us to create a component-driven frontend for AllEvently. Separating the user interface into reusable components was one of its biggest strengths. For example, we developed components like EventCard.vue, PublicEventCard.vue, Sidebar.vue, and TopPanel.vue, which could be reused across multiple parts of the application. This helped to reduce development time and minimized code repetition, making it easier to maintain and debug. Vue.js's scoped CSS also allowed us to encapsulate styles within individual components, ensuring that styles didn't unintentionally affect other application areas.

We had initially considered React.js or Next.js for our client-side framework, but Vue was selected because of its seamless integration with modern tools, such as Vite, which we used as our build tool. Vite's fast hot-module replacement feature enabled us to see real-time changes, significantly speeding up the development and debugging processes.

While Vue.js proved to be effective, it wasn't without its challenges. Managing state across components became increasingly complex as the project grew, particularly for features like user authentication and event management. Although it was difficult at times, we could make Vue.js work for our project, and the final result was a responsive, flexible, and maintainable website.

Node.js

Node.js is a framework for creating the backend functionality of a website. We chose it because it is an easy framework to use and implement.

Only one of us was initially competent in Node.js. However, Node.js was fairly easy to use despite this. It was also easy for the rest of us to learn because they were in a class teaching how to use it.

Design

Database Tables

People

This table stores information about individuals who have signed into the website and those invited to events. The rows contain an individual's first name, last name, and email address. The primary key is the email address.

Sessions

This table keeps track of users that are logged in. The rows contain a user's email and session ID. The session ID is the primary key, and the email is a foreign key referencing the People table.

Accounts

This table keeps track of users who have created an account on the website. The rows contain a user's email and password. The primary key is the email, which is also a foreign key that references the People table.

Reset_Credentials

This table keeps track of credentials that can be used to change an account password. It contains the account email and a temporary password. The primary key is the email, which is also a foreign key that references the Accounts table.

Events

This table keeps track of all events that have been created. The primary key of this table is the Event ID, and the host email is a foreign key that references the People table. For the contents of the Events table, refer to Figure 1.

Events	
PK	**EventID: int NOT NULL**
FK1	Event_Host: text NOT NULL
	Host_Name: text NOT NULL
	Event_Name: text default 'My Event'
	Event_Location: text NOT NULL
	Event_Start_Date: timestamp default currentdate()+1
	Event_End_Date: timestamp
	Event_Time_Zone: text default 'UTC'
	Invitation_Layout: text default 'Center'
	Background_Image: int default 0
	Font_Background_Color: text default '#000000'
	Font_Color: text default '#FFFFFF'
	Font: text default 'italic bold 20px arial serif'
	Is_Public: boolean default FALSE
	Reoccurs: boolean default FALSE
	Reoccur_Freq: int
	End_Reoccur: timestamp
	Request_Child_Num: boolean default FALSE
	Limit_Additional_Guests: boolean default FALSE
	Max_Additional_Guests: int default 0
	Notify_Host: boolean default TRUE

Figure 1. This figure is an ERD table representing the database Events table.

Guests

This table keeps track of people invited or registered for an event. The rows contain an event ID, a guest's email address, whether the guest has RSVPed, whether they were sent an invite, whether they accepted, a child count, and a guest count. The event ID and email address are both primary keys. The event ID is a foreign key referencing the Events table, and the email is a foreign key referencing the People table.

Chat_Messages

This table keeps track of messages sent by users in each event. The rows contain a message ID, an event ID, a sender's email address, message content, sent date, and whether it is an acceptance, rejection, or neither. The message ID is the primary key. The event ID is a foreign key referencing the Events table, and the email is a foreign key referencing the People table.

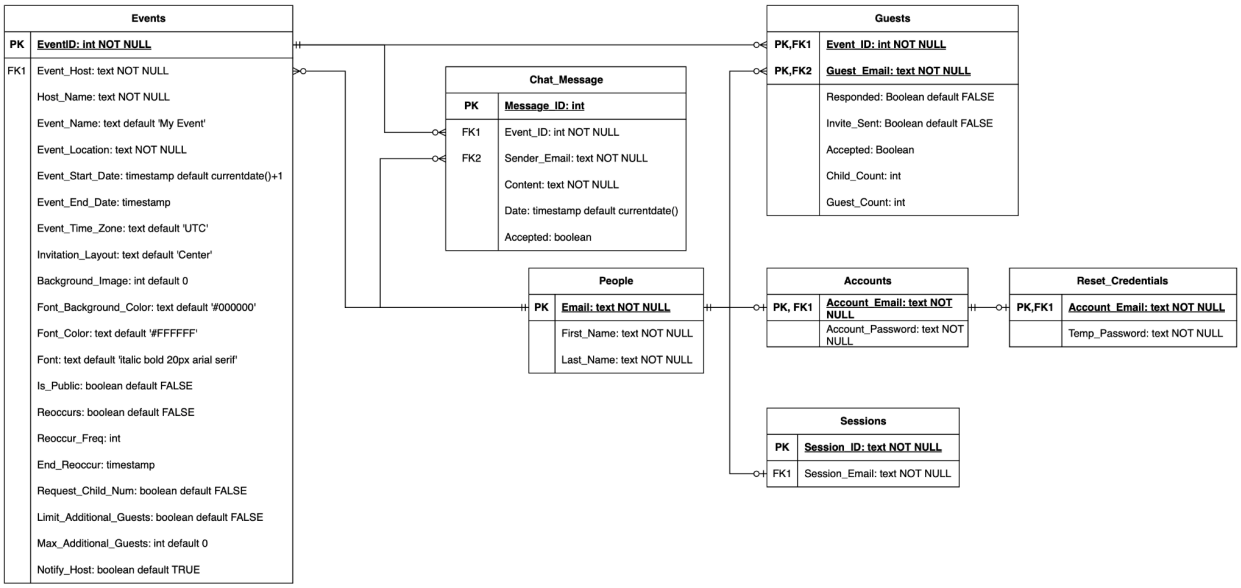


Figure 2. This figure shows the ERD diagram of our database

Script Files

Vue Files

The script part of the Vue.js code defines how the website functions and behaves, impacting how people interact with the page and the data presented. Below is a summary of the main components of the AllEvently project:

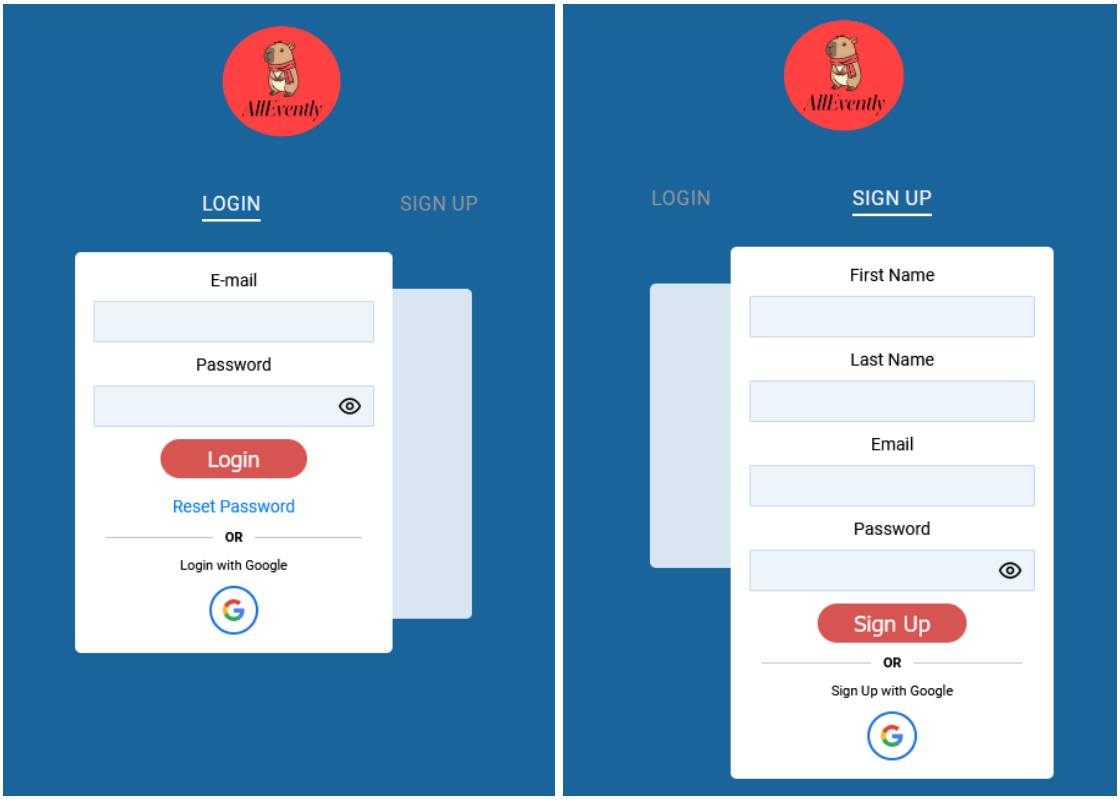


Figure 3. Login page

Login and Sign Up Page

The Login.vue script builds a TypeScript-based Vue component for logging in and signing up. It includes the necessary features such as user authentication and input validation. The scoped CSS powers the form by switching from the Login dialog to the Sign Up dialog, enabling both functionalities on the same page. It imports necessary dependencies, such as `ref` from Vue to manage reactive state, `useRouter` for navigation, and other assets, such as our logo and icons for visual aspects.

The script defines multiple `ref` variables to accept user input like `firstName`, `lastName`, `email`, `password`, and state management variables like `showPassword`, `currentIcon`, and `userId`. They store the form's data and the UI's dynamic behavior. Some of the key features include:

1. Login Functionality: The `loginUser` function sends a POST request to the backend server to authenticate the user's credentials. On successful login, it stores the `userId` in `localStorage` and routes the user to the Events.vue page. It also handles error responses from the server gracefully.
2. Email Validation: The `validateEmail` function uses a regular expression to ensure the email format is valid, updating the `isEmailValid` state accordingly.

3. Signup Functionality: The `handleSignup` function validates all input fields for completeness and format before sending a POST request to create a new user account.
4. Password Visibility Toggle: the `togglePasswordVisibility` function dynamically changes the password field's visibility and updates the corresponding icon.
5. Form Switching: The `toggleForm` function manages the UI's state, allowing users to seamlessly switch between login and signup forms.
6. Password Reset Navigation: The `handleResetPasswordLink` function redirects users to the PasswordReset.vue page.

Login and Sign Up dialogs have been integrated with the GoogleSSO.vue component, providing users with a quick and convenient Google Single Sign-On option.

The Sign Up dialog allows users to create an account by entering their first name, last name, email, and password. However, account creation will only succeed if the email is not already in use or the correct format.

The Login dialog allows users to access their accounts by submitting their email and password. If successful, the user is redirected. Additionally, a Reset Password hyperlink is available, guiding users to the ResetPassword.vue page for assistance with recovering their password.

Password Reset Page

The PasswordReset.vue script defines a TypeScript-based Vue component for handling password reset requests. This page provides user input validation and a simple process to request a new password. This component imports familiar dependencies from the Login.vue components such as `ref`, `router`, and logo. Some of the ref variables for managing user input include `email`, `isEmailValid`, `showSuccessDialog`, and `showError`. These variables capture the user's email, track if the entered email is in the valid format, and then control the confirmation display for a successful request or show error messages for invalid or missing input. Some of the key features include:

1. Email Validation: Using regular expressions, the `validateEmail` function validates the email from the specified pattern. The result updates the `isEmailValid` variable and gives instant feedback on whether an input is valid.
2. Handle Password Reset Requests:

    a. The `handleResetPassword` function validates the email input and handles the process of sending a password reset request.

    b. If the email input field is empty or invalid, it sets `showError` to true to display an error message.

    c. If it is a valid email, it hides the error message and displays a success dialog, letting the user know the request was received. The function provides a place for injecting backend API logic to issue the password reset request.

3. Success Navigation: The `handleOkButtonClick` function activates when the user clicks the “OK” button on the success dialog. It hides the dialog and navigates the user to the UpdatePassword.vue page using the router.

This component makes the password reset quick and easy and provides clear messages throughout the process.

Update Password Page

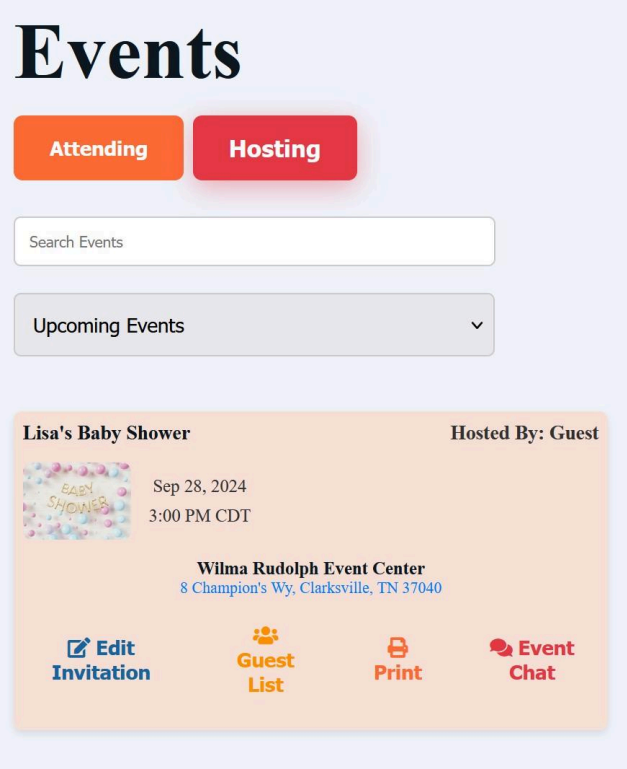
The UpdatePassword.vue script builds a TypeScript-based Vue component that is used to update an account’s password. The page is designed to guide users through verifying their emailed reset code and creating a new password. This page provides input validation and an “Update Password” button to submit the change. Like other pages, the component imports `ref`, the project logo, and the Vue `router` for navigation. The `ref` variables include `emailCode`, `newPassword`, `confirmPassword`, `codeError`, `passwordError`, and `confirmPasswordError` and `showSucessDialog`. The variables help store the entered verification code, the new desired password, confirmation of the new password, track any validation errors, and control the visibility of the success dialog. The key features include:

1. Form Validation: The `validateForm` function checks all the user input to ensure the emailed code isn’t empty, the new password is entered, and the new password and confirmation password fields match.

2. Password Update Process: The handlePasswordUpdate function validates the form using `validateForm`. If all the inputs are valid, the function makes an API call to update the password and displays a success dialog box.

3. Success Navigation: The `handleOkButtonClick` function is triggered when the user confirms the password update in the success dialog. It redirects them to the Login.vue page, providing a smooth transition after completing the process.

If the reset code is incorrect, the password will not be updated.



Events Page

The Events.vue script defines the core functionality of the events page for the AllEvently website. This page dynamically fetches and displays the users' hosted events and the events they've been invited to, which could be public or private. The page provides filtering between Attending and Hosting events and navigation options. The script integrates TypeScript and Vue.js features, leveraging `ref`, `computed`, and lifecycle hooks to manage state and data flow.

The component imports key dependencies, including reusable components like `TopPanel`, `Sidebar`, `SearchBar`, and `EventCard`, along with the Event type for strong typing. It also utilizes the Vue Router for navigation and local storage for persistent user data. Key features include:

1. User Data Management: The `getCurrentUser` function fetches the current user's details, including their first name, last name, and email, from the backend. This data is used to personalize the page and facilitate API requests.
2. Event Fetching: The `getPublicEvents` function retrieves a list of public events from the backend and stores them in the `publicEvents` reactive variable. The `getHostedEvents` function fetches events the current user hosts, storing them in

the `hostedEvents` reactive variable. Both functions map backend responses into a standardized format for display.

3. Sidebar Responsiveness: The sidebar adjusts dynamically based on the window width using the `updateSidebarWidth` function, ensuring a responsive design. The width updates in real time when the window is resized.

4. Dynamic Filtering and Tabs: The `filteredEvents` computed property filters events based on the active tab (`attending` or `hosting`), enabling users to view relevant events easily. The `activeTab` variable manages the current selection, dynamically updating the displayed events.

5. Processed Events: The `processedEvents` computed property decorates events with `isHost` and `isGuest` flags based on the current user's relationship to the event.

6. Navigation and Routing: Navigation items (`navItems`) are defined with labels, paths, and optional query parameters, providing routes to related pages such as Account.vue and Public.vue. If no user ID is found on the page load, the script redirects the user to the login page for authentication.

7. Lifecycle Management: In the `onMounted` hook, the script initializes the page by clearing local storage, validating the user's session, and fetching events from the backend if they have not been loaded yet. The `onUnmounted` hook removes the window resize event listener to clean up resources.

The EventCard.vue component is a reusable Vue component that displays detailed information about a single event. It uses props to define the event's details and to determine the actions available to the user. The props include:

1. `event`: An object containing the event's details, such as its title, type, host, date, time, venue, and related links.

2. `isHost`: A boolean value indicating whether the user is hosting the event.

3. `isGuest`: A boolean value indicating whether the user is attending the event as a guest.

The EventCard.vue component displays information about an event and provides user-specific actions based on their relationship to the event. It renders event details like the title, type (public or private), host name, date, time, and venue, along with an invitation image and a hyperlink to the venue's address. The component adjusts its actions depending on whether the user is a host or guest. Hosts can edit the invitation,

view the guest list, and print event details, while guests can view the invitation. Both hosts and guests can access the event's chat via a dedicated button.

The script builds a highly interactive and responsive events page serving hosts and attendees. It personalizes content based on the current user data, maintains data integrity by fetching and processing events from the backend, and gives easy navigation and filtering capabilities. This page acts as the hub for event management on the AllEvently website through the combination of dynamic elements and responsive design principles.

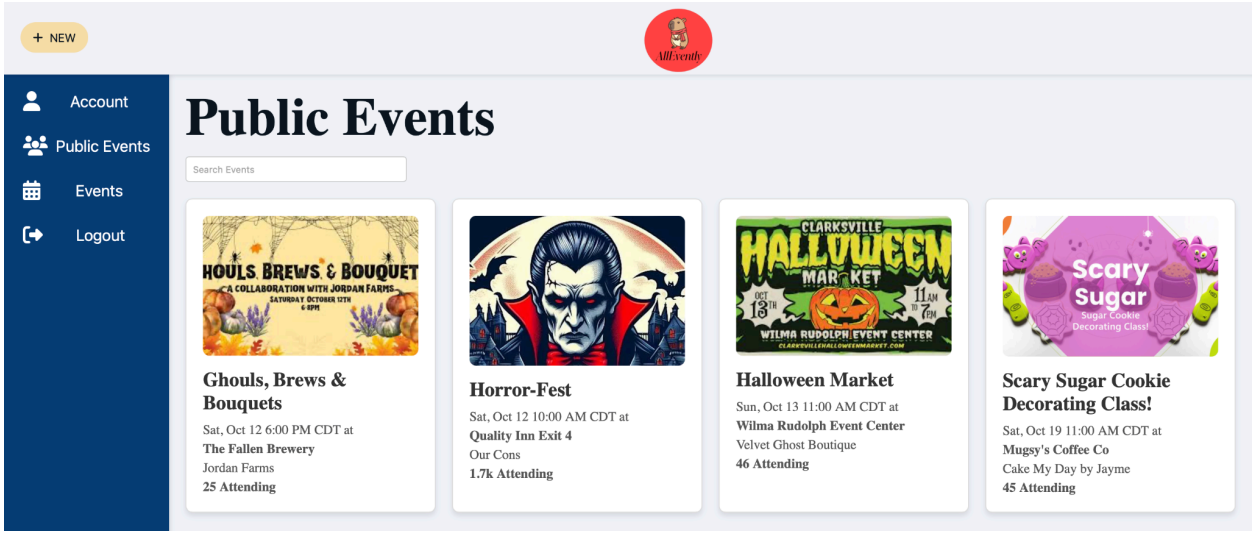


Figure 4. Public Events page

Public Events Page

The PublicEvents.vue page lists public events for users to explore. It displays the same top panel, sidebar, and search bar components from the Event.vue page with a grid of public event cards rendered using the `PublicEventCard` component. The page is designed for user experience, integrating Vue.js functionality and a clean visual layout. Key features include:

1. Top Panel and Sidebar: The TopPanel.vue component displays the application's logo. The Sidebar.vue component is conditionally visible (`isSidebarVisible`) and has navigation across pages while maintaining responsiveness.
2. Search: The SearchBar.vue component lets users quickly search for public events.
3. Event Display: Public events are in a grid format using the PublicEventCard.vue component. Each card includes the event title, date, time, venue, location, attendee count, and an image.

4. Dynamic Event Rendering: The script contains a place for dynamic data fetching.

The PublicEventCard.vue component is another reusable component that displays public event details. It also uses Props that accept several props to define an event, including:

1. `title`: The event name.
2. `date` and `time`: The event schedule.
3. `Venue` and `venueLink`: The venue name and a hyperlink to its location.
4. `location`: Additional location details.
5. `attendees`: The number of people attending the event (supports both string and number).
6. `imageUrl`: The URL of the event image.

This page is a central hub for browsing public events, offering a clear layout of events. The script is structured to integrate dynamic data fetching for live updates.

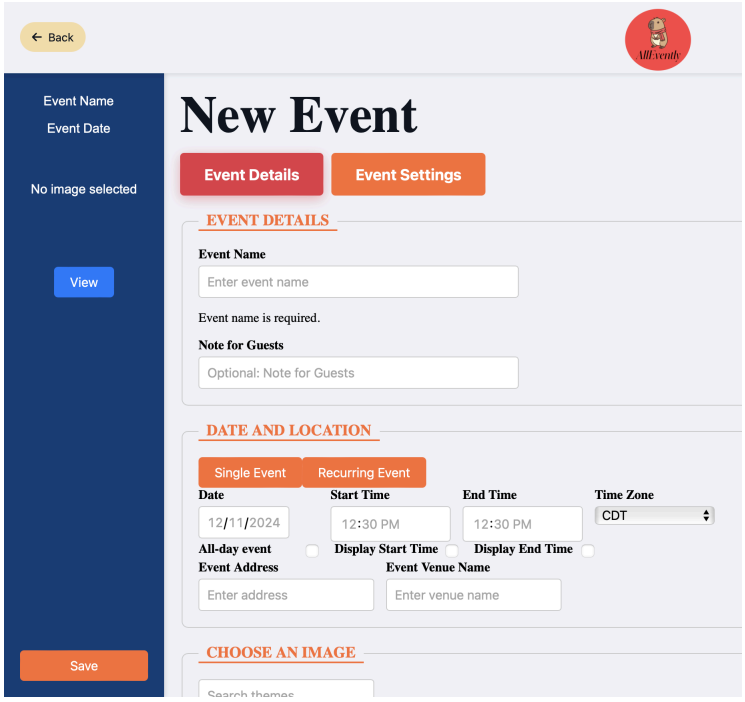


Figure 5. Event Creation Page

Event Creation Page

The EventCreation.vue page allows users to create and customize events with comprehensive tools. It incorporates a different top panel, TopPaneWithBack.vue, and sidebar, SideBarWithPeview.vue, for different functionality but resembles the same previous look. The page integrates Vue.js functionality with various interactive components to simplify event creation. The key features include:

1. Top Panel: The TopPanelWithBack.vue component displays the application's logo and provides navigation to return the user to the Events.vue page.
2. Dynamic Sidebar Preview: The SidebarWithPreview.vue component is a computed property that generates a real-time preview of the event, displaying the selected details, images, and styles in the sidebar.
3. Event Details Input: Users can provide event details such as name, location, date, start and end times, and additional notes for guests. A validation function ensures the event name is valid before proceeding.
4. Google Maps Integration: The page uses Google Maps Autocomplete for address input, enabling easy location selection and displaying a dynamic map preview of the event location.
5. Customization Options: A gallery of themed invitation images allows users to search and select a suitable design. Users can customize font styles to match the event's theme, including font family, size, bold, italic, underline, and color options.
6. Advanced Settings: The page supports creating recurring events with frequency, intervals, and end conditions options. Additional settings include limiting guest numbers, requesting RSVP responses, and adding links or resources related to the event.
7. Event Creation: The `createEvent` function gathers all input data and sends it to the backend server for processing, ensuring the event is saved with the specified configurations.

This page provides a feature-rich environment for event creation, combining advanced customization options. Its integration with external services like Google Maps ensures a pleasant user experience when planning events.

Invitation Page

The Invitation.vue page is another dynamic, reusable component displaying event invitation details. It provides users with all the relevant event information while offering interactive options like RSVP and chat links for communication between the host and other guests. Key features include:

1. Dynamic Event Details: The component uses props to receive the event details, such as the event name, venue, address, date, time, and headline.
2. Interactive Features: Includes an RSVP button with an `onRSVPClick` method to handle RSVP's. The RSVP status and button styling are dynamically controlled

through `rsvpStatus` and `rsvpClass` props. Users can also navigate the chat page using a chat hyperlink.

3. Event Host Information: Displays the host's name and a link to additional details about the host. A dedicated section for host notes allows personalized messages to guests.
4. Event Enhancements: Includes a Google Maps preview for the event venue, making it easy for attendees to find the location. A wishlist link, added on by the host, lets users view the event's wish list.
5. Custom Styling: Displays the chosen background image, providing the event's theme to the guests.

This component creates a visually engaging experience, effectively communicating all event details while providing practical features for guests and hosts.

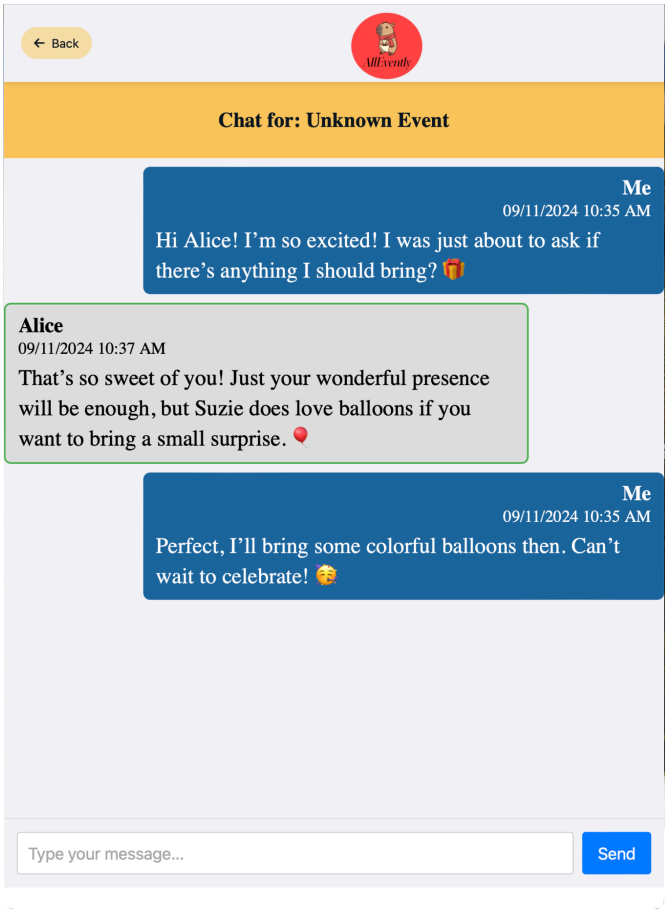


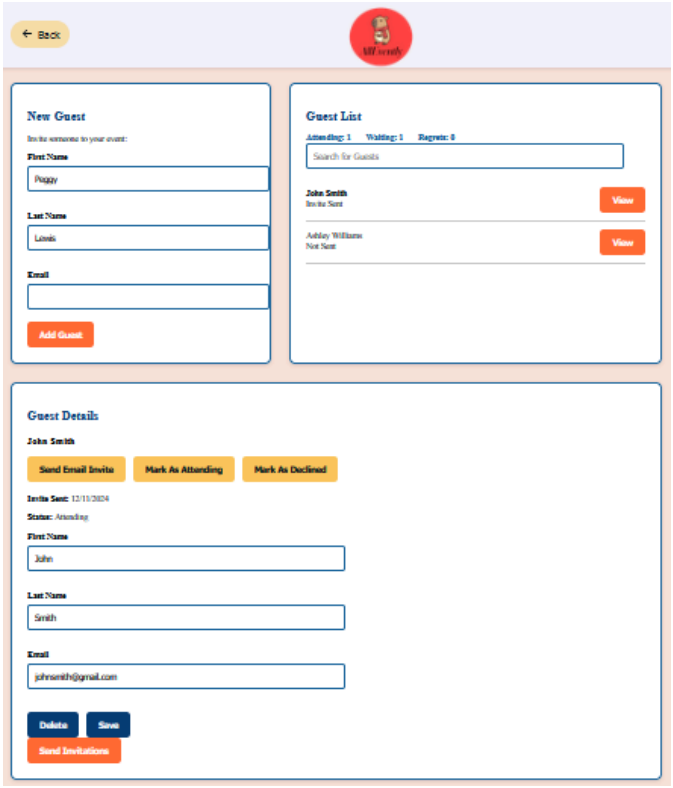
Figure 6. Chat page

Chat Page

The Chat.vue component provides live messaging where users can converse about a specific event. It is a reusable Vue component designed to dynamically render chat messages and update them in real time based on user interaction. Key features include:

1. Dynamic Event Context: The component accepts an `eventId` as a prop, which helps to determine the associated event. The `eventName` is dynamically updated based on the `eventId`, ensuring the chat is contextually tied to the correct event.
2. Interactive Chat Interface: Users can view past messages and send new ones. Messages are displayed with the sender's name, timestamp, and text.
3. Auto-Scrolling: The chat view automatically scrolls to the most recent message whenever the component updates, keeping the latest messages in view.
4. Message Sending: A text input allows users to compose and send messages. Sent messages are added to the `messages` array, which updates the chat display in real time. Input validation prevents empty messages from being sent.
5. Top Panel: The TopPanelWithBack.vue component navigates the Events.vue page.

The Chat.vue component also has a place for data fetching the event names. This component facilitates event-related communication, creating a connected and engaging environment for all participants.



Guest List Page

The GuestList.vue component is designed to manage and display a guest list for an event, offering tools for inviting, updating, and managing guest statuses. Key features include:

1. Dynamic Guest Management: Guests are represented by an interface, Guest, that includes fields like name, email, invite status, and RSVP status. New guests can be added dynamically using a form, and existing guests can be updated or deleted.
2. Filtering and Searching: A search feature filters guests by name, making it easier to locate specific individuals in a large list.
3. Guest Actions:
    a. *View* Button: Provides the guest's detailed information that can be viewed and edited.
    b. *Send Invitations* Button: Email invites can be sent to individual guests, updating the `inviteSent` status and the invitation date.
    c. *Mark RSVP*: Guests can be marked as "Attending," "Waiting," or "Regrets" to reflect their RSVP status.

    d. *Delete Guests*: Guests can be removed from the list as needed.

4. Counts and Statistics: The component tracks the number of attendees based on their RSVP status (Attending, Waiting, or Regrets) and updates these counts when statuses change.
5. Invitation Sending: An invitation link can be configured and sent to all guests via the `sendInvitations` method, which uses an API call to the backend.

This component provides guest management for event organizers, offering straightforward functionality for managing invitations and RSVP statuses.

Typescript Files

These files represent backend functions that process requests from the user interface and make requests to the database.

authentication.ts

authentication.ts handles the backend login functionality. It accepts an email and password. It pulls the encrypted password from the database and uses Bcrypt to pull the plaintext value from the encrypted password. Then, it calls the database function Authenticate_User to determine the credentials' validity by comparing the retrieved, decrypted password to the provided password.

signup.ts

signup.ts handles the backend signup functionality. It accepts an email and password and encrypts the password using the Bcrypt library. Then, it calls the database function Create_Account to determine the credentials' validity and send the information to the database if the credentials are valid. It then sends a message back to the frontend with the status code notifying the frontend of the success or failure.

currentuser.ts

currentuser.ts determines a user's first name, last name, and email based on a session ID. It calls the database function Get_User, which retrieves this data and sends it to be used in the frontend through the post method message the credentials were sent with.

attendingevents.ts

attendingevents.ts retrieves all events attended by a user based on their email. It calls the database function Get_Attending_Events to retrieve the data. It then formats the data retrieved into an array, and sends it to the frontend to be used in page displaying.

hostedevents.ts

hostedevents.ts retrieves all events hosted by a user based on their email. It calls the database function Get_Hosted_Events to retrieve the data. It then formats the data retrieved into an array, and sends it to the frontend to be used in page displaying.

publicevents.ts

publicevents.ts retrieves all public events. It calls the database function Get_Public_Events to retrieve the list of public events. It then formats the data retrieved into an array, and sends it to the frontend to be used in page displaying.

eventcreation.ts

eventcreation.ts submits parameters to the database to create new events. The parameters required to perform this operation are the host's name, email, and the event location. Other optional parameters include the following:

- Name of the event: the default 'My Event'
- Start date that includes the time: the default is the day after the current day
- End date that includes the time: the default is none
- Event time zone: the default is UTC
- Invitation layout: the default is 'center'
- Background image
- Font background color: the default is white
- Font color: the default is black
- Font: the default is 'italic bold 20px arial,serif'
- Whether the event is public: the default is false
- Whether the event reoccurs: the default is false
- The recurrence frequency in days: the default is null
- The date when reoccurrences end: the default is null
- Whether to request a child count: the default is null
- Whether to limit additional guests brought by guests: the default is false
- What the maximum number of additional guests: the default is null
- Whether to notify the host when a guest RSVPs: the default is true

The backend then takes note of which parameters are required and which are allowed a default value. It merges the information based on what was provided and inserts default values where necessary. A status code and message are then sent back to the frontend to confirm that the event was created successfully or to notify of failure.

Data Files

Our project boasts 24 images that we can use to compile invitation posters. We found these images on www.unsplash.com. Various artists and the citations provided these images for these images can be found in Credits.txt in the documents folder.

Deploy/Build Instructions

Vercel Project Deployment Instructions

First, go to https://github.com/BenStorm2514/AllEvently and create a fork of the repo.

Then, create an account on www.vercel.com and choose the hobby plan or higher. Then, create a new project. Continue with GitHub to connect to. Then log in to your GitHub account. You will be asked to select a git repository. Select the repo fork you created.

After it is created, it will offer deployment settings. Set the root directory to AllEventlyProjectFiles/frontend and set the build command to npm i --save-dev @types/node && npm run build as shown in figure 7.

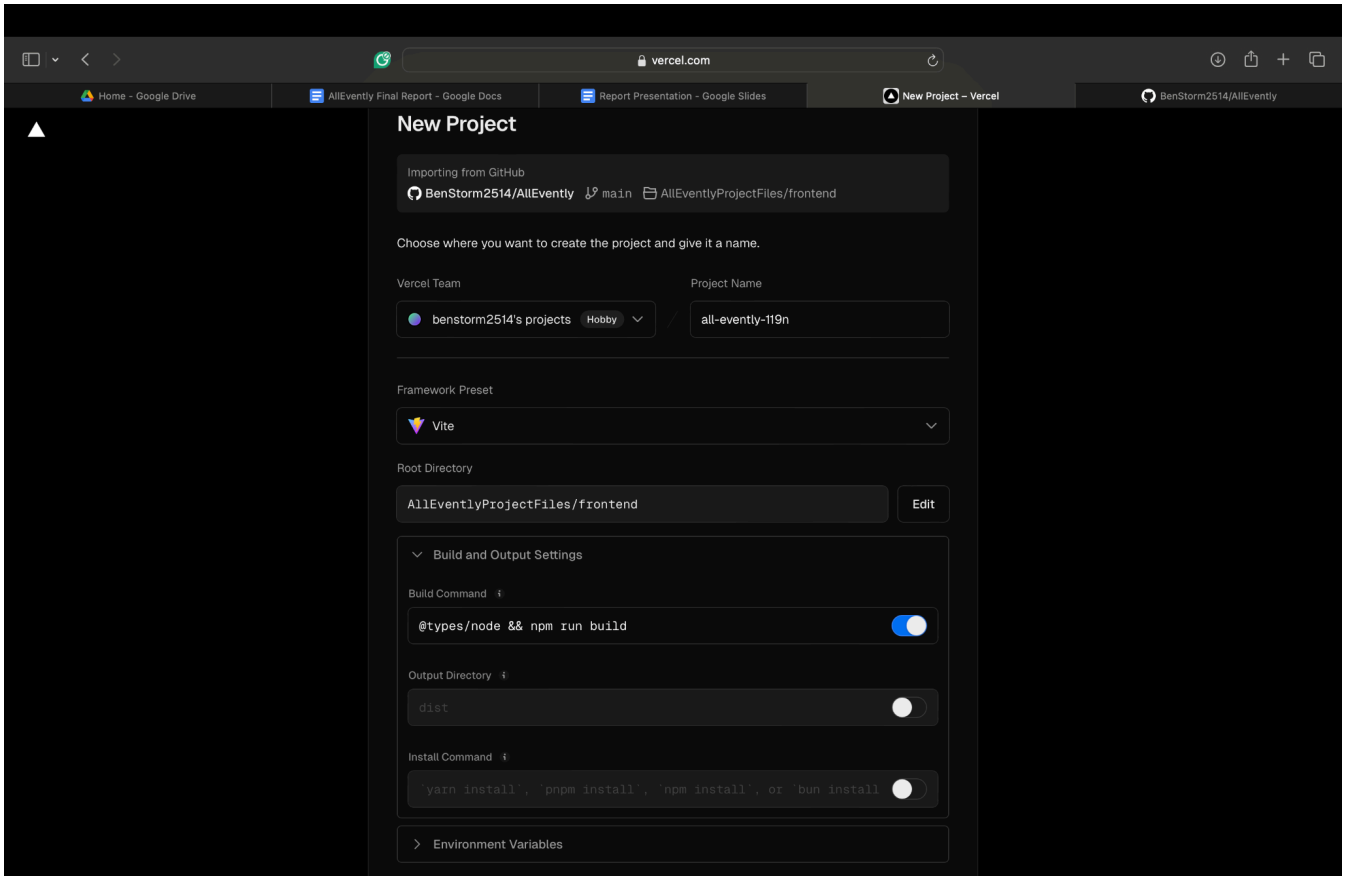


Figure 7. Initial deployment settings for the Vercel Project

You will also need a Google Maps API key. To get a Google Maps API key, go to the Google Cloud Console and log in with your Google account. Create a new project by clicking the project dropdown at the top and selecting New Project. Name the project, like "AllEvently Project," and create it. Once the project is created, go to APIs & Services > Library in the left sidebar and enable the Maps JavaScript API, Places API, and Static Maps API. After enabling the APIs, go to APIs & Services > Credentials and click Create Credentials > API Key. Copy the generated API key and edit it to restrict its usage. Under Application Restrictions, choose HTTP Referrers (websites) and add http://localhost/* for local development and https://your-vercel-domain.vercel.app/* for production. Also, restrict the key to the APIs enabled earlier and save the changes.

Once you have the Google Maps API key, go to the Vercel Project settings. Go to the Environment variables tab, create a new variable named VITE_GOOGLE_MAPS_API_KEY, and set the Google Maps API key as the value.

Vercel Postgres Database Deployment

Open the Storage tab in the account that controls the Vercel deployment listed. Create a database and choose Postgres. After it is created, select the project tab in the created database. Select the project created in the previous step and press the connect project button.

Next, go to the file git repo and go to Documents->Database->AllEventlyDB.sql. Copy the contents of this file and go back to the Vercel database. Go to the query tab, paste the SQL code into the prompt, and run the query.

Known Bugs

In events.vue we have trouble reading the results of our event fetching functions.

The links to the event chats, invitations, and event editing do not work.