CS 501 E1: Mobile Application Development
Owen Strumwasser, Richard Lin, Eric Nohara-Leclair
Prof. Czik
Fall 2025

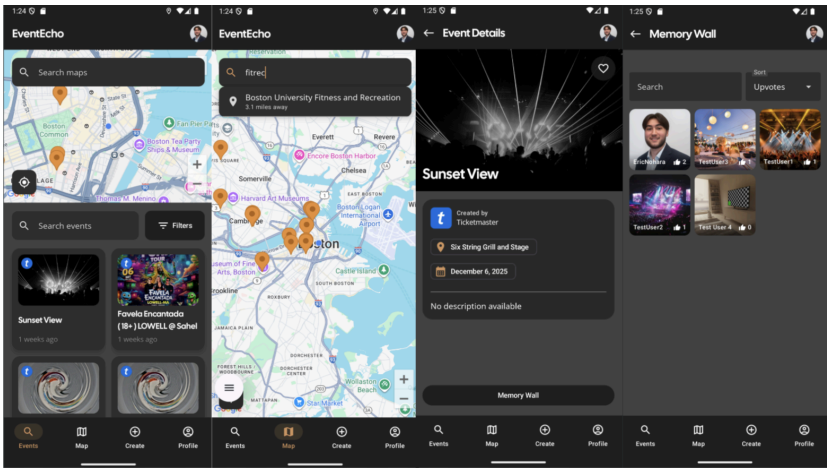
Event Echo: Final Report

I. OVERVIEW

a. Problem Definition & Solution

After attending an event of any kind, many people are left without anything to remember it by. This problem gets worse if they frequently attend events as it can be challenging to distinguish one experience from another. Additionally, there is no centralized social network to track these events, share a memory, and see how others experienced these events. If you host an event, it can be difficult to receive honest and insightful feedback from attendees. Finally, it can be hard to judge whether you would enjoy an event before attending it, as purely written reviews have a limit to expressing the true experience of attendees.

Event Echo solves all of these problems by providing a social platform for sharing events and memories. Users can browse events from Ticketmaster or create their own. Just like the popular app BeReal where you can upload a single image a day, Event Echo allows users to upload a single memory consisting of an image and a caption to an event. As users aggregate their memories for an event, they build out a full memory wall that fully encompasses the experience of the event. Users may browse through these event walls to look back on previous experiences, check out how events look before they happen, or view the experiences of attendees to your events. To supplement this core idea, our application provides several event-related functionalities such as browsing through them on a map or scrollable grid with useful filters. Finally, the app hosts social features, allowing users to upvote events of other users.



In short, Event Echo provides a place to remember and share events and memories with others while keeping track of all attended events and makes it easy to find nearby events based on defined filters.

b. Challenges and Lessons Learned

We ran into many challenges while developing Event Echo. The scale and wide range of features was difficult to implement as inexperienced Android mobile developers.

One large challenge we faced was simultaneous, asynchronous development. While our team was very communicative and collaborative, difficulties with merging versions of any large-scale application with conflicting files are unavoidable. There were some instances during development where merging new features caused conflicts that made traditional merging impossible. In these cases, we had to manually integrate the changes from each branch. However, over time we communicated better about conflicting features and got better at managing who works on what feature at what time to mitigate large conflicts.

The designing and planning process for the application was also hard because every feature we added had to fit with the other aspects of the application to ensure cohesion. We initially had many features we wanted to implement, but over iterations of development, we filtered out what we wanted to include and what features did not fit with our app and were not worth the trouble. Thus, we focused more on the mandatory features for our app that were difficult to implement. The lesson we learned from our development challenges was the importance of planning, since some features needed to be implemented in a certain order. Solid planning throughout the development process allowed us to have a very concrete idea of what we'd done, what we needed to do next, and the overall structure of the application.

Deciding on the look and feel for the application was also a difficult and iterative process. We worked together on a Figma file to build out a general color theme and design for how the app should look, while using Figma AI to build a quick visual mock up for how the app would look with our layout. This helped us in our development process as we had a skeleton for the screens to look off of when designing our components.

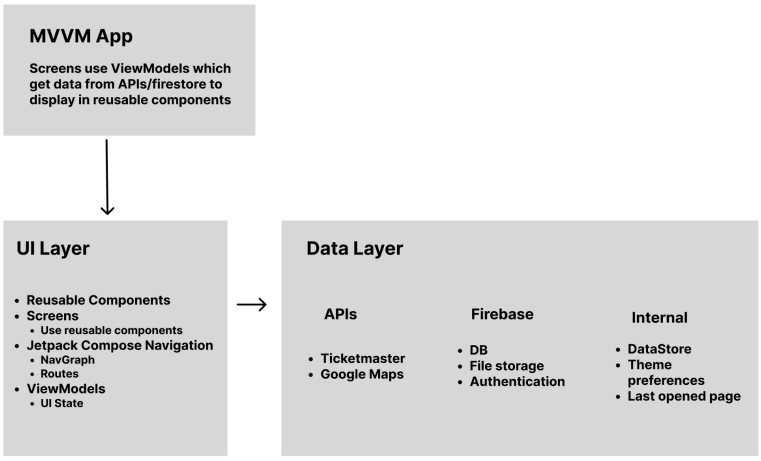
The development itself also proved itself to be quite difficult in general, especially the components that none of us had worked with before, like Firebase, Google Maps, Places, and sensors. We overcame these challenges by reading through documentation, reviewing sample code, watching YouTube tutorials, and consulting AI when needed. Using a wide range of sources allowed us to filter out the useless information and find what we really needed for our project and how to use it.

II. TECHNICAL IMPLEMENTATION

a. Architecture

Our application uses the MVVM architecture, with the Model, View, and ViewModel separated. We broke up our project into screens, view models, reusable components, and repositories to modularize the project. We used Jetpack Compose Navigation with a NavGraph and Routes to handle our internal navigation and inject relevant view models to our various screens. Our screens shared reusable components when possible to follow the DRY principle. We used MaterialTheme color schemes to define our own color

scheme for the app, allowing for easy light and dark mode toggling. Our data layer handled API calling, Datastore, and Firebase operations, exposed via repositories that are used in our view models.



b. API Usage

Ticketmaster: We used the Ticketmaster API for retrieving events from Ticketmaster. We stored these events in our database in a generic format so our app works with **both** user defined and ticketmaster events. As Ticketmaster provides an enormous number of events, we only retrieve events that the user requests, and store them in our database. For example, if a user requests events within 50 miles today, we only retrieve those events from the API and store it in our database. The next time a user requests those events, we return the events from our database instead. This way, the event database is built out as users request for new events.

Google Maps: Our application has two screens that use a map, the Home screen which has a small map near the top and a full map screen where it takes up the entire page. Users can fully interact with these maps that are integrated with our application's custom filtering and clickable pins at locations for nearby events. We also used the Google Maps API for geocoding throughout the application.

Places API: Similar to the Google Maps API, we used the Places API to provide autocomplete features for location based searches within the app. This lets a user start typing in an address or location and they will see a dropdown for possible locations.

c. Compose Usage

Jetpack Compose and its presets played an important role in building Event Echo's interface. For our UI we had a Scaffold with bottom and top app bars, and a Theme application styling. We also used many composables such as Columns/LazyColumns, Buttons, Surfaces, Icons, and more when appropriate.

We created reusable composables to avoid clogging files for screens with too much code. Our application has a dedicated folder for smaller composables, which are referenced throughout the application.

d. Firebase Usage

Authentication: We used firebase email and password authentication for users to sign in and sign up for the app. This was done to balance simplicity of set up with functionality, allowing us to work on other more important sections of our app while still having secure authentication.

Database: We used a firestore database for storing persistent data for our application. This data includes user account information, event data, and memory wall data. This data needed to be stored on firebase so changes made by one user would be visible to all.

File Storage: We used firebase storage for storing image files for our application. This includes user profile pictures, event pictures, and memory pictures. These images are stored so that other users can view the images.

e. Sensor Usage

We used the GPS sensor and the camera for our application. The GPS sensor is used to detect the user's current location to set their default location to find and post events to, and the camera is used to quickly and easily add images to the app, either as a profile picture, an event picture, or a memory wall entry.

III. TEAM ENGAGEMENT AND PROCESS

Our team was highly collaborative when developing Event Echo. We met in-person whenever we needed to divide labor and discuss next steps. This was approximately every other week. During these meetings, we would discuss our current progress, future features to, and how to best split up the work according to our preferences and skillsets.

We used a GitHub repository where the dev branch held the most recent version of our application. We created feature branches with staged changes that were either merged directly into dev using pull requests or used as a reference to integrate the changes manually to avoid merge conflicts. This allowed us to work on different features at the same time while avoiding conflicts.

In addition to our team meetings, we maintained an active text message group chat where we would often update each other about progress, issues, or functionality related to the project. We made sure to mention if we created a branch or needed another team member to review and test a new feature. This was essential to our development process because we each had different responsibilities, mental models of our application, and working habits that needed to be coordinated properly to ensure project success.

IV. TESTING APPROACH AND RESULTS

Our team utilized multiple testing methods to ensure a high quality final product. We used logging throughout our application whenever making an API call, a database lookup, a navigation, or using a sensor. This was done to easily diagnose issues in log cat after a crash. Additionally, we utilized some

automated testing to ensure that our components were working as expected. We also used test data for quick UI testing, and interacted with the app as an end user would to make sure we did not miss any edge cases. Finally, we used the firebase console to make sure that the data and files we were storing were correct.

As a result, our app is a polished and functional app that does not crash. It handles every edge case (that we thought of) and is reliable. This makes for an easy to use, simple, and reliable app for sharing events and memories with others.

V. AI REFLECTION

a. AI Use in Development

AI played a role in our application's development process. We used it for generating screen wireframes and assisting with project code directly, such as generating, debugging, and troubleshooting. In many ways, it is quite helpful and accelerates the development process and clears up confusion, but it also has limitations and almost all of its suggestions require thorough evaluation and restructuring.

b. Tools Used, Where & How

Figma: We used Figma for screen wireframes, most of which were created manually using the drag-and-drop system but a couple were generated by Figma's AI, which we used as a skeleton/mockup to base some of our screen designs off of.

ChatGPT: We used ChatGPT for generating, debugging, and troubleshooting code. Almost all of its suggestions require thorough evaluation and restructuring.

Firebase Gemini Assistant: We used firebase gemini assistant to evaluate which firebase products best suited our project and how to best integrate them with our app. Although it provided good suggestions, the official firebase Kotlin documentation was much more helpful.

Gemini: Like ChatGPT, we used Gemini (Android Studio's built-in version and Google's web version) for generating, debugging, and troubleshooting code. Almost all of its suggestions require thorough evaluation and restructuring.

c. Example Prompts

General Examples

- "How can I design a system that uses both live location and custom google maps navigation so that we don't fetch from ticketmaster api too often?"
- "Would a high accuracy or low accuracy priority be better?"
- "How do I interface with firestore within my kotlin project to upload data to our database?"
- "Generate a simple wireframe for the profile page with generic fields for username, profile picture, and bio."

Detailed Examples

- "Can you adapt this to a new composable so that it has a card on each side like this" (Composable for a list of events & image of a desired Event Grid also provided as context)
    - **Response & Corrections Required:** ChatGPT responds with a new composable, but it has some extra whitespace, it's not fully integrated with the theme, and doesn't gel with our viewModel and data, so these need to be resolved manually.
- "Which color in the theme is the card taking after?"
    - **Response & Corrections Required:** ChatGPT responds by saying that a Card uses MaterialTheme.colorScheme.surface, which can be used to figure out why the card displays as a certain color, eventually deciding to use a Surface instead.

d. General Corrections Required

Styling Fixes: The screen wireframes generated very bare bones skeletons for our screens without the necessary styling our app required with our own custom theme.

Debugging Context: Utilizing AI to help debug is useful but also difficult to do reliably since the context of our codebase is so large. It is usually faster to debug manually, unless the bug fix is simple to detect.

App-Specific Instructions: Our app has some pretty specific functionality that AI does not quite understand. Thus, we have to adapt the output from AI to our own specific needs for what we want the app to do. For example, AI is good for showing the general structure for uploading data to firestore, but we had to adapt it to follow our specific schemas and app rules.

Reviewing & Refactoring: We needed to review and refactor any AI-generated for quality, security, and Kotlin/Compose idioms. Fully understanding what every line of AI-generated code does and making sure that it does not have any issues were essential to this process. Because of this, we prioritized code that was concise and familiar to us given our understanding of best general and Android-specific development practices. We did not use it as a crutch to develop our application for us, but rather a tool.

e. Analysis of Helpfulness and Limitations

These AI tools were helpful as they accelerated the design and development process. For the UI, for example, the wireframes we generated using Figma guided how we styled certain pages. They were also helpful for the project code because they can create samples of certain elements, especially considering these LLMs are aware of Jetpack Compose and its capabilities and structuring, which is helpful for generation but also debugging. If you are very specific about prompts and context, AI can help clear up confusion as long as its responses are critically evaluated and/or fact-checked.

However, these AI tools also come with many limitations, the most notable being that we had our own perspectives of how our codebase is structured and how we want certain elements to look and integrate with the rest of our project. Any code generated using AI had to be adapted to gel with the rest of our application, like certain state variables or a ViewModel/NavController. These AIs also can't see what

they’re creating, so usually their responses, if visual elements are at play, would display strangely or not integrate with a theme and need adaptation.

It is also worth noting that often, the contexts of these LLMs are outdated or contain outdated information, which is especially problematic given our application relied on the Android SDK, which is constantly evolving. These LLMs often used elements that did not work with our development environment.