

---

---

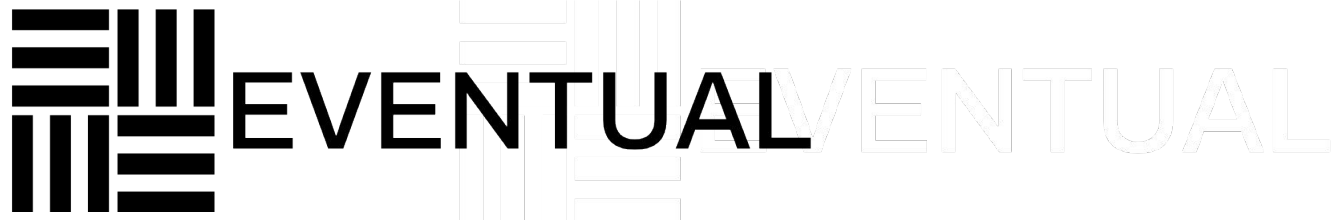
# Citrus - Final presentation

Brought to you by, you guessed it... Citrus

---

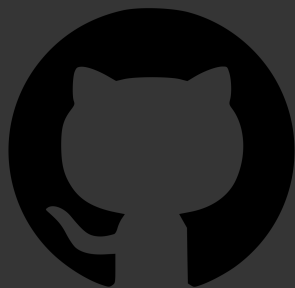
# So, what problem did we solve?

Eventual aims to connect users with experiences that pique their interest while simultaneously allowing event organizers to easily publicize their events and advertise them.



# Process Discussion:

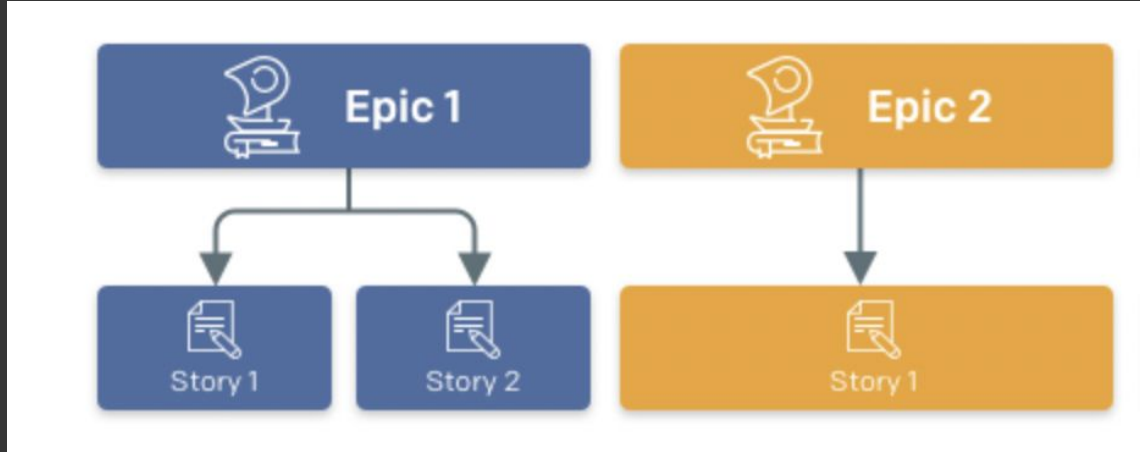
Workflow



# Assigning Tasks



# Process Decisions



---

# Technical discussion and software architecture

---

---

# Technical item #1

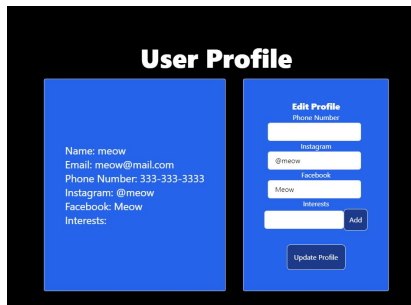
## Using a mix of server and client components

With Next.js 13+, documentation everywhere advocates for mixing server and client components

Issue is, both have access to completely different sets of functionality and are hard to mix appropriately

Using them as intended greatly simplifies the development workflow

---



The image shows two side-by-side forms on a black background. The left form, titled 'User Profile', has a blue background and displays user information: Name: meow, Email: meow@mail.com, Phone Number: 333-333-3333, Instagram: @meow, Facebook: Meow, and Interests: (empty). The right form, titled 'Edit Profile', has a blue background and contains input fields for Name, Email, Phone Number, Instagram, Facebook, and Interests, along with an 'Add' button. Below these fields is an 'Update Profile' button.

---

## Technical Item #2

How do we interface with the database?

For our first sprint, we had not settled on how to interact with our database to persist information.



**Prisma**

Running raw SQL is possible, but not recommended due to risk of attacks. **Would have been the most customizable, but also least out-of-the-box friendly**

Settled on using Prisma

---



---

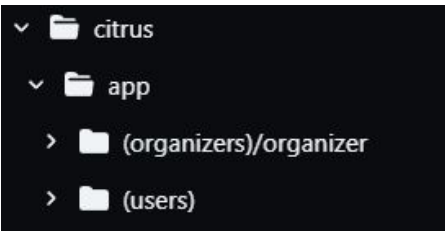
## Technical Item #3

How do we have different experiences for users and organizers on the same domain?

Solving this problem required an entire rewrite of our application to use Next.js' support for multiple layouts

Can be seen in routing on the right – all organizer related pages are grouped under **(organizers)** and are accessed under **/organizer**

---



---

# Software Architecture

- Don't have conventional components, backend/frontend distinction or classes
    - Analogous to classes, we have database models (`schema.prisma`)
  - Our models have conventional relations with each other
    - E.g. Each event must have an organizer
  - We do have a frontend/backend following the Model-View-Controller architecture
    - Only that it is not necessary with Next.js App router
-



## Software tools

### → **Next.js**

We used Next.js as our web framework, with the newly updated app router.

### → **Prisma + CockroachDB**

CockroachDB is a SQL-based database with strong data integrity. Prisma is an ORM used for easy interfacing.

### → **Stripe, Ably, Vercel**

Our third-party providers for payment processing, realtime messaging and app deployment respectively.

---

# Interesting technical challenges

- Next.js App router had been newly announced when work began
    - Prior to this, **pages** router was the convention
    - Entire overhaul of framework's architecture
  - Managing and persisting login states, intelligent sessions
    - Used NextAuth library to automate this process, enabling simple handling of sign-in detection
  - Integrating non-trivial features such as payments and realtime messaging
    - Would have required from-scratch implementations of WebSockets and other fragile pieces of code that would have been too complicated to finish
-