Turnip

Team 2 - Design Document

Abigail Alderson, Cole Johnson, Keegan Irby, Kevin Cardona, Kyle Pollina

Purpose

With the wide variety of applications like Facebook, Instagram, and Snapchat, users have had to keep up on multiple accounts in order to benefit from the most popular features of each. Our project aims to consolidate the unique aspects of all of these platforms into one simple, easy-to-use application. Creating a Turnip account will allow its users to achieve the simplicity of Instagram, spontaneity of Snapchat, and invite component of Facebook all with very little onboarding.

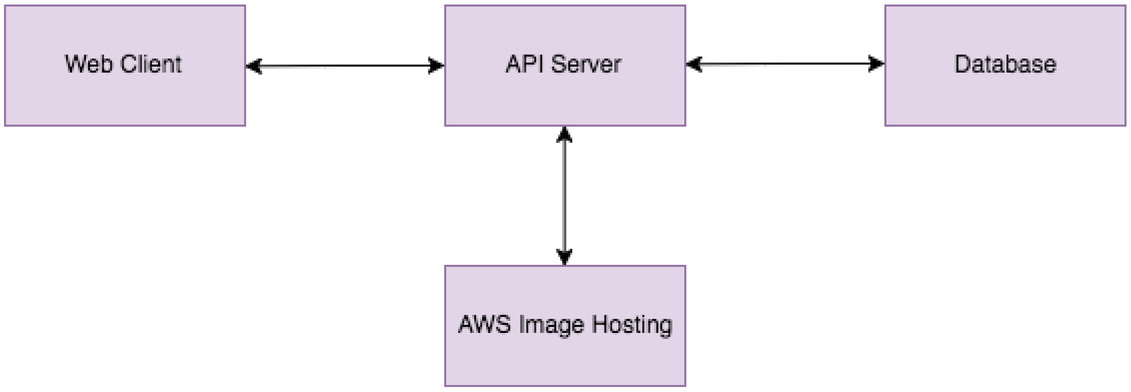
Our functional requirements include:

1. User Profiles
    a. Every user will have a profile on Turnip that stores content from past events as well as other information the user would like to publicly share like phone number, usernames for other applications, etc.
2. Event Invitations
    a. Hosts will be able to create and send event invitations with pertinent information to selected guests. Invitations can be public or private, and specified guests can be selected to have co-host privileges.
    b. Guests will be able to view and accept/reject incoming event invitations.
3. Event Details
    a. Hosts will send information like location, time, and fees to selected guests.
    b. Only accepted guests may view this information.
4. Easy Pay
    a. Hosts may request a fee from guests before their acceptance to the event via Venmo integration.
    b. Guests may quickly and easily pay these fees in order to gain access to the main event page.
5. Content Threads
    a. Each uploaded piece of content may be commented on or liked/disliked by any other guest that attends the event.
    b. Hosts will have administrative power and will be able to delete threads.
6. Memory Albums
    a. As content is added to an event page, Turnip will auto-create a memory album that encapsulates the top content of the event and displays it in a nostalgic fashion.

7. Collaborative Music Playlists
    a. Users will be able to collaborate to create a playlist full of their favorite songs via Spotify integration.
8. Content Sharing
    a. Users will be able to export/share content from Turnip to other platforms.
9. Easy-to-Follow Directions
    a. Hosts will specify an event location when creating event details.
    b. Guests can follow simplified directions to the event integrated directly into Turnip.
10. Media Uploads
    a. Users will be able to upload from their devices via Turnip directly, or through SMS text messaging.

Design Outline

Turnip will use a client-server model. The server will communicate with a database to store and receive information regarding user account and event information. The server will use a RESTful API to provide the client with data. The client will be a standard web browser.



Web Client

- The client will be the user's internet browser. Users will use the client to interact and use the Turnip application. Using HTTP requests, the client will be able to communicate with the server to send/receive data. The client will be built using HTML, CSS, and AngularJS.

Server

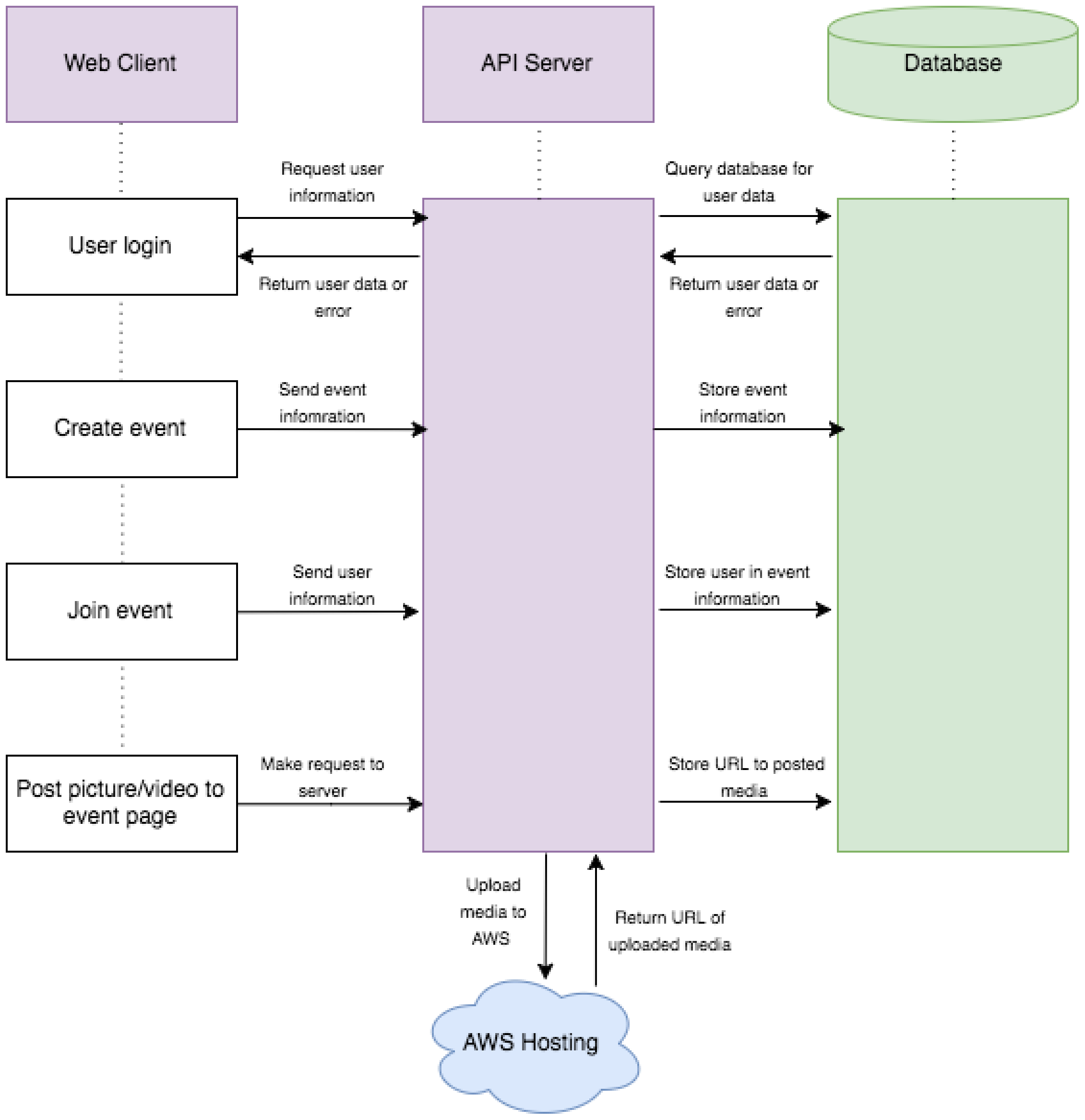
- The server will use a RESTful API architecture to handle HTTP requests. The server will handle routing the user to the correct webpage, and retrieving/storing information from the database. Information from the server will be sent in JSON format. The server will be built using Node.js.

Database

- The database will be used to store all the information for Turnip. This includes user account and event specific information. The database will be queried by the server, then information will be passed from the database to the server, and then to the client. The database will be built using MongoDB.

AWS Image Hosting

- In order to allow users to post pictures and videos to event pages, the media needs to be hosted online for our server to fetch for the client. We will use AWS to host the media that users post on event pages. Our database will store the URL for each piece of media hosted on AWS and allow it to be found by the server.


Web Client
API Server
Database
User login
Request user information
Return user data or error
Query database for user data
Return user data or error
Create event
Send event infomration
Store event information
Join event
Send user information
Store user in event information
Post picture/video to event page
Make request to server
Store URL to posted media
Upload media to AWS
Return URL of uploaded media
AWS Hosting

Design Issues

Functional Issues

Issue: How will users receive event invitations or notifications?

- **Option 1**: SMS message
- **Option 2**: Email
- **Option 3**: By logging into their account

Decision: We decided that the user should have access to all forms of communication and let them choose. We plan on using the Twilio API for the SMS messages. Our backend framework has many mailer packages available for emails.

Issue: How should users view what events they have been invited to?

- **Option 1:** Central Dashboard
- Option 2: Direct links from other users

Decision: To provide a more complete experience, giving the users a dashboard is the best option. Only going off of links in the invites would work, but Turnip wouldn't feel like a complete system. We plan on implementing the dashboard by having one section of upcoming events, and one section of past events.

Issue: How should we order event comments?

- Option 1: Vote based
- **Option 2:** Chronologically

Decision: The options presented promote different cultures of posts. If the posts were to be ordered based on vote counts, users would feel more self conscious about what they post. Ordering the posts chronologically doesn't give any priority to the posts, but it still allows for the individual posts to gather votes and user feedback. This will be useful when creating the memory albums because it will give a better idea of what the guests like.

Non-Functional Issues

Issue: What platform should we build the app on?

- **Option 1:** Web
- Option 2: Mobile

Decision: We decided to use the web as Turnip's platform. Web offers universal access to all types of devices, so there will be less of a barrier to entry for users. Web is also advantageous because we won't have to specialize in two or three platforms. Also, our group has the most experience with web technologies, so we are going to be the most efficient this way.

Issue: What type of architecture should we use?

- **Option 1:** Client-server
- Option 2: Unified

Decision: Our team plans to develop Turnip with a separate frontend and backend, which offers several advantages. It will allow our team to easily and equally split the tasks we need to complete and will also allow for easier deployments when pushing to production. Additionally, if we decide to make the app suitable for a platform other than web, having a pre-built API will make that task much easier.

Issue: What backend framework should we use?

- Option 1: Ruby on Rails
- **Option 2:** Node.js
- Option 3: PHP

Decision: We considered using both PHP and Ruby on Rails because they each have been around for over a decade, while the original release of Node.js was only six years ago. Members of our team have experience with each of the frameworks, but we ultimately chose Node.js. Using Node.js is advantageous because we will be using the same language (Javascript) for both our frontend and backend apps. This will provide less experienced team members with an easier learning curve. Node.js also offers many pre-made packages through npm which will save a lot of development time.

Issue: What frontend framework should we use?

- Option 1: React
- **Option 2:** Angular
- Option 2: Ember

Decision: Each of these frontend frameworks have their advantages, but we decided to go with AngularJS. None of our team members have experience with React, so we decided that wouldn't be a good choice. We ended up choosing AngularJS over EmberJS just because of personal preference.

Issue: What database should we use?

- Option 1: MySQL
- Option 2: Postgres
- **Option 3:** MongoDB

Decision: Our team decided to go with a NoSQL database because of its flexibility. We chose MongoDB because it is the most widely adopted. MongoDB also has a lot of built in features for scaling such as sharding and replica sets. Additionally, the majority of our team has experience with MongoDB.

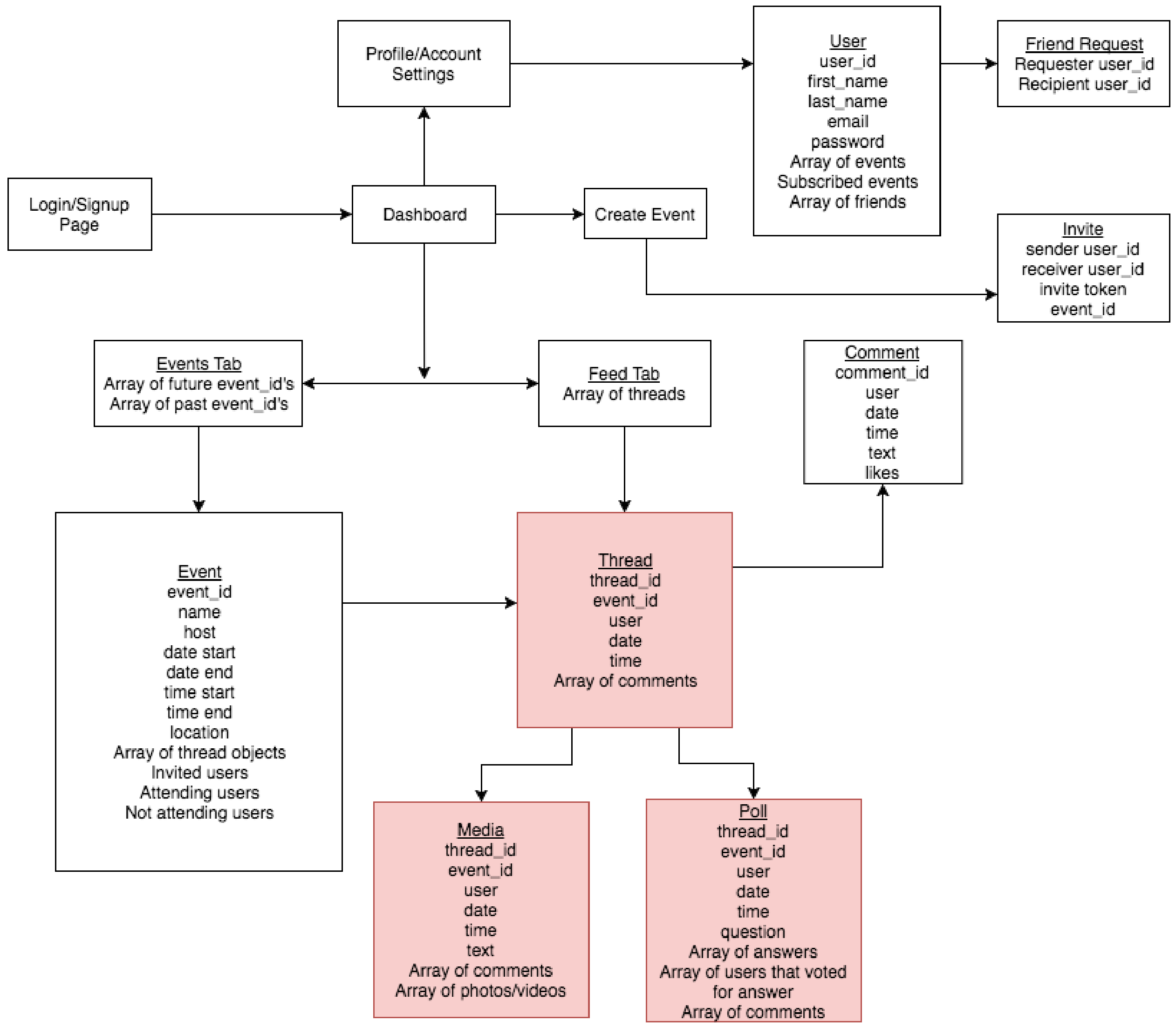
Issue: How should we authenticate API requests?

- **Option 1:** Our own token-based system (JSON Web Token)
- Option 2: OAuth through a social login (Facebook, Twitter, etc)
- Option 3: Sending a username and password combination with each request

Decision: One of the main goals of Turnip is to get away from social giants like Facebook. Because of this, we have decided to implement our own authentication system. Many people either don't like using social platforms or don't have accounts with any of them. A token-based system is the most secure option. Using JSON Web Tokens (JWT) is the best choice because they don't require additional database queries for additional authentication and they are self contained.

Design Details

System Diagram



Key: Red colored squares are related classes. Thread is an abstract class and Media and Poll build off of Thread.

Class Mockups

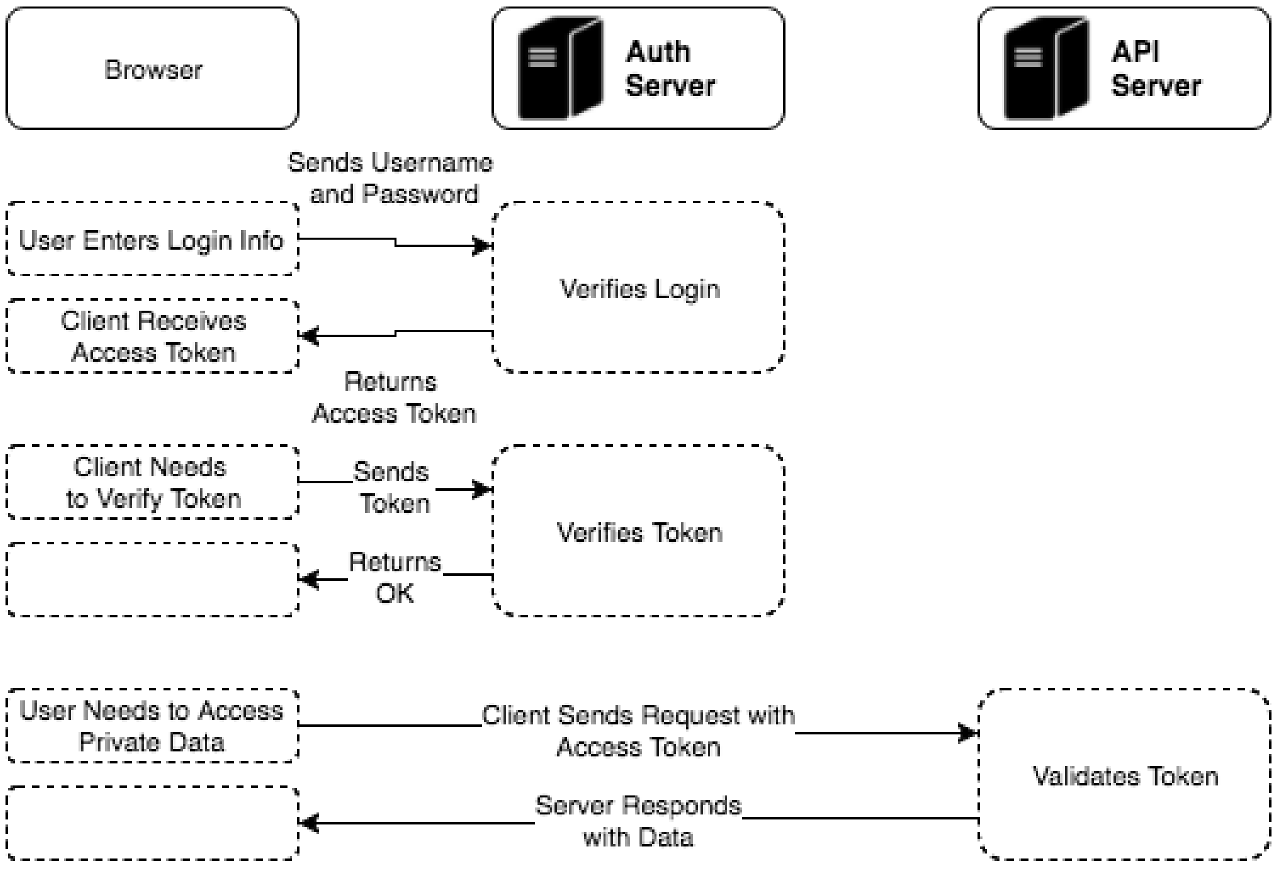
1. User
    a. The user class will contain information about the user such as a unique id, first and last name, email, password, an array of all events that user has been a part of, what users are their friends, and subscribed events (events the user wants to see content from in the Feed tab).
    b. Users can be hosts of events but information on which user is a host will be stored in the Event class.
    c. Users can only see their own profiles. A user can not see the information of another user other than their first and last names. You can not know what user has what friends or anything like that. There will be no public user pages.
2. Event
    a. Every event will have its own event page. The event class will store all of the information about the event.
    b. The event class will hold information about the event, including a unique event id, name of the event, which users are hosts of the event, when it starts and when it ends, location, all invited users, all attending users and all not attending users.
    c. Each event page will contain its own feed of threads in chronological order of when each thread is posted, starting from newest at the top to oldest at the bottom.
3. Thread
    a. A thread is a post by a user to an event page. Threads can be either photo/video albums, text posts, or polls for right now. But since thread is an abstract class we can add more as development goes on.
    b. Users can add comments to each individual thread, unless the user who created the thread disables comments from being posted.
    c. Each thread contains information on the thread, including a unique thread id, the event id of the event it was posted in, the user who posted it, date and time of when it was posted, and an array of comments from users who comment on it.
4. Media
    a. A media thread is a thread that contains either one or more photos/videos and a string of text.

    b. Media threads can have just photos/videos, just text, or both. Photos/videos are not required to post a media thread.

5. Comment
    a. Comments are text replies to a thread post. If a thread allows comments to be posted on it, anyone who is included in an event can comment on a thread.
    b. A comment class will have to include a string of text, which user posted the comment, the date and time of the post, and how many likes the comment has.
6. Invite
    a. An invite is a private invitation from a host to any user the host knows. We want to implement a 2nd degree invite system. What we mean by this is a host can invite anyone they are friends with, and also anyone who is friends with any of their friends.
    b. Invites will contain information on who sent the invite, who received the invite, what event a user is being invited to, and a unique invite token that only the invited user has access to.
7. Friend Request
    a. A user can request to be friends with another user. When this happens a friend request is sent to the recipient. If the recipient accepts this friend request, the sender will be added to the recipient's friend list and the recipient will be added to the sender's friend list.
    b. The only information a friend request will hold is the sender's user id and the recipient's user id.

JSON Web Token Authentication

In order to validate requests made to the server, we are planning on using JSON web token authentication. This will allow us to validate users without having to store any user information in the session.



Browser
Auth Server
API Server
Sends Username and Password
User Enters Login Info
Verifies Login
Client Receives Access Token
Returns Access Token
Client Needs to Verify Token
Sends Token
Verifies Token
Returns OK
User Needs to Access Private Data
Client Sends Request with Access Token
Validates Token
Server Responds with Data

Fig 1. User Authentication

This diagram illustrates that once the client has a valid access token, the server has no need for a username/password.

Sequence Diagrams

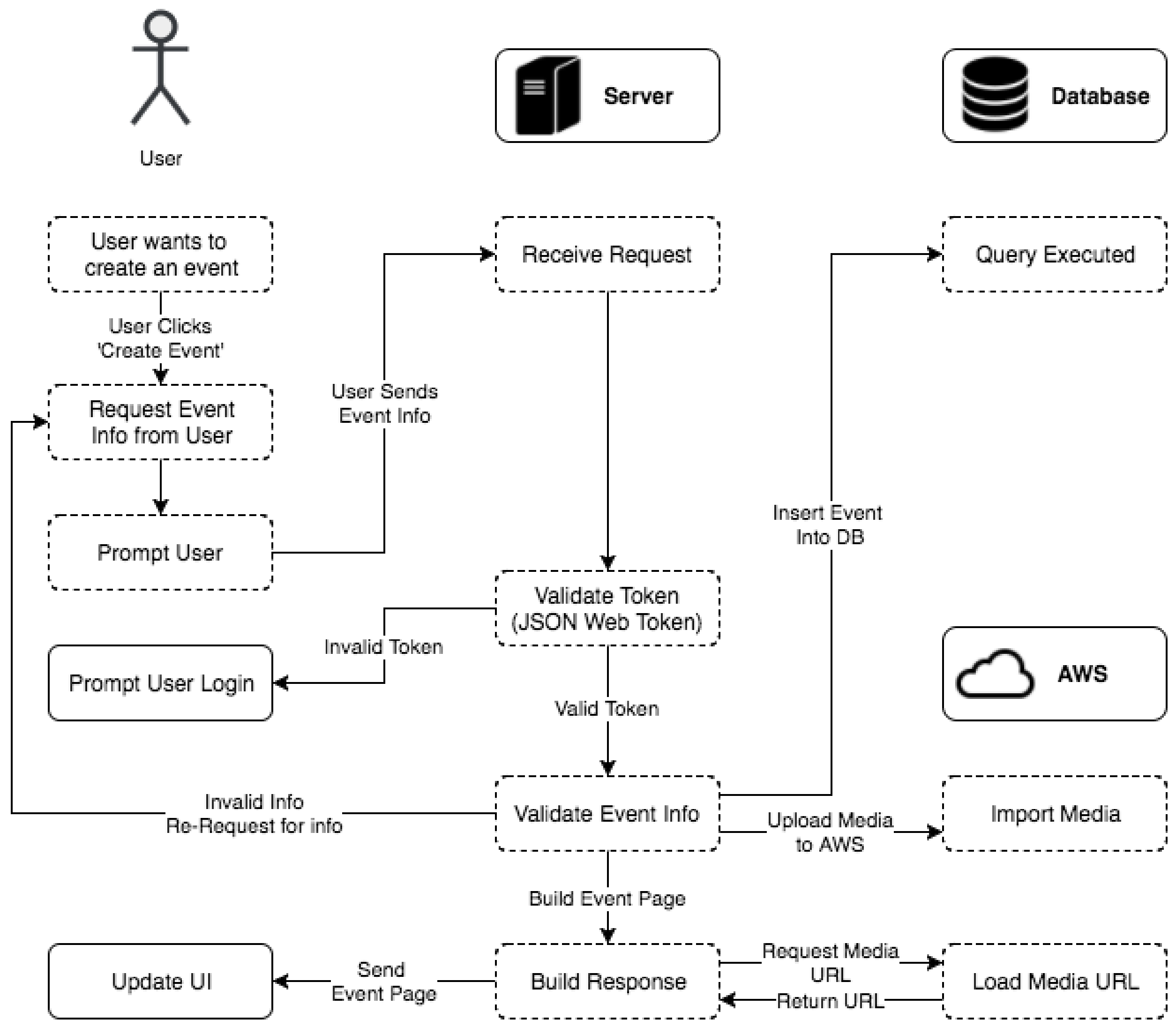


Fig 1. User creates a new event.

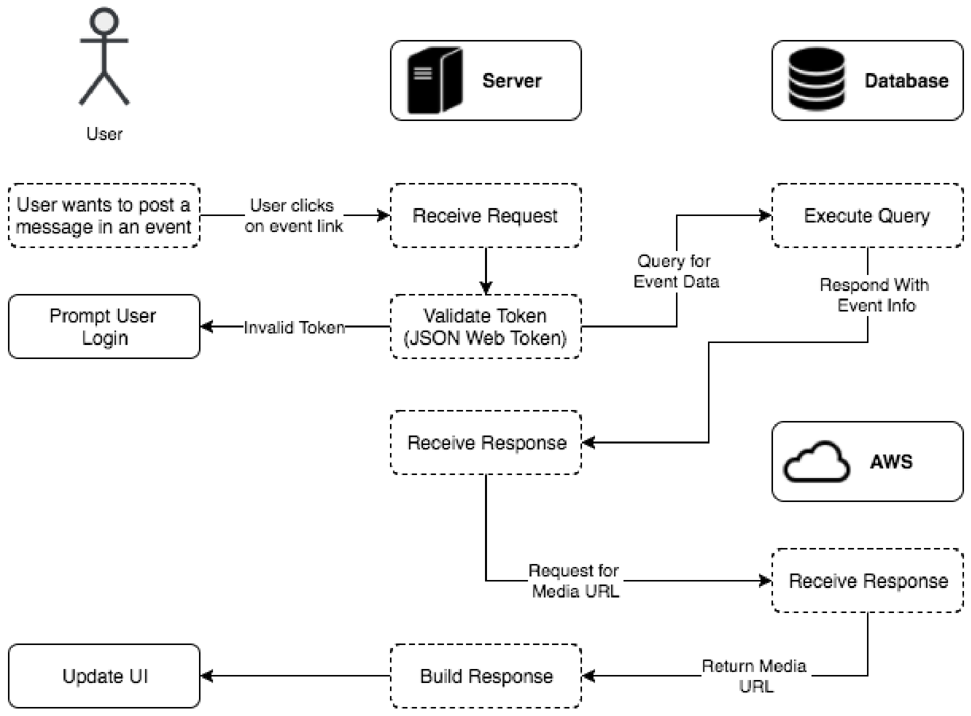


Fig 2. User requests an events page.

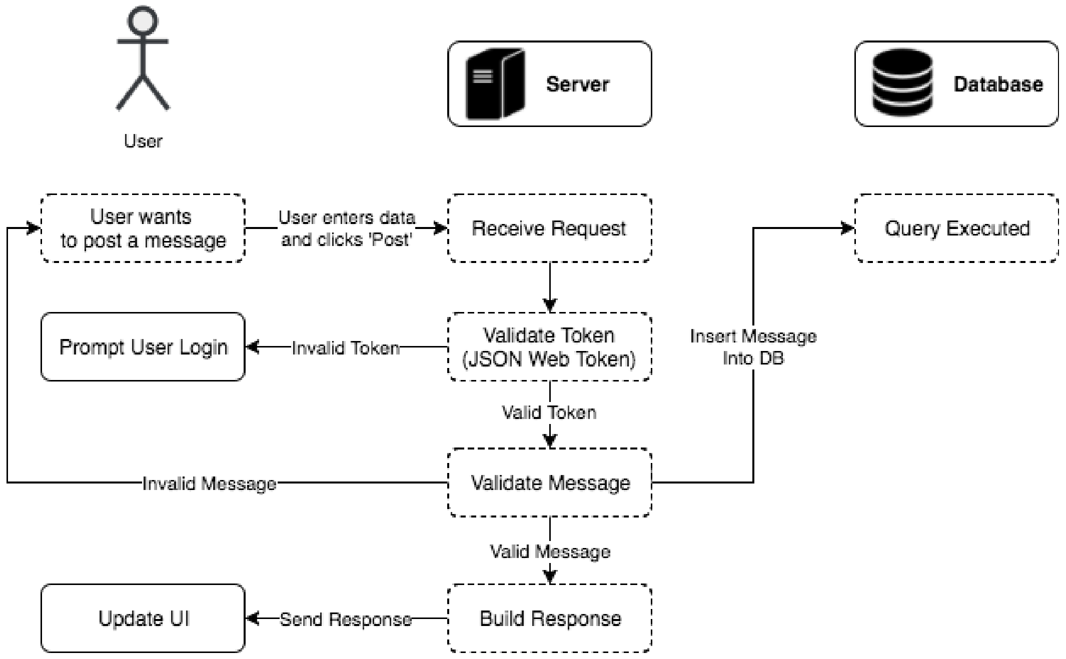
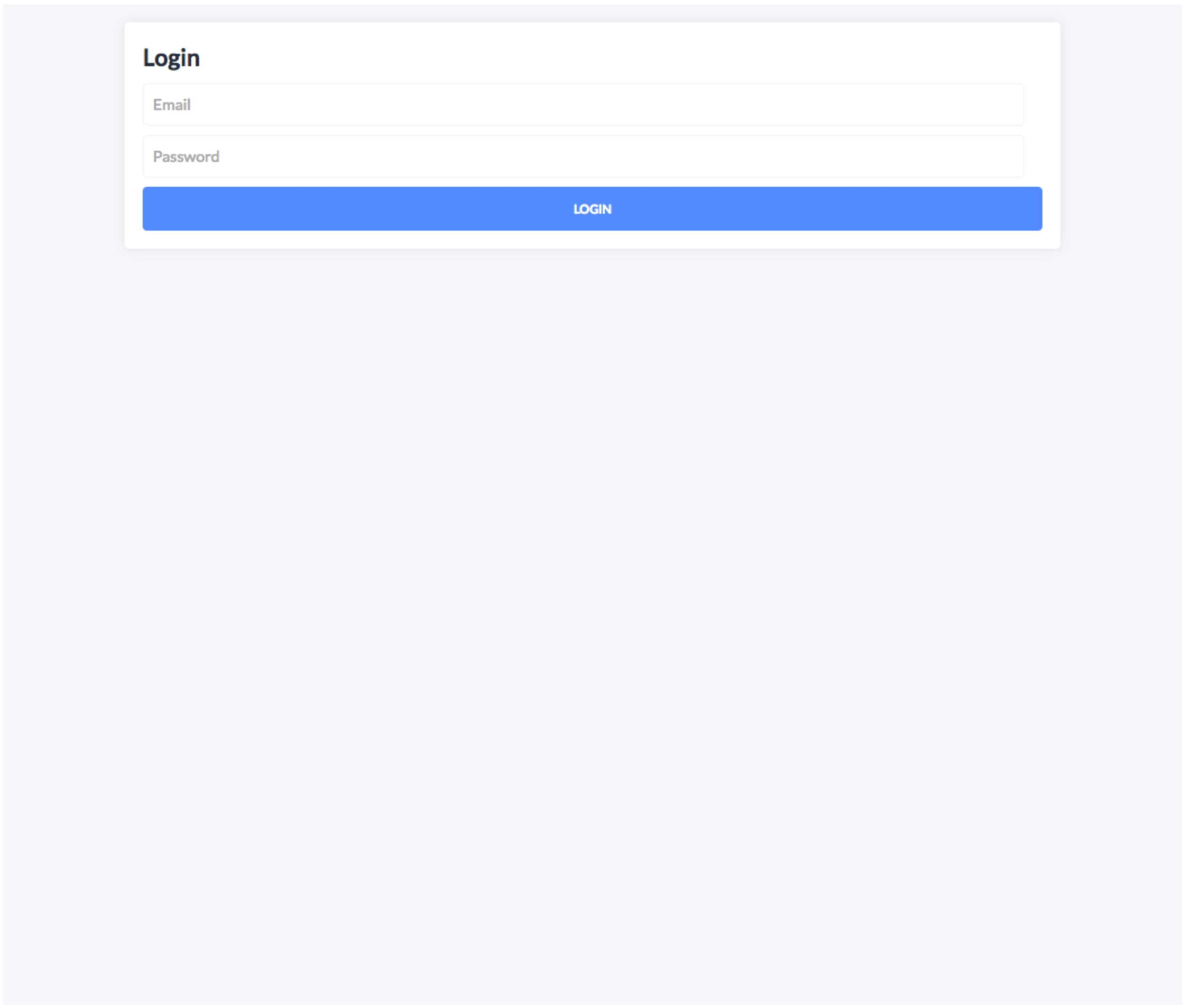


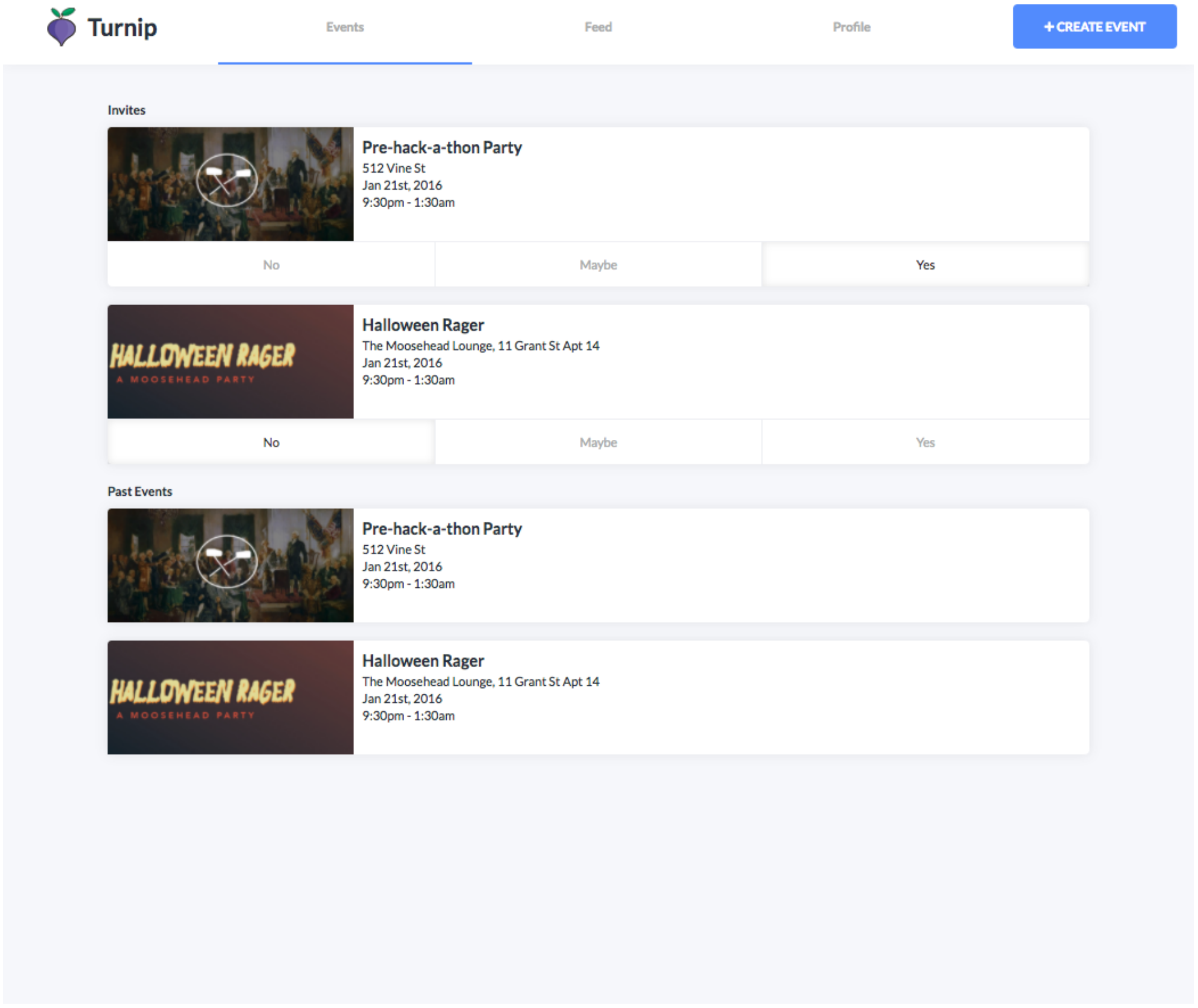
Fig 3. User posts a message on an event page.

Design Mockups - Desktop

User Login Page



Dashboard



Invites

Pre-hack-a-thon Party
512 Vine St
Jan 21st, 2016
9:30pm - 1:30am

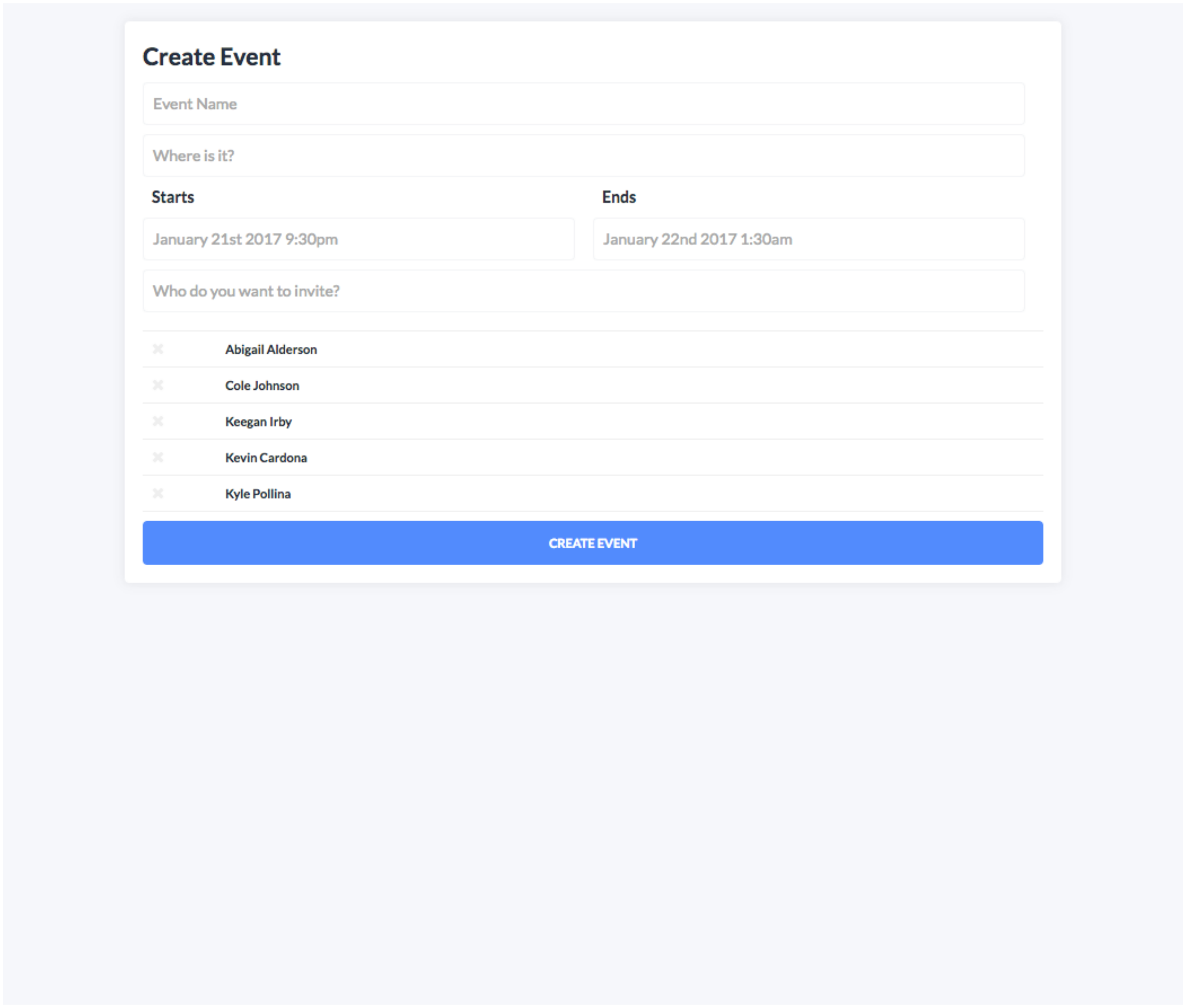
Halloween Rager
The Moosehead Lounge, 11 Grant St Apt 14
Jan 21st, 2016
9:30pm - 1:30am

Past Events

Pre-hack-a-thon Party
512 Vine St
Jan 21st, 2016
9:30pm - 1:30am

Halloween Rager
The Moosehead Lounge, 11 Grant St Apt 14
Jan 21st, 2016
9:30pm - 1:30am

Create Event Page



Create Event

Event Name

Where is it?

Starts	Ends
January 21st 2017 9:30pm	January 22nd 2017 1:30am

Who do you want to invite?

Abigail Alderson

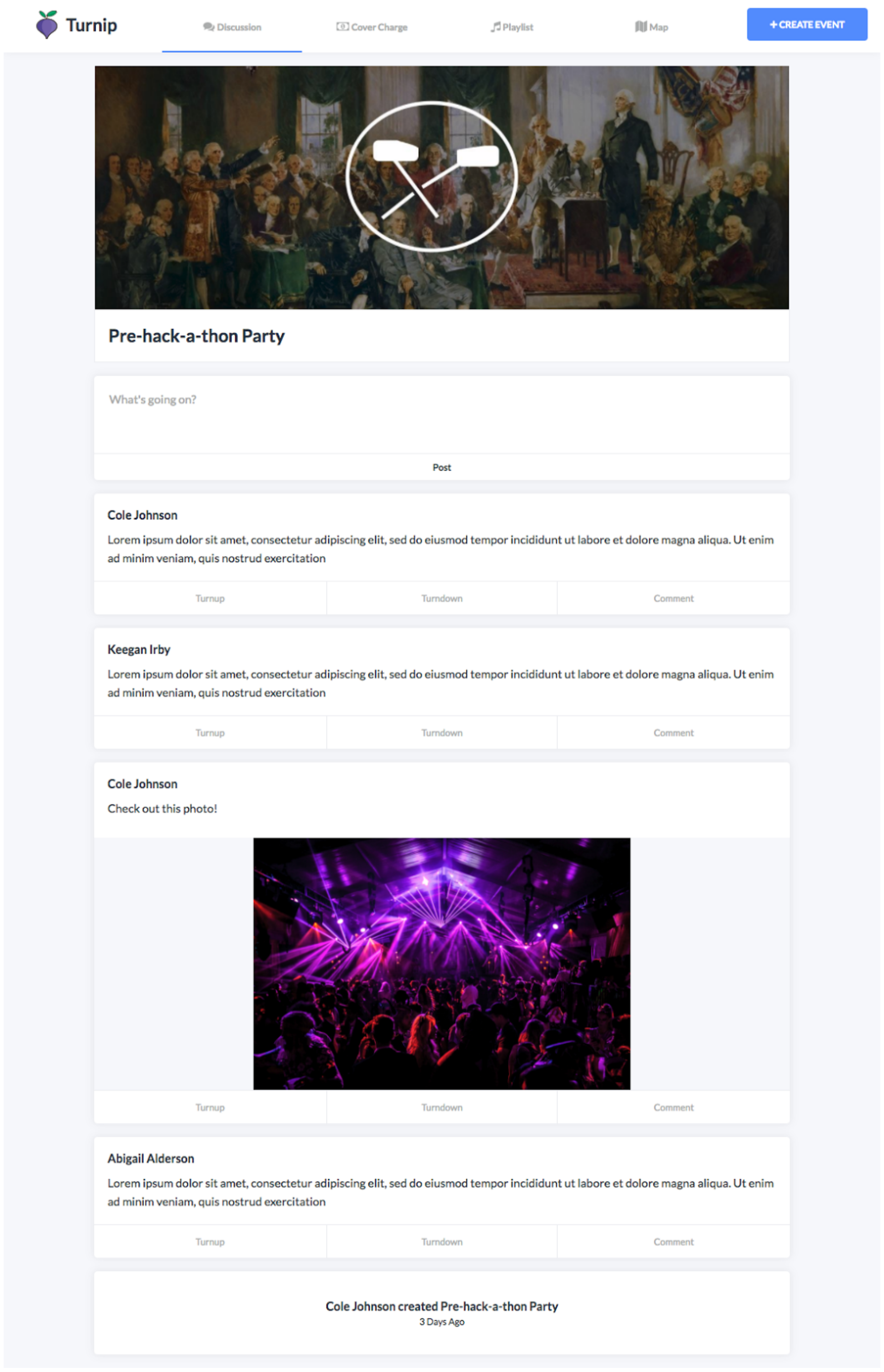
Cole Johnson

Keegan Irby

Kevin Cardona

Kyle Pollina

Event page



Pre-hack-a-thon Party

What's going on?

Post

Cole Johnson

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation

Keegan Irby

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation

Cole Johnson

Check out this photo!

Abigail Alderson

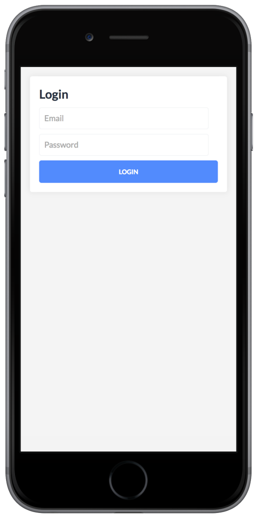
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation

Cole Johnson created Pre-hack-a-thon Party

3 Days Ago

Design Mockups - Mobile

User Login Page



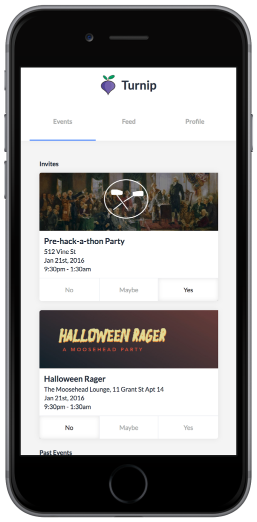
Login

Email

Password

LOGIN

Dashboard



Turnip

Events Feed Profile

Invites

Pre-hack-a-thon Party
512 Vine St
Jan 21st, 2016
9:30pm - 1:30am

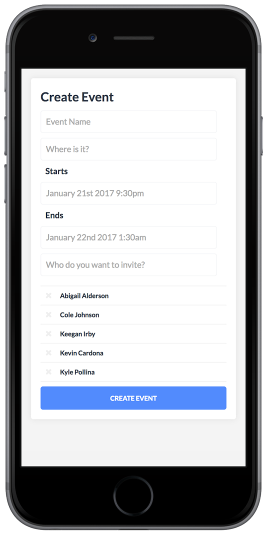
No Maybe Yes

Halloween Rager
The Moosehead Lounge, 11 Grant St Apt 14
Jan 21st, 2016
9:30pm - 1:30am

No Maybe Yes

Past Events

Create Event Page



Create Event

Event Name

Where is it?

Starts

January 21st 2017 9:30pm

Ends

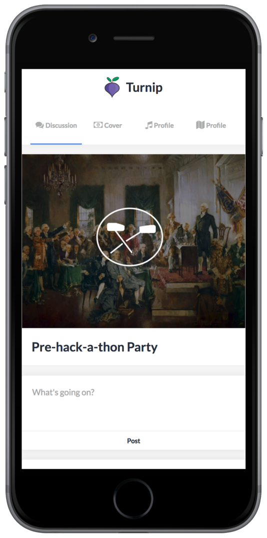
January 22nd 2017 1:30am

Who do you want to invite?

- Abigail Alderson
- Cole Johnson
- Keegan Irby
- Kevin Cardona
- Kyle Pollina

CREATE EVENT

Event Page



Turnip

Discussion Cover Profile Profile

Pre-hack-a-thon Party

What's going on?

Post