# PHRASECARDS

Daniel Choi
Sean Kelley
Robert Hromada
David Su
Mike Turley
Colby Stone
Yue Shing

## Team Roles

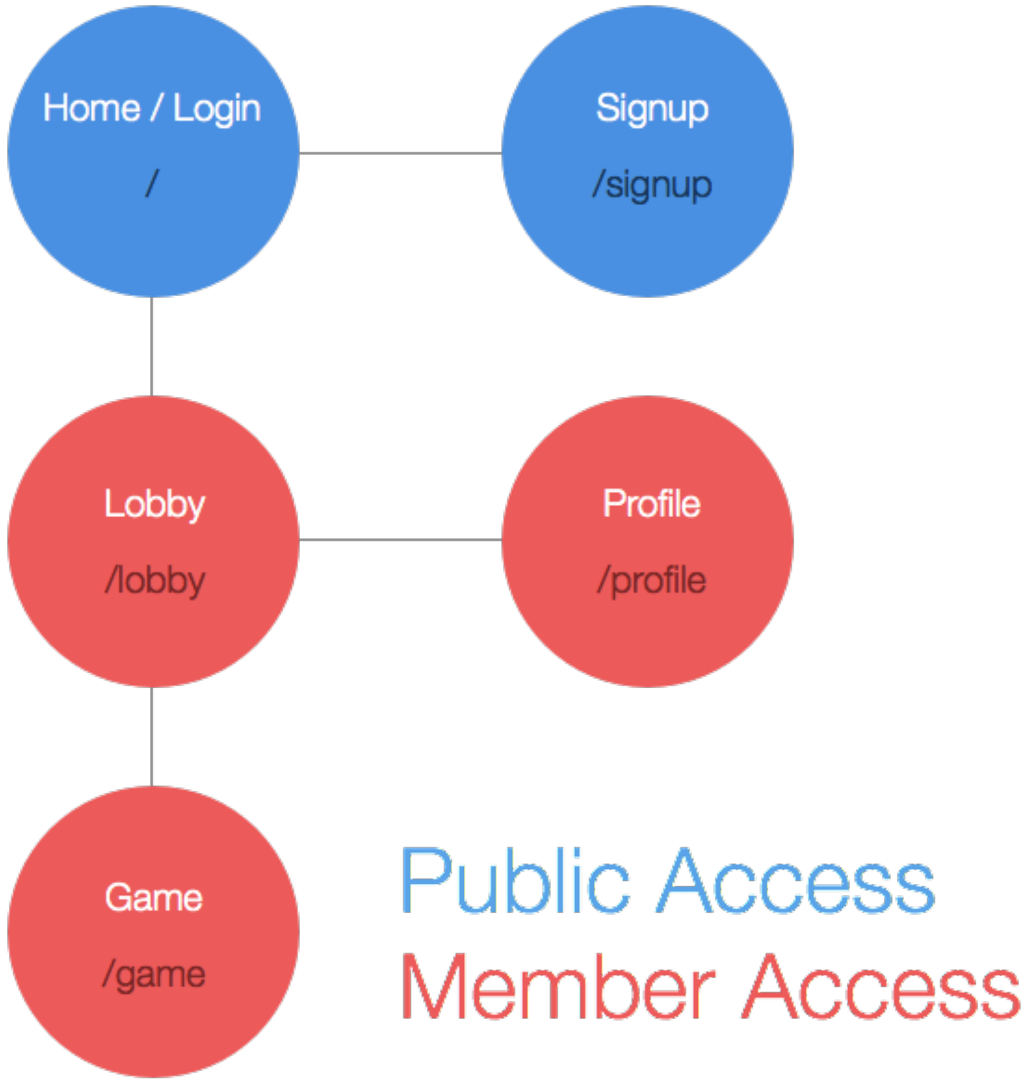
| | |
|---|---|
| Colby Stone | • Project Manager |
| Sean Kelley | • Creative Director |
| David Su | • FrontEnd Programmer |
| Yue Shing | • FrontEnd Programmer |
| Mike Turley | • BackEnd Programmer |
| Daniel Choi | • BackEnd Programmer |
| Robert Hromada | • Documentation Scribe |

# Project Idea

This project is an interactive multi-player on-line competitive madlib web service. The game is centered around filling in an incomplete story. The players submit words to the "card czar” who chooses which word given is used to fill in the story. The card czar is a role which changes every turn, so every player gets a chance to create the story, using their favorite submitted word given by the others.

The application presents a user friendly gui, a playing area with real-time updates, personal profile page with a fiends list, search functionality for other players profiles and an in-game chat room. Our back end development uses MongoDB to store user data, game state and structured story data, and shares this data with the client via AJAX and Socket.IO implementations.

# Birds Eye View



# Components

**Overview**

The application is built in Javascript on top of a Node.js server environment customized with the ExpressJS web framework for Node. The server uses EJS to render static pages and initial page templates, and some of the client-side pages use dynamic reactive template updates powered by Blaze. CSS styles based on Foundation with some customizations are used across all pages of the application for the look and feel. The website contains features from the individual aspect, such as personal profile, to the community aspect (ie. lobby and chat room). The remaining components are focused on the game aspect of the app. Certain components, like stories, are necessary for the game to run properly, while others simply enhance the players' experience.

**Login**

For the player login we used a library called Passport. We set it up to allow users to sign into their account by filling in two fields: Email and Password. We used HTML5 fields to deal with simple validation. Once authentication is completed, the user successfully log in and redirected to the lobby page. If authentication fails, an error message will appear, and user will be asked to re-enter information into the two fields. There is a link to the sign-up page, in case the user trying to log in does not have an account.

**Signup**

Passport also deals with the sign up. The account creation requires an email address, a nickname, a password and a password confirmation. The feature checks the database and makes sure the nickname has not been taken already. Once the account is created the user will be re-directed to the lobby page.

## Components

**Lobby**

The lobby function is is an essential feature of the web app that allows all users to see the games in progress, games being created, and games being filled up. Players not in a game will spend most of their time here. There will be a list of players currently in the lobby, in which a player can click and view their profiles. Moreover, in the lobby, players will be able to chat with other players in the lobby, create a game, and join a game.

**Story**

All Madlib stories will be stored in a JSON object. These objects will contain information about the story like it's name, search tags, and length, as well as the story itself. The stories will be broken into "story chunks", which will provide the pieces of data that the different player views, either submitter or voter, will use inside the game. This object will be updated as the game progresses to store game state, and can be queried for any game information for rendering or other purposes.

**Game**

This is our main game view for when the mad-lib game is running. Users enter this view when they join a game room from the lobby. This page implements Blaze to render the current state of a game in real time based on data coming in from socket.io connections and ajax. The game will support the chat features and contain likes to players profiles through their names or Gravatar. The game views also contain the progressbar.js library for use during the game's timed turns.

# Components

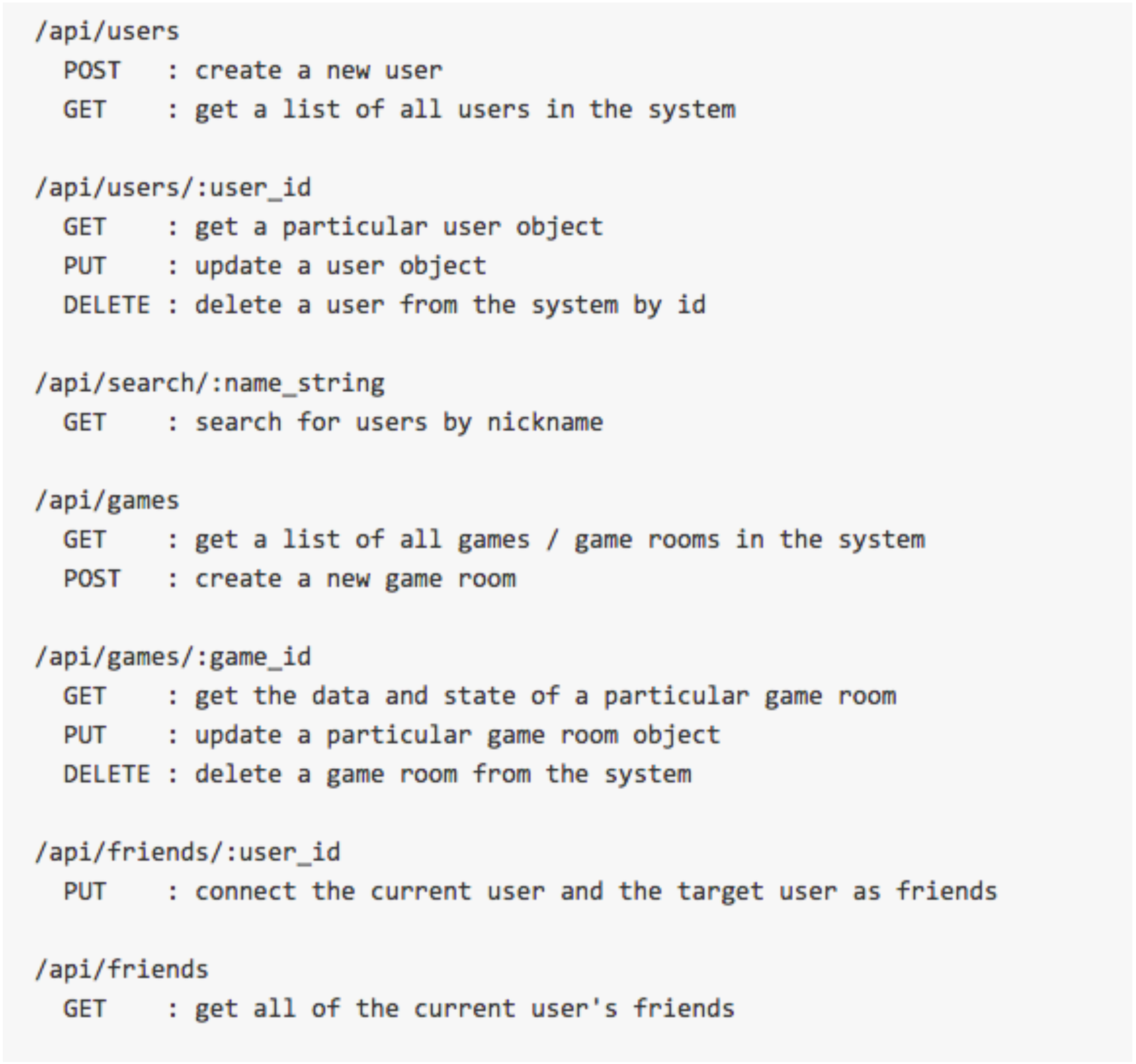
**Chat**
Our chat is built using socket.io. Anyone participating in that particular game will be able to use the chat feature. We also set up a global variable to hold the current users nickname and gravatar and display them when a user is sending a message in chat. This feature is meant to enhance players' game experience by increasing their interaction with each other.

**Profile**
Profile will hold all the information about the account creator. The information shown includes a Gravatar picture, total number of games won, the players friends and a search for friends area. This feature allows players to distinguish themselves from other players and give their personal profile a unique feel. Furthermore, profiles enables players to view any other player's basic information.

**Search**
User search functionality is available within the profile page. This functionality allows searching for friends dynamically amongst the database of PhraseCards users. This feature uses jQuery's AJAX functionality to fetch data from the back-end API.

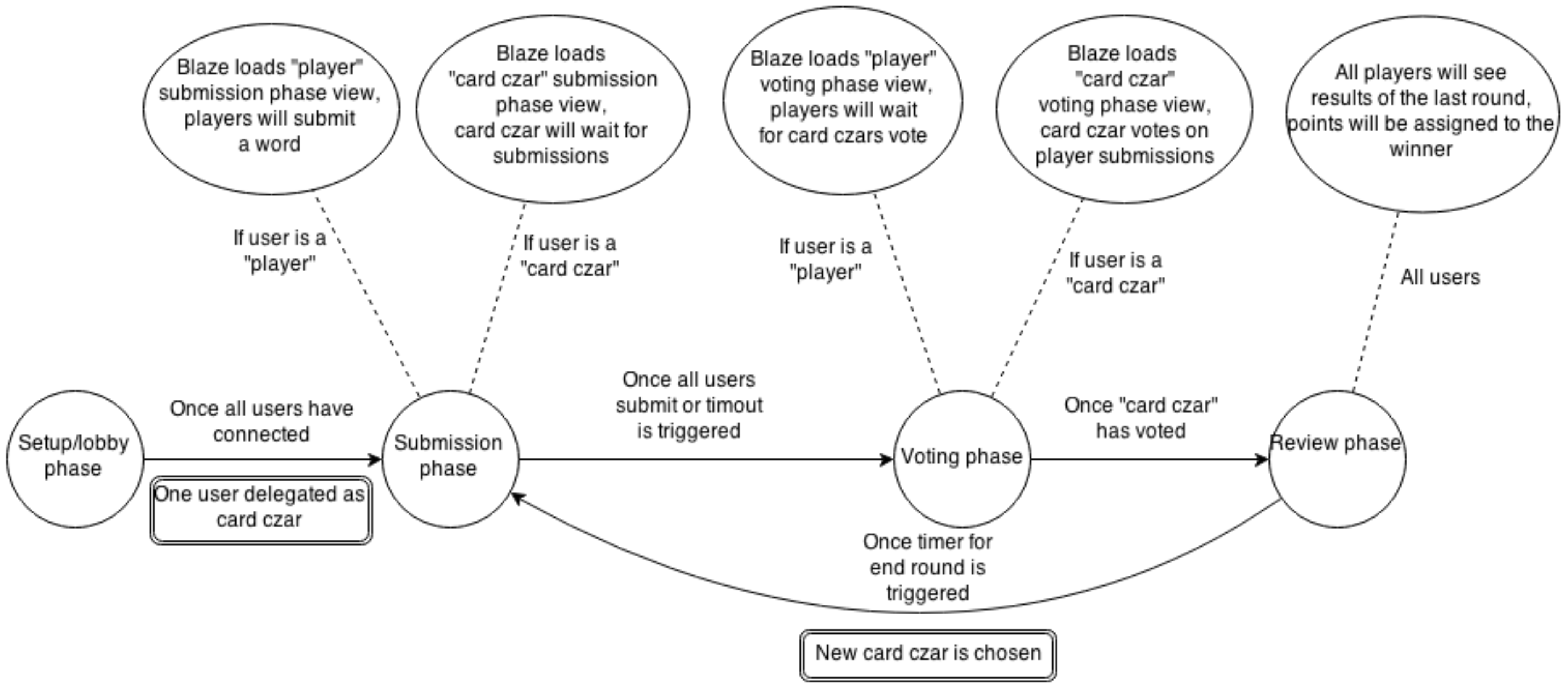
**Friends**
Friends consists of a list of a persons friends. It is part of the profile functionality and is used in conjunction with search.

## API

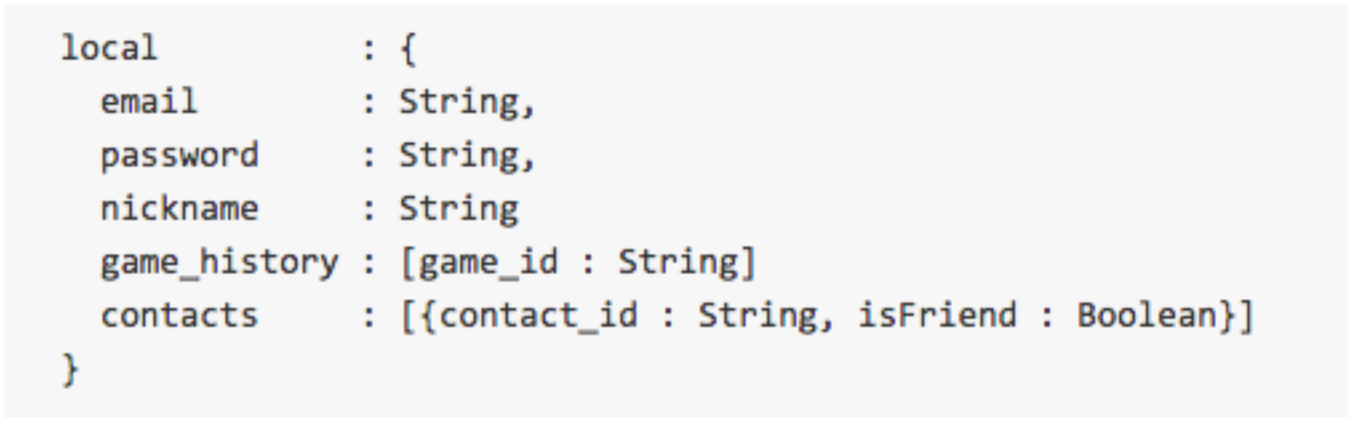


```
/api/users
  POST   : create a new user
  GET    : get a list of all users in the system

/api/users/:user_id
  GET    : get a particular user object
  PUT    : update a user object
  DELETE : delete a user from the system by id

/api/search/:name_string
  GET    : search for users by nickname

/api/games
  GET    : get a list of all games / game rooms in the system
  POST   : create a new game room

/api/games/:game_id
  GET    : get the data and state of a particular game room
  PUT    : update a particular game room object
  DELETE : delete a game room from the system

/api/friends/:user_id
  PUT    : connect the current user and the target user as friends

/api/friends
  GET    : get all of the current user's friends
```

# In-Game Data Flow and State Transitions

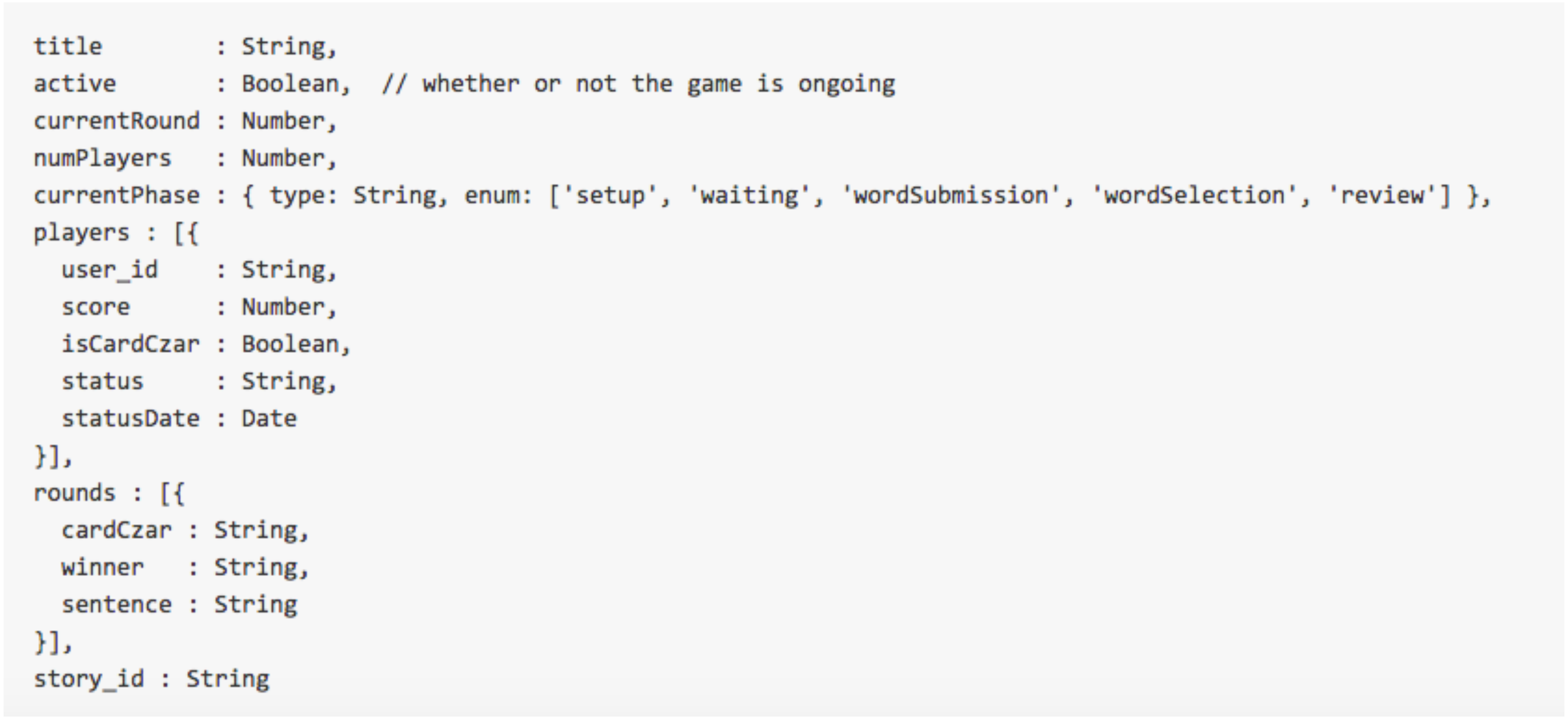

Blaze loads "player" submission phase view, players will submit a word
Blaze loads "card czar" submission phase view, card czar will wait for submissions
Blaze loads "player" voting phase view, players will wait for card czars vote
Blaze loads "card czar" voting phase view, card czar votes on player submissions
All players will see results of the last round, points will be assigned to the winner
If user is a "player"
If user is a "card czar"
If user is a "player"
If user is a "card czar"
All users
Setup/lobby phase
Once all users have connected
One user delegated as card czar
Submission phase
Once all users submit or timout is triggered
Voting phase
Once "card czar" has voted
Review phase
Once timer for end round is triggered
New card czar is chosen

# Database Design User



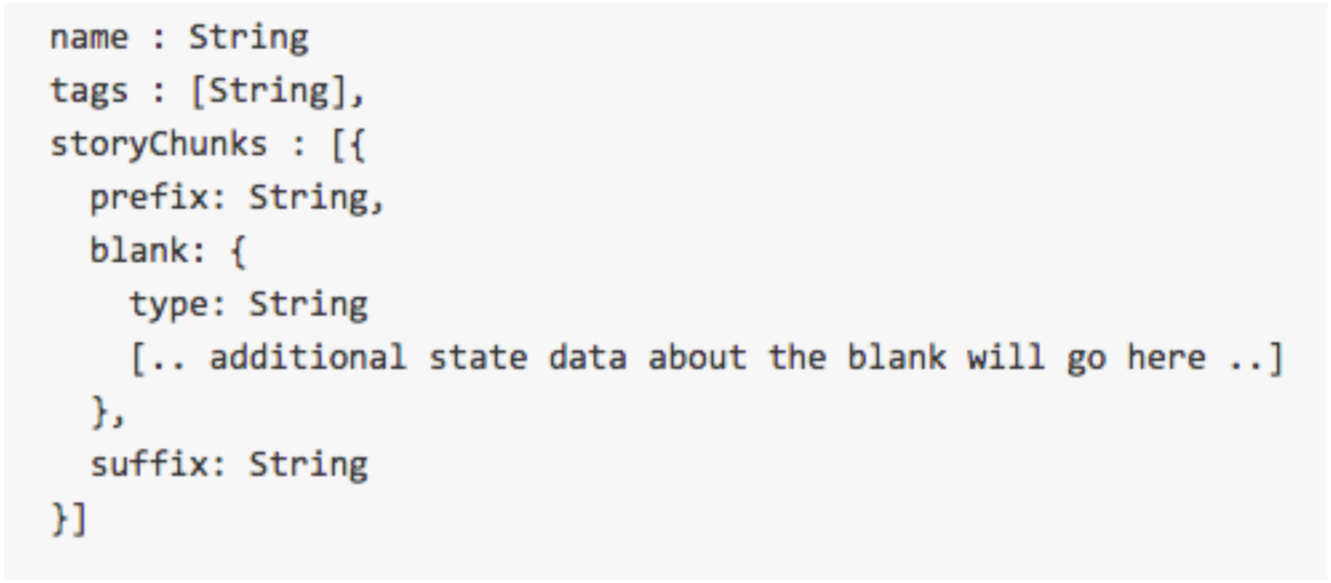
```
local          : {
  email        : String,
  password     : String,
  nickname     : String
  game_history : [game_id : String]
  contacts     : [{contact_id : String, isFriend : Boolean}]
}
```

# Database Design Game



```
title        : String,
active       : Boolean,  // whether or not the game is ongoing
currentRound : Number,
numPlayers   : Number,
currentPhase : { type: String, enum: ['setup', 'waiting', 'wordSubmission', 'wordSelection', 'review'] },
players : [{
  user_id    : String,
  score      : Number,
  isCardCzar : Boolean,
  status     : String,
  statusDate : Date
}],
rounds : [{
  cardCzar : String,
  winner   : String,
  sentence : String
}],
story_id : String
```

# Database Design Story



```
name : String
tags : [String],
storyChunks : [{
  prefix: String,
  blank: {
    type: String
    [.. additional state data about the blank will go here ..]
  },
  suffix: String
}]
```

# External Libraries

**CSS**:

Foundation: bootstrap our css

Font-Awesome: icons

Animate.css: animations

# External Libraries

**Javascripts:**

Jquery: for some basic DOM manipulation and event handling

Blaze: for reactive DOM templating

Foundation: bootstrap our js

progressbar.js: for pretty-looking timers

# External Libraries

**Notable node libraries:**

Express: General backend framework for our web application

Passport: Used for user authentication (registration, login, logout)

Mongodb: Framework for database design

Ejs: Javascript template that cleans html code

Gravatar: Display profile pictures

Socket.io: Used to create Chat and Rooms