# GameMon Documentation

Josephine Chen,
Eric Forsell,
Kevin Jang,
Andy Walz

Hack Reactor HRR21 Towering-Cranes

# Contents

# 1 Introduction

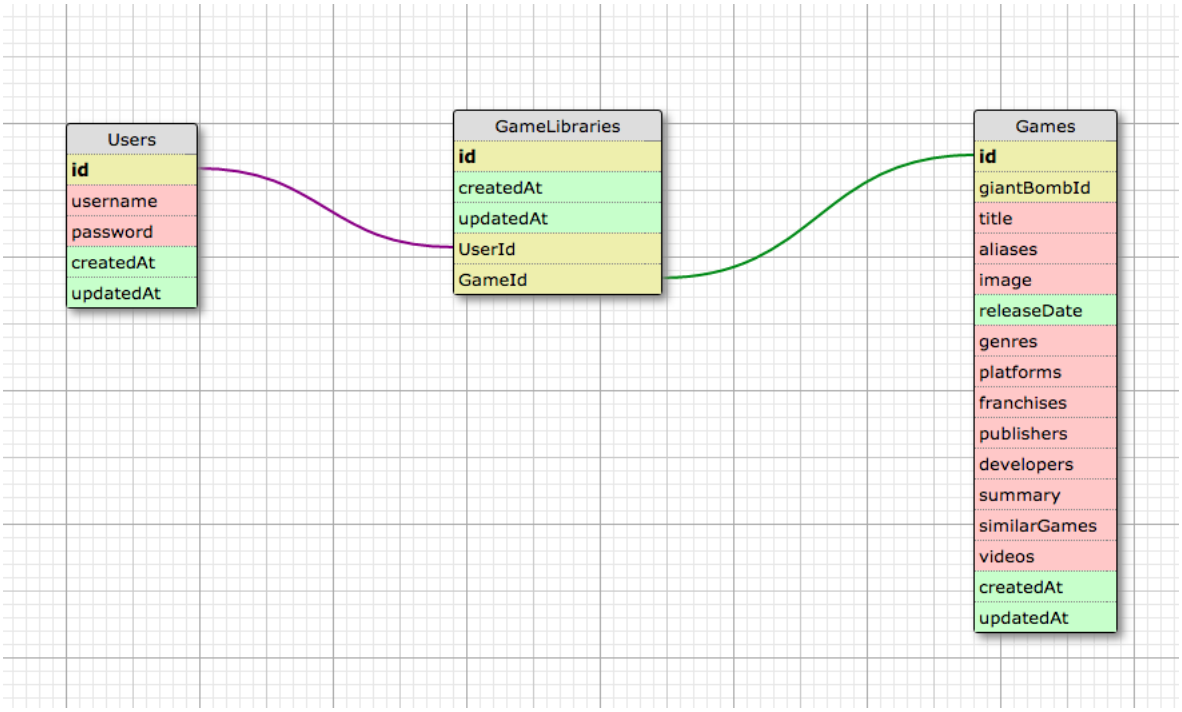
Welcome to GameMon! We hope this documentation helps you navigate around the project quickly and efficiently. GameMon is an app for tracking what video games you have in your collection. Users signup and build out a collection based on the thousands of games available in the GiantBomb API.

# 2 File Structure

```
towering-cranes
├── client
│   ├── assets
│   │   ├── andy.jpg
│   │   ├── eric.jpg
│   │   ├── favicon.ico
│   │   ├── gamemon-collection.png
│   │   ├── gamemon-details.png
│   │   ├── gamemon-search.png
│   │   ├── josephine.jpg
│   │   ├── kevin.jpg
│   │   └── Octocat.png
│   ├── home
│   │   └── home.html
│   ├── main
│   │   ├── gameCollection.js
│   │   ├── main.html
│   │   ├── search.js
│   │   └── selectedGame.js
│   ├── styles
│   │   └── styles.css
│   ├── users
│   │   ├── lock.js
│   │   ├── signin.html
│   │   └── signup.html
│   ├── app.js
│   └── index.html
├── documentation
│   ├── documentation.pdf
│   ├── documentation.tex
│   └── documentation.toc
├── node_modules
└── server
    ├── config
    │   └── keys.js
    └── database
        └── databaseHelpers.js
```

```
        db.js
    giantBomb
        giantBombHelpers.js
    server.js
.gitignore
_.editorconfig
_.gitattributes
_.gitignore
_.jshintrc
_.travis.yml
_CONTRIBUTING.md
_PRESS-RELEASE.md
_STYLE-GUIDE.md
package.json
README.md
```

# 3 Database Schema




# 4 Server-Side

Server-side files hold the routes and the helper functions needed for the routing.

## 4.1 Routes

**Database Routes**

- '/users'
  - POST - receives a username and password from the client and adds each to the database for a specific user
- '/games'
  - POST - receives a user and a game from the client and adds the game to that user's collection in the database
  - DELETE - receives a game title and a user from the request body and removes the game from the users collection
  - GET - receives a username from the client as a parameter in the url and sends all of that users games back to the client

**Giant Bomb API Routes**

- '/games/search/keyword/:keyword'
  - GET - receives a keyword from the client as a parameter and returns up to 10 games that match the keyword
- '/games/search/id/:id'
  - GET - receives a game id from the client as a parameter and returns up to 10 games that match the id

## 4.2 Database Helper Functions

- createUser - receives a user object from the server and either finds a user with that existing name or creates a new user
- addGameToCollection - receives a username and a game object and adds the game to the specified users collection.
- getGamesFromCollection - receives a username from the server and finds/returns all of that users games.
- removeGameFromCollection - receives a username and a game from the server and deletes that game from the specified users collection

## 4.3 Giant Bomb API Helper Functions

Note: ES6 syntax used here in the request calls

- searchForGames - receives a search term and uses expresss request module to send that request to the Giant Bomb api.

  - Example: To search for all Pokemon games our url in the options object would be the following: ‘http://www.giantbomb.com/api/search/?api_key=$YOUR-API-KEY&format=json&query=”$pokemon”&resources=game‘

- getGameById - receives an id from and uses that id to get the game with the corresponding id form Giant Bomb.
  - Example: To search for metroid prime and list its genres and name based on id do the following: ‘http://www.giantbomb.com/api/game/3030-4725/?api_key=$YOUR-API-KEY&format=json&field_list=genres,name‘

# 5 Client-Side

Front-end uses AngularJS with Materialize/Angular-materialize. Materialize/Angular-materialize can be substituted with Bootstrap or any other front-end framework. Angular-materialize is a set of AngularJS directives to use features in Materialize that requires jQuery. It is NOT the same as Angular Material.
**Resources:**

- AngularJS
- Angular-materialize
- Material Icons
- Materialize CSS

## 5.1 Files

The roles of the different files are as follows:

- /client/styles/styles.css
  - Custom stylesheet for overrides and other styles not included in Materialize
- /client/assets
  - Images and other files to be used across client pages
- /client/index.html
  - Loads JS libraries
  - Sets up header, navigation, and footer
- /client/app.js
  - Handles client side routing to load templates in /client/main or /client/home
- /client/home/home.html
  - Template for product landing page

  - Default page if root directory is loaded or if an invalid url is provided

- /client/main/main.html
  - Uses ternary operator on gameCollection to see if search sidebar should be displayed. If it should be displayed, the column width is set to 8, if not, it is set to 11.
  - Uses custom filter in gameCollection.js to allow filtering by title, franchise, platform, and genre
  - Allows collection view to change between list and grid and sort by title or release date
  - Holds modal markup that appears when a game is clicked on

- /client/main/modal.js
  - Controller to load data into the scope for modal information
    * similar games

- /client/main/gameCollection.js
  - Holds a factory that allows
    * getUserCollection that loads a game collection for the signed-in user
    * addGameToCollection adds a selected game to the current users collection
    * removeGameFromCollection removes the selected game from the current users collection
  - Holds a custom filter that takes in setFilter(filterOpt) as an input where filterOpt is an array
    * custom filter checks for a match in the respective locations of all the objects and returns all matches
    * first element of filterOpt is the search term to find
    * second element is what type of filter term it is: text, platform, or genre
    * returns all objects that will be displayed after filtering
    * items holds all possible objects

- /client/main/search.js
  - Holds a factory that allows
    * searchByTerm
    * searchById

## 5.2 User Authentication

We implemented a client-side authentication using Auth0. The toggle.js files holds the logic for the authentication. First, we instantiate a module gameMon.toggle. We include the dependency 'auth0' in order to use the Angular wrapper for Auth0. We pass the domain and client id in as an argument to

```
authProvider.init({
  domain: <our domain key>,
  clientID: <our client ID>
  });
```

After this we create a controller, LoginController, which runs when a user clicks signup and when a user logs out. The important takeaway from this controller is its use of local storage. When the user logs in, their profile information and Auth0 access token is added to the browsers local storage. We set and remove the profile and token by calling localStorage.setItem and localStorage.removeItem.

# 6 Deployment

Towering-Cranes used DigitalOcean for deployment. The droplet uses a MEAN base image (0.5.0) running on Ubuntu 16.04 with MySQL and several other technologies.

**It is best to have ONE team member work on deployment. This removes the need for multiple SSH keys and rule out issues with individual development environments. Steps to get it running (this will help ensure there are no issues):**

1. Create Droplet with 1-Click App: MEAN 0.5.0 on 16.04
   - Smallest droplet size is fine
   - Datacenter is whatever you want – closest to your users is best
   - **Add SSH keys here and now!** Adding them later is a headache
2. After the droplet is created, UPDATE your droplet
   (a) Log in using root user: **ssh://root@YOUR_DROPLET_IP**
      - If you set up ssh keys correctly, the server shouldn't ask for your root password
      - If you run into problems, use the droplet console via DigitalOcean's dashboard to log in as root and troubleshoot. On the console page, there is also a link to reset root password if necessary

   (b) Run the apt-get commands to update
      i. **sudo apt-get upgrade**
      ii. **sudo apt-get update**
3. Add swapfile (reference)
   (a) This is **necessary** because MySQL needs more RAM than the server has been allocated. Npm install might also fail if the swapfile isn't created

   (b) Referencing the link above, run the following commands
      i. **sudo fallocate -l 4G /swapfile**
      ii. **sudo chmod 600 /swapfile**
      iii. **sudo mkswap /swapfile**
      iv. **sudo swapon /swapfile**
      v. **sudo nano /etc/fstab**

vi. **sudo sysctl vm.swappiness=10** (persists until system reboot)

vii. **sudo sysctl vm.vfs_cache_pressure=50** (persists until system reboot)

4. Install MySQL (reference)

   (a) Run the following commands referencing the above link as necessary

      i. **sudo apt-get install mysql-server** This will prompt you to create a root password. Write it down!

      ii. **sudo mysql_secure_installation**

   (b) MySQL should be up and running

   (c) Log into the MySQL CLI

      i. **mysql -u root -p** (enter password at prompt)

   (d) Create a database named gamemon

      i. **CREATE DATABASE gamemon**

5. Get a Giant Bomb API key (write this down)

   - Giant Bomb API

6. Clone repo (with work tree and git tracking separation) (reference)

   (a) Navigate to a directory you want to store both the repo and a separate git folder (towering-cranes used root/)

   (b) Install PM2 for running the web app server in the background

      i. **npm install -g pm2**

      ii. Create a config file for PM2 with env variables

         A. **touch gamemon.config.js**

         B. **nano gamemon.config.js**

         C. Copy and paste this replacing the db password and Giant Bomb API key:

```
module.exports = {
  /**
   * Application configuration section
   * http://pm2.keymetrics.io/docs/usage/application-
       declaration/
   */
  apps : [
    {
      name      : "server",
      script    : "node towering-cranes/server/server.js",
      env: {
        PORT: 80,
        DB_PASSWORD: password_here******,
        GIANTBOMB_API_KEY: api_key_here******
      }
    }
  ]
```

         D. Save file

(c) Clone the web app repo
   i. **git clone REPO_LINK_HERE**
   ii. cd into the repo folder
   iii. **npm install**
   iv. Run the server by cd to where gamemon.config.js is (probably up one directory)
   v. **pm2 start gamemon.config.js**

(d) Create bare git
   i. **cd ..**
   ii. **mkdir SOME_NAME_HERE.git**
   iii. **cd SOME_NAME_HERE.git**
   iv. **git init –bare**
   v. **cd hooks**
   vi. **nano post-receive**
   vii. Add the following lines to the post-receive file (press Ctrl+X to quit and save)
      A. #!/bin/sh
      B. git –work-tree=PATH_TO_REPO_DIRECTORY –git-dir=PATH_TO_BARE_GIT checkout -f
      C. pm2 stop server
      D. sudo fuser -k 80/tcp
      E. pm2 restart server
   viii. chmod +x post-receive

(e) Set up local workspace (best for only one team member to do this)
   i. cd to your local clone of the repo on the server
   ii. Add a remote
      - **git remote add live ssh://root@your_droplet_ip/path_to_bare_git**
      - When deploying, **git push live master** should push all the recent changes and restart the PM2 server instance

(f) Sometimes the post-receive hook doesn't instantiate the env vars. A workaround when parts aren't working, e.g. search bar, is to ssh into the droplet and run these commands
   i. **sudo fuser -k 80/tcp**
   ii. **pm2 restart server**

# 7 Documentation

If you haven't noticed already, this document is written using LaTeX, a typesetting system that is great for technical documents. **Steps to start using LaTeX:**

1. Download one of the TeX distributions
   - For Mac users, go to MacTeX.
   - For Windows users, check out MikTeX or TeXLive.

- For Linux users, go to TeXLive

2. Download a viewer with inverse search
    - For Mac users, download Skim PDF Viewer.
    - For Windows users, download Sumatra PDF.
3. Install one of the LaTeX packages in Sublime (LaTeXTools or LaTeXing)
4. Build your .tex document and start using LaTeX!