UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Puzzle Generator Requirements Document

Team Prometheus

01 July 2020

Clients: Mark Anthony and Francios

Name	Student Number	**Email**
Yuval Langa	u18174142	u18174142@tuks.co.za
Tsholofelo Gomba	u17391718	u17391718@tuks.co.za
Tapiwa Mazibuko	u18203541	u18203541@tuks.co.za
Charlotte Jacobs	u15165622	u15165622@tuks.co.za
Jaynill Gopal	u15175295	u15175295@tuks.co.za

1 Architectural structural design requirements

1.1 AI Puzzle Creation

1.2 Front End Implementation

We are making use of the MVC architectural pattern to implement this portion of the system, structured as follows:

- Model - we are making use of a Postgres database to store data such as user and puzzle details
- View - we are using the Angular framework to create components representing individual pages to be rendered in the browser.

- Controller - we are making use of an API written using NodeJS and the Sequelize ORM to query the database.

The user interacts with the website in the browser (view using Angular). The request generated from the user interaction is send to the API (controller using NodeJS) which will manipulate the data in some way and render the results to the user in the view once more. Some requests might need database access (model using Postgres). Appropriate data is then retrieved or saved as needed.

1.3 Technologies

- Hosting
  Heroku's free tier is used to host the website, allowing the user to access and create puzzles.

- Server
  NodeJs using Express framework is used to host the API on the server side, which is used to allow interaction between the user and database as well as hosting the automatic puzzle creation.

- Web page
  The web page will allow users to access and to have puzzles generated, which they can then download and save as a 3D printable file.

- Database
  PosgreSQL will be used to store data required for the users, there are three main tables: Users, Puzzles and PuzzleRating

- Frameworks
  - AngularJS will be used so that the web page will have a responsive design without needing to load a new page.
  - Sequilize will be used to allow for querying the PostgreSQL database.

2 Quality requirements

2.1 Usability

Users with a basic technological skill level must be able to freely create puzzles and use the website without any difficulty.

- This requirement is addressed by the layout and design of the user interface.

- The front end of the website is easy to navigate and visually pleasing with clear buttons relating to each task for a better user experience.

- The front end uses an angular framework to serve web pages as well as form error detection. Angular increases the user experience by the Dialog component which allows draggable, resizable, closable, and responsive features.

2.2 Performance

The AI should be able to generate a new puzzle between 30-60 seconds. Due to the real time nature of the website.

- This requirement is very important and will be addressed by a combination of architectural design patterns.
- The performance of the of the manual puzzle generation is less computationally taxing and will be a part of the MVC design pattern.
- The AI puzzle generation will be efficient because of the genetic algorithm using the master slave design pattern which will allow the same algorithm to run efficiently with different inputs by dividing work between the master and slave components.

2.3 Reliability

The website should be able to handle traffic from hundreds of users at any given time without crashing or errors.

- This requirement is very important and will be addressed by both the web server and back-end architectural design.
- The website uses Heroku web server which is a highly reliable website hosting service.
- The MVC pattern is being used with an angular framework and node js to control the requests made to the website and database. Node js allows capturing of exceptions to avoid the server crashings.

2.4 Availability

The website should be up 99.9% of the time.

- This requirement will be addressed by both the web server and database.
- The website uses Heroku web server which is a highly reliable website hosting service.
- The website uses postgre which is also available at all times. Postgre has tools such as PostgreSQL Automatic Failover which maintain high availability.

2.5 Cost

The software and libraries used must be free/open source.

- This requirement will be addressed by using open source libraries and software.
- The website uses the free tier of the Heroku web server.
- The website uses open source architectural technologies such as angular and node express. Which are free for commercial use.

2.6 Security

The system requires security measures to protect user data.

- This requirement is necessary because the website will store sensitive user information.
- The MVC architecture allows for separation of API request to allow more security.
- The website will also protect user information by using password encryption and database management tools such as sequelize. Sequelize protects against vulnuabilites and sql injections.

3 Coding standards document

3.1 Server component

NodeJs Express applications has it's own folder structure. The folder structure of the server is as follows:

- Front-end files are listed under a directory called **Public**. The structure of the directory is as follows:
  - /stylesheets
  - /javascript
  - /images
  - index.html
- Back-end related JavaScript files are listed under the directory **Router**, after which the routing of the API call are in separate files:
  - api.js
  - user.js
  - puzzle.js

Unit testing of back-end the API is tested through Mocha/Chai frameworks. The testing file is listed as test.js in the root folder of the application base.

3.2 Angular component

Angular by design has its own folder structure which we maintained during the creation of our front-end.

- Every website page is a component (e.g. landing page, login page etc) are stored within the pages folder. Please follow the link to see the structure on our Github page
- The shared navbar is in the root app folder for easy access to components needing it
- API calls are defined in a service stored under the services folder
- Models are defined and stored under the models folder
- All images can be found in the assets folder
- All components have a .css, .ts and .html file. Component specific code is defined in the relevant files while global styles are defined in the app css, ts and html files.

3.3 Node API component

We made use of NodeJS to write our API. Calls to the database are made using the Sequelize ORM. Our file structure is as follows:

- Please follow this link to visit our Github page to see the structure
- config folder - it contains the database configurations
- routes folder - it contains files which define the routes for particular endpoints. A typical endpoint route is "api/users/createUser", and the folder therefore contains a users.js file contain routes concerned with the user which in case would be createUser
- models folder - this folder allows the definition of models as defined in the database as per Sequelize standards. An example would be the User class defining the fields which are found in the users database table. A model is referenced when doing database operations

4 User manual

4.1 Server setup

4.1.1 Prerequisites

Ensure that you have the following installed on your system.

1. NodeJS version 3.0+

2. npm
3. PostgreSQL

4.1.2 Installation on localhost

In order to use the server and load files to be hosted on the website the following needs to be done.

1. Open terminal to the directory that the
2. Enter the following command to install dependencies: **npm install**
3. Once the dependencies have been downloaded, you are now ready to run the server.
4. To run the server enter the following command into your terminal in the root folder of your app: **node start**.
5. You are now ready to access your server at **localhost:3200** in your browser.

4.1.3 Customizing your server

1. Changing server's default port
   If you wish to alter the default server port on localhost, you may do so by editing the file:**/bin/www** and locating the variable `PORT`

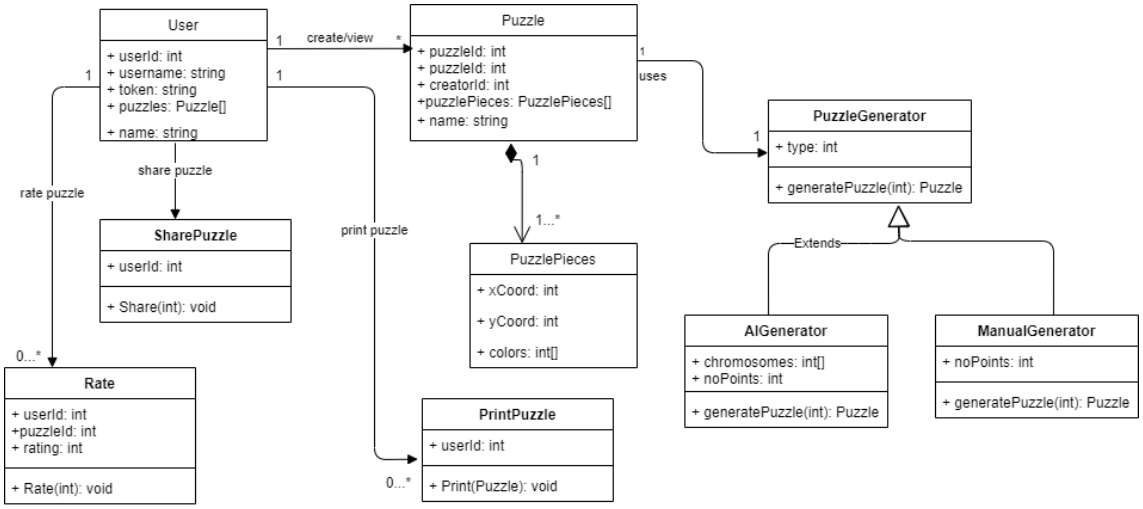
4.1.4 Adding your web pages to be sent

1. All additional web pages and subsequent CSS and JavaScript files intended for the use of the user should be placed in /public /public/stylessheets /public/js
2. to access the API the following routes are used:
   - /api/user/login
   - /api/user/createUser
   - /api/user/resetPassword
   - /api/puzzle/create
   - /api/puzzle/rate
   - /api/puzzle/view
   - /api/puzzle/viewRatings

5 Corrections of SRS document from demo 1

Please make the corrections on the relevant document not here

5.0.1


User
+ userId: int
+ username: string
+ token: string
+ puzzles: Puzzle[]
+ name: string
create/view
1
*
Puzzle
+ puzzleId: int
+ puzzleId: int
+ creatorId: int
+puzzlePieces: PuzzlePieces[]
+ name: string
uses
PuzzleGenerator
+ type: int
+ generatePuzzle(int): Puzzle
share puzzle
rate puzzle
print puzzle
SharePuzzle
+ userId: int
+ Share(int): void
PuzzlePieces
+ xCoord: int
+ yCoord: int
+ colors: int[]
1...*
Extends
AIGenerator
+ chromosomes: int[]
+ noPoints: int
+ generatePuzzle(int): Puzzle
ManualGenerator
+ noPoints: int
+ generatePuzzle(int): Puzzle
0...*
Rate
+ userId: int
+puzzleId: int
+ rating: int
+ Rate(int): void
PrintPuzzle
+ userId: int
+ Print(Puzzle): void