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

Puzzle Generator System Requirements Document

Team Prometheus

11 August 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

Contents

1 Introduction **4**

2 User Characteristics **5**

3 Functional Requirements **5**
3.1 Use Cases 5
3.2 User Stories 6
3.2.1 Use Case Diagram Description 7
3.3 Requirements 7
3.4 Subsystems 9
3.5 Trace-ability Matrix 9

4 Domain Model **10**
4.1 Description of classes 10
4.1.1 User 10
4.1.2 Puzzle 10
4.1.3 SharingPuzzle 10
4.1.4 Rate 11
4.1.5 PrintPuzzle 11
4.1.6 AI Generator and Manual Generator 11

5 Architectural structural design requirements **11**
5.1 Master Slave Architecture 12
5.2 MVC Architecture 12

6 Quality requirements **12**
6.1 Usability 12
6.2 Performance 13
6.3 Reliability 13
6.4 Scalability 14
6.5 Security 14
6.6 Cost 15

7 Technology requirements and Reasons **15**
7.1 Hosting 15
7.2 API Server 15
7.3 Website 15
7.4 Database 16
7.5 Puzzle Generation 16
7.6 Unit Testing 16
7.7 Integration Testing 16

8 Coding standards document **17**
8.1 Server component 17
8.2 Angular component 18
8.3 Node API component 18

9 Technical Installation manual **18**

10 User manual **18**

11 Deployment Model **19**

1 Introduction

1.1 Product Name

Prometheus Puzzles

1.2 Purpose of the Software System:

The Puzzles Generator website aims to efficiently provide an interface for users to generate puzzles through the use of AI or create them manually. The AI based system will make use of a genetic algorithm.

1.3 Scope:

This system will aim to allows for the user to create a puzzle based on a pattern/style of choice as well as with different number of pieces per puzzle.

The system aims to allow users to share creations on the website and rate creations made by other users. A user should finally be able to use the system to generate a printable 3D model of a puzzle.

1.4 Business need:

The business need of this system is to automate the process of creating puzzles. The website design must be user friendly and appealing in order to attract users to interact with the system. Added to this, the steps taken to create the puzzle must be easy to follow and the process efficient.

1.5 Overview:

Puzzles have been a hobby of many people over the centuries, coming in various forms which have various ways of solving. Puzzles have played a role in people's problem solving skills. Puzzle generators have allowed for more puzzles to be created, using various techniques to create interesting and challenging puzzles.

The system involves the creation of 3-Dimensional puzzles (manually and from the use of AI), testing of puzzles, sharing and rating of puzzles by other users, as well as the ability of downloading 3D printable files.

2 User Characteristics

The general user of the website is someone proficient in English so they can easily navigate the website and perform tasks as instructed. The user also needs to have an understanding of how to use technology, even though we aim to make our application very user friendly.
As such, we classified our users according to the following categories:

- Puzzle Enthusiasts - these can be users of any age, be it children, teenagers or adults who like playing with puzzles.
- Parents - some parents may want to make puzzles for their children to play with.
- Educational Users - these can be teachers who want to teach children with challenges such as autism or limited motor skills.

3 Functional Requirements

3.1 Use Cases

- UC1: Register
- UC2: Login
- UC3: Logout
- UC4: View Own Puzzles
- UC5: View Rated Puzzles
- UC6: Update Name
- UC7: Update Username
- UC8: Reset Password
- UC9: View Puzzles
- UC10: Rate Puzzles
- UC11: Update Puzzle Ratings
- UC12: Share Created Puzzles
- UC13: Search Puzzle
- UC14: Export Printable 3D File
- UC15: Manually Create Puzzle
- UC16: Manhattan Puzzle Creation

- UC17: Euclidean Puzzle Creation
- UC18: Generate Puzzle Using AI
- UC19: Select Puzzle
- UC20: Test Puzzle
- UC21: Delete Rating
- UC22: Delete Puzzle
- UC23: Sort Puzzles

3.2 User Stories

1 As a User I want to create an account so that I can create my own puzzles.

2 As a User I want to have a manual generation option so that I can create my own puzzle designs.

3 As a User I want to have a AI generation option so that I can tell it to create a puzzle based on certain options.

4 As a User I want to share my puzzles creations so that I can other users can solve and rate them.

5 As a User I want to stop sharing my puzzles creations so that other users will not be able to print my creations.

6 As a User I want to rate created puzzles so that other users can be encouraged to work on creating more puzzles.

7 As a User I want to update the rating on puzzles so that other users can see how I feel about their puzzles.

8 As a User I want to view all shared puzzles so that I can see what other users are sharing.

9 As a User I want my own profile page so that I can update my personal information.

10 As a User I want a tutorial so I know how to generate puzzles on the creation pages

11 As a User I want to have a delete puzzles option so that I can delete my puzzle creations that I do not like anymore

12 As a User I want to have a delete ratings option so that I can delete my puzzle ratings at will

13 As a User I want to have a search functionality so that I can find puzzles easily

14 As a User I want to have sorting functionality so that I can sort puzzles using different criteria

3.2.1 Use Case Diagram Description

Profile Subsystem
A user can use the register use case to create a new account, login as a registered user, and logout of his/her account. Once the user has a profile, the user can view a list of the puzzles that s/he has created and rated.

User Interaction Subsystem
This subsystem aims to highlight how the user interacts with created puzzles on the website. The user can view created puzzles, rate these puzzles, share their own puzzle creations and export puzzles to printable 3D files.

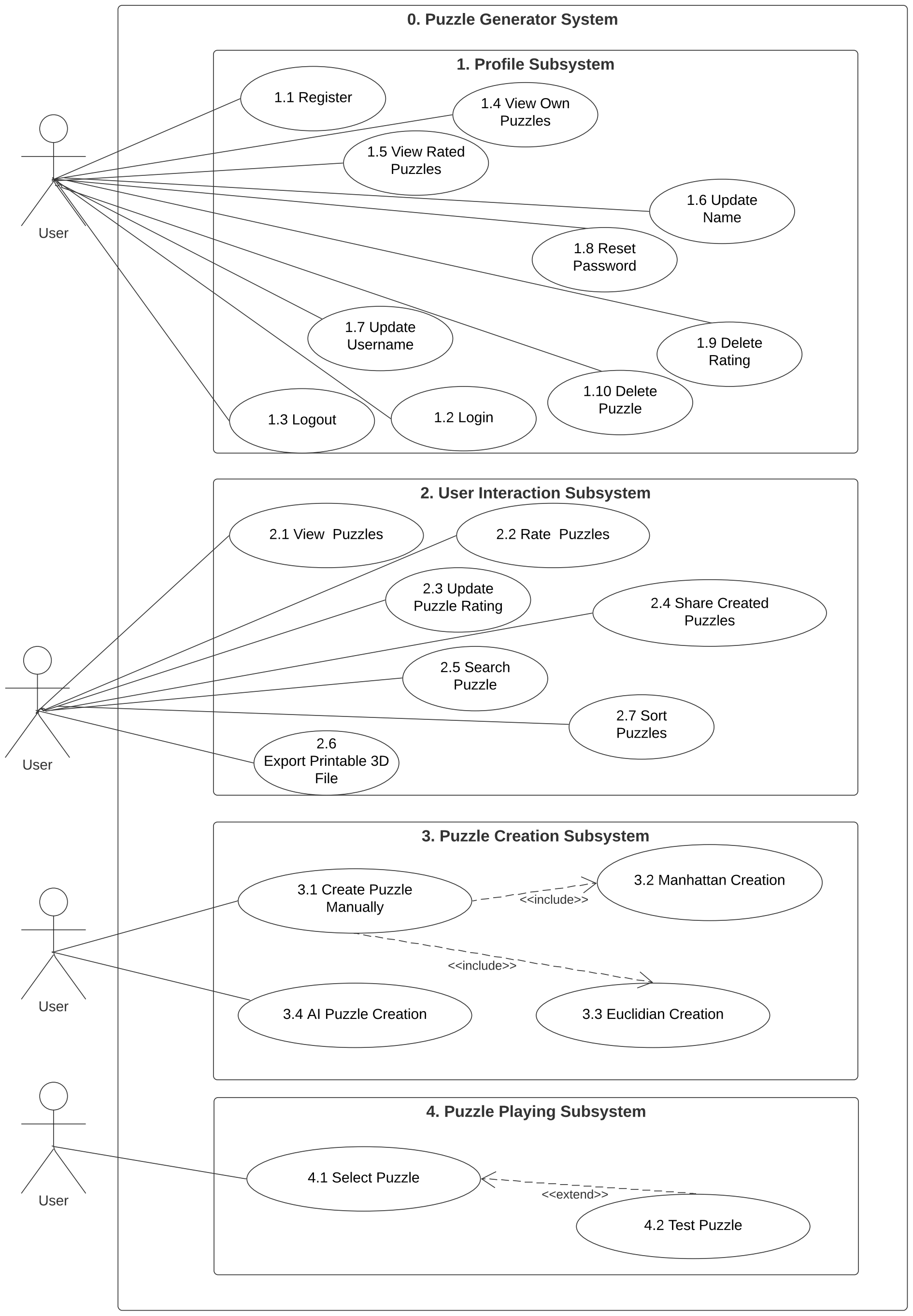
Puzzle Creation Subsystem
The user can uses this subsystem during puzzle creation. The user selects a desired shape from which they can decide to create the puzzle manually or use the AI to generate the puzzle. If a user decides on manual puzzle creation, they may upload an image for the puzzle

Puzzle Playing Subsystem
The user interacts with this subsystem when playing a puzzle. The user selects a desired puzzle and tests it (plays it).

3.3 Requirements

- R1: The System must allow the user to register, and login to, a user profile.
- R2: The system must allow the user to rate puzzles.
- R3: The system must allow the user to view or play puzzles.
- R4: The system must allow the user to create a puzzle.
- R5: The system must be able to generate a puzzle through the use of Artificial Intelligence.
- R6: The system must be able to share puzzles.
- R7: The system must be able to export a puzzle to a 3-dimensional printable file.
- R8:The system must be able to store puzzles that have been created.
- R9: The system must be able to store user profiles.
- R10: The system must be able to search the puzzles.

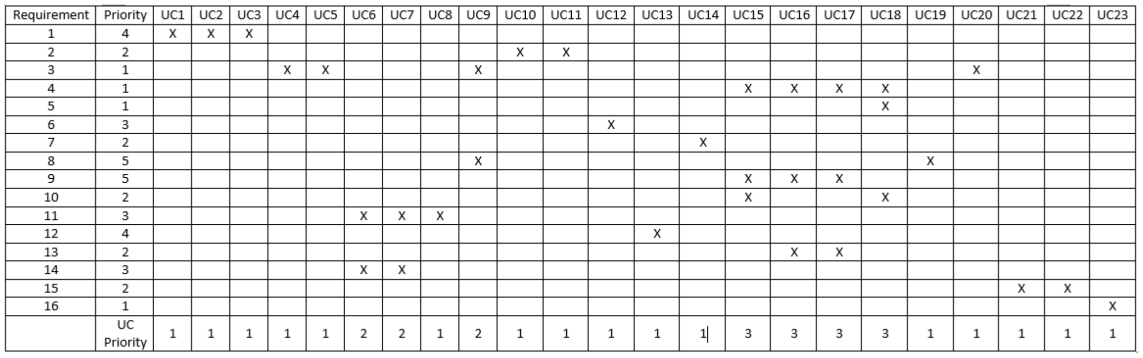
Figure 1: Use Case Diagram



- R11: The system must be able to reset the user password.
- R12: The system must be able to update the username and the name of the user.
- R13: The system must be able to use Euclidean distance or Manhattan distance to make the puzzle.
- R14: The system must update the username, name of the user.
- R15: The system must be able to delete puzzles and ratings as well.
- R16: The system must be able to sort puzzles.

3.4 Subsystems

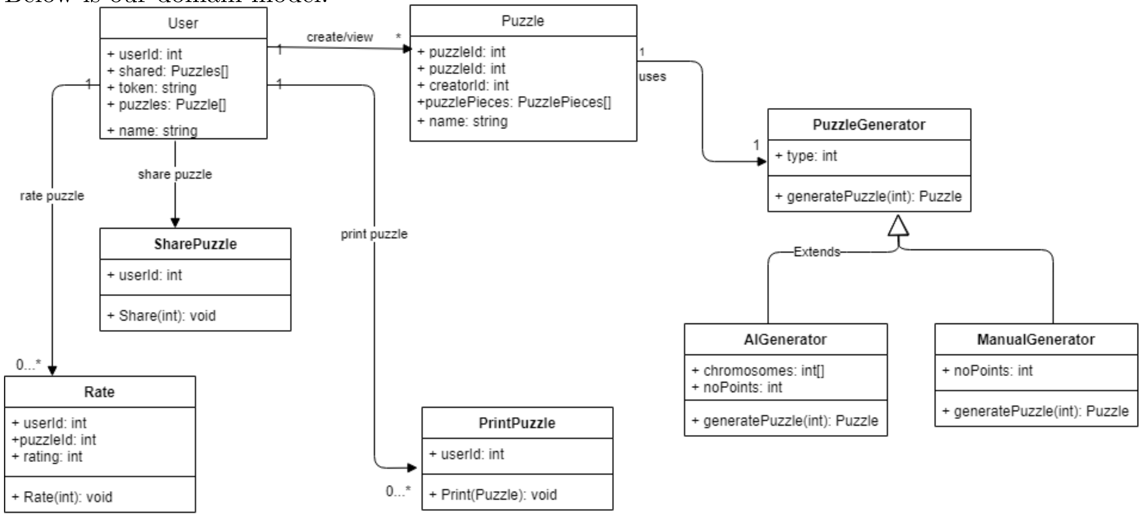
3.5 Trace-ability Matrix



Requirement	Priority	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20	UC21	UC22	UC23
1	4	X	X	X																				
2	2										X	X												
3	1				X	X				X											X			
4	1															X	X	X	X					
5	1																		X					
6	3												X											
7	2														X									
8	5									X										X				
9	5															X	X	X						
10	2															X			X					
11	3						X	X	X															
12	4													X										
13	2																X	X						
14	3						X	X																
15	2																					X	X	
16	1																							X
	UC Priority	1	1	1	1	1	2	2	1	2	1	1	1	1	1	3	3	3	3	1	1	1	1	1

4 Domain Model

Below is our domain model.



4.1 Description of classes

4.1.1 User

The User class will be used for user management. Each user can create as many puzzles as they want and view shared puzzles. Each user can also rate, share and print as many puzzles as they want hence the link between the User class and SharingPuzzle, Rate and PrintPuzzle classes. The user also has an array of all their created and shared puzzles.

4.1.2 Puzzle

The Puzzle class is used to maintain the different puzzles. When the AI is used, i.e. the puzzle is not generated manually; the puzzle class makes use of the puzzle generator class which in turn uses a genetic algorithm to create the different puzzle pieces. The puzzle class is linked to the user class as the user interacts with the puzzle and each puzzle is owned by a user. The Puzzle class is also linked to the SharingPuzzle, Rate and PrintPuzzle classes as each of these services is linked to a certain puzzle.

4.1.3 SharingPuzzle

The SharingPuzzle class is responsible for sharing puzzles. The SharingPuzzle class is linked to the user and puzzle classes as this is needed when using the service. Sharing a puzzle enables other users on the website to view and test the puzzle.

4.1.4 Rate

The Rate class is responsible for rating puzzles. The Rate class is linked to the user and puzzle classes as this is needed when using the service.

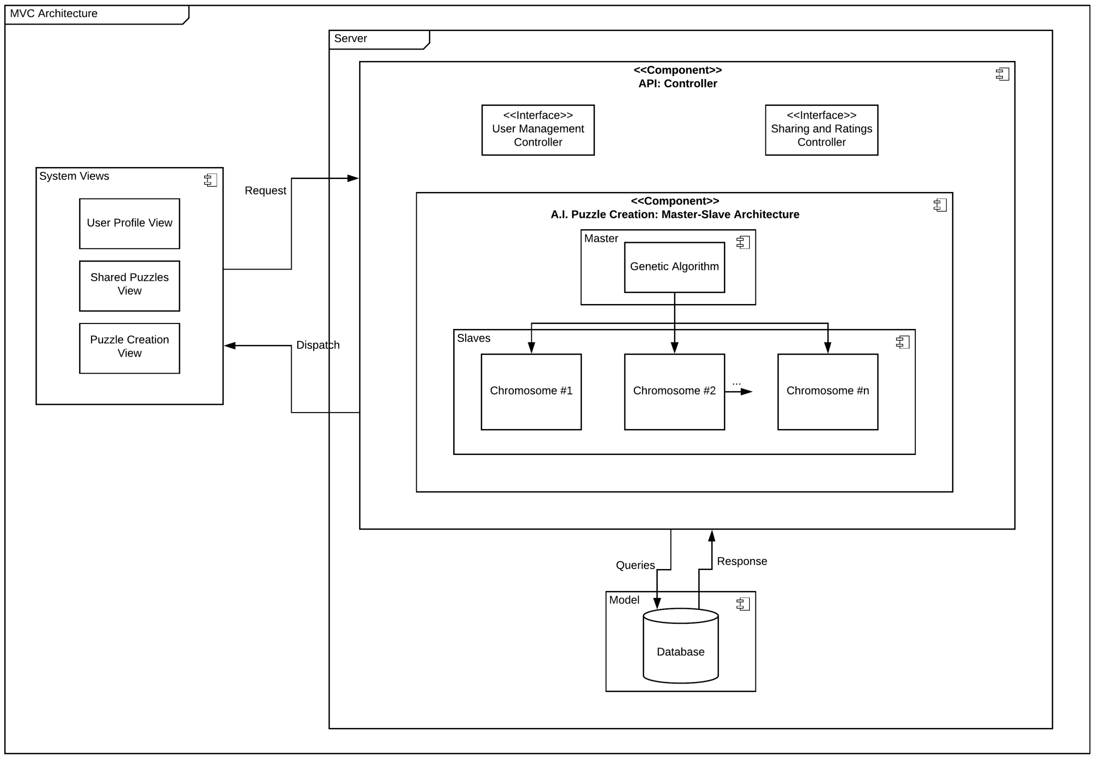
4.1.5 PrintPuzzle

The PrintPuzzle class is responsible for printing puzzles. The PrintPuzzle class is linked to the user and puzzle classes as this is needed when using the service.

4.1.6 AI Generator and Manual Generator

These two classes are subclasses of the Puzzle Generator class. These classes follow the prototype design pattern as the subclasses each contain a seperate implementation of the generatePuzzle() function.

5 Architectural structural design requirements



We made use of the MVC and Master-Slave architectures to represent our system as described below:

5.1 Master Slave Architecture

We are making use of this architectural pattern to implement the AI Puzzle generation portion of the system as follows:

- Slave - the chromosomes represent the Slaves, which in our case are the puzzles generated from the Genetic Algorithm
- Master - the Genetic Algorithm itself is the Master, as it manipulates the data in the Slaves

The architecture helps us achieve the performance aspect of our quality requirement. The user creating a puzzle defines the puzzle criteria, which is the input for our Genetic Algorithm (Master). The algorithm instructs the slaves to generate puzzle objects meeting the criteria and the puzzle object best fitting the criteria is returned to the user as the puzzle image.

5.2 MVC Architecture

Taking into consideration that our system is a game playing system, the MVC architectural pattern was the best choice for us. We decided to use the separation of concerns approach, into designing our system as follows:

- Model - the user data and all data pertaining to puzzles had to be stored in a model.
- View - the user interface happens to be one of the most important aspects of our system as players will user it to share, generate and solve puzzles.
- Controller - this provides a point of data control for our system as we needed a mechanism for communication between the user views and data store.

The user interacts with the website in the browser (view using Angular). The request generated from the user interaction is sent 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.

6 Quality requirements

6.1 Usability

This refers to how the system is easy to use, easy to learn and easy to remember how to use.

The MVC architectural pattern helps us achieve this due its modularization of components. As a result we are able to work on the view independent of other functionality of the website in order to provide the best user experience as follows:

- The website must be easy to learn and self explanatory for users within and outside of our target groups
- Since it is a game playing system, the website must be aesthetically pleasing. Special attention is paid to color and font usage and text readability.
- We provide tutorials on how to generate puzzles and correct user errors when filling in forms
- We will be able to measure usability through usability testing which aims to measure the amount of time taken on average by users to perform tasks, average number of errors encountered when navigating the site and general feedback given. We aim to have our website be at least 80% usable without the need for tutorials.

6.2 Performance

This refers to the system's ability to achieve its intended purpose efficiently and the speed the system takes to do tasks. Performance is essential for our system because:

- The system is largely reliant on puzzle data from the database so the database needs to be able to handle simultaneous requests instantaneously, this being at least 30 requests per second.
- Our system is a real time puzzle generator and solving system. A puzzle should be generated by the genetic algorithm AI system within 30-90 seconds so that the user does not have to wait long periods of time but rather has a good user experience.
- Using the Master Slave architecture for AI puzzle generation, the master instructs the slaves to generate puzzle objects as per user specifications and the slaves will all do the same task, with the best puzzle object generated chosen as the solution

6.3 Reliability

This refers to the endurance and consistent performance of the system in the real world during its lifetime.

- Since users can be expected to generate or solve puzzles at any time of the day, our website needs to be available at least 89% of the time on our server.
- The system needs to be able to load to the website view with correct and accurate information 99% of the time i.e. the correct puzzle images must be matched up to the correct creator, rating and other information.

The separation of concerns of the MVC architecture allows our system to achieve reliability:

- We can identify failure points at any point in time, if there are any.
- We are able to test the functionality of the Model, View and Controller independently through unit testing and as a whole through integration testing
- Should a component/module e.g. model or controllers encounter an issue, the view will still be available allowing the user to interact with the website to some extent

6.4 Scalability

This refers to the ability of our system to scale efficiently to be able to handle a growing number of users and operations. The Puzzle Generator system needs to be scalable because:

- We know a large number of puzzles will be generated by our users. Our database needs to be able to store up to a thousand records related to these puzzles, including shared statuses, ratings and the puzzle images themselves so that when users request to view this information, it can be readily available.
- As a result, we are making use of pagination to display puzzles on each page to cater for the large number of puzzles.
- We except to have at least over 100 users registered on our system as the popularity of our website grows. Therefore, our website needs to be able to handle a large number of requests at the same time (e.g. 20 users may be actively sharing puzzles while other 20 are making ratings). The website should be able to ensure these users get the best experience when doing so.
- Our filter queries happen in two ways: filtering on the View component of the MVC architecture, and filtering on the Model and Controller MVC components to cater for large scale searches.

6.5 Security

The puzzle creations made by users are their intellectual property. Therefore, users have the have the right to decide which of these puzzles to share and keep private. This along with user details need to be kept secure and only for the eyes of the user. As such:

- Our system makes use of a token system. A user is assigned a token on registration which will be used with every request made to the database, such as when deciding to share a creation.
- If a user has been inactive for at least one hour, they are automatically logged out of their account

- MVC also allows us to achieve security due to its modular structure: all sensitive data is handled by the controller and model.

6.6 Cost

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

- The website runs on a free tier web server.
- The website uses open source architectural technologies such as Angular and Node Express.

7 Technology requirements and Reasons

7.1 Hosting

- Heroku's free tier is used to host the website
- This is to allow users to freely access the Prometheus Puzzles website in order to generate, share, rate and solve puzzles
- Our website is freely accessible through this link.

7.2 API Server

- We made use of NodeJS using the Express framework to host our API on the server side.
- We made use of the REST framework in order to write our endpoints. This allows us to easily verify the functionality of our system as known status codes can be tested in our unit tests for the various endpoints defined.
- Since our API handles CRUD database operations, we made use of the Sequelize ORM framework instead of normal SQL queries. This largely helps make our code compact and readable
- We chose to use NodeJS, Express and Sequelize ORM for our API because of the familiarity of these technologies to us as a group, the efficiency in which they allow us to pass puzzle data back and forth between View and Model, as well as the fact that they are widely used and adopted technologies in the real world.

7.3 Website

- Our website is made using Angular 9

- Angular allows for a single page website with various components attached to it, each performing its own set of functions. This was the major selling point of this technology to us as it allows us to separate our website functionality into - sharing components, rating components, creation components, etc which all work together at the end of the day to ensure the smooth running of of website.
- Angular's separation of concerns further allows us to define separate service files which we used for communicating with our API
- The technology's modular structure allows us to identify errors easily as only the affected component will stop working should an error exists.
- Additionally, Angular makes use of TypeScript, HTML and CSS which are all technologies we have learnt over the years and are fairly easy for use to implement

7.4 Database

- We are making use of the PostgreSQL database to store our data.
- Our system makes use of simple relational data which can be easily stored using a relational model database
- Tables are defined for each main entity, i.e. Users, Puzzles, PuzzleRatings. The tables are also normalized to ensure referential and entity integrity.

7.5 Puzzle Generation

- Both AI and manual puzzle generation algorithms were written in JavaScript
- This was ideal as our system in largely comprised of Javascript variations namely NodeJS and TypeScript so integrating that into the Angular website was an easy task

7.6 Unit Testing

- We are making use of Chai for unit testing
- This tester integrates well with NodeJS hence it was the ideal technology to use to test our API endpoints as our API server is written in NodeJS

7.7 Integration Testing

- We are making use of Selenium Webdriver to test the integration of our system

- Our tester aims to emulate user operations such as logging into the website, sharing a puzzle and more. This verifies communication is indeed happening between the website view, API server (by making the request) and database server (returning requested data).
- We used the technology because our system is website based and the best way to verify that all components are working as intended was to emulate user actions on the website

8 Coding standards document

8.1 Server component

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

- Front-end files are listed under a directory called **/src**. The structure of the directory is as follows:
  - /app
    - /app/models
    - /app/navbar
    - /app/pages
    - /app/rate-dialog
    - /app/services
  - app.module.ts
  - app.component.ts
  - app.component.spec.ts
  - app.component.html
  - app.component.css
- 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
  - users.js
  - puzzles.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.

8.2 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.

8.3 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 directory - it contains the database configurations .
- /routes directory - 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 directory - 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.

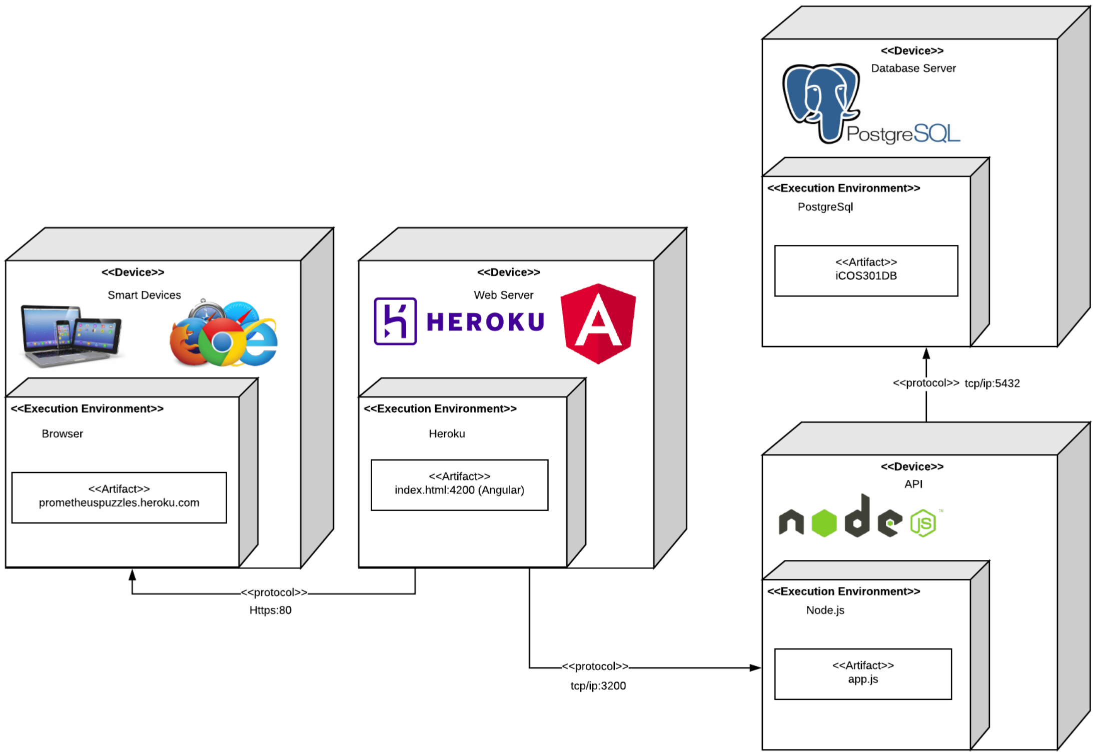
9 Technical Installation manual

The Technical Installation manual can be found at the following link:
https://www.overleaf.com/read/zhgbzhfdwmbt

10 User manual

The User manual can be found at the following link:
https://www.overleaf.com/read/hkvxmjmcfhcq

11 Deployment Model



The image above is our deployment model showing the various components of our system and technologies used to bring our system together.

The user will use a browser on any smart device to access our website at prometheuspuzzles.herokuapp.com.

The website is hosted on our web server. The web server we are using is Heroku and we used Angular as our programming language. The web server use the http protocol to serve our website on port 80. Angular runs on port 4200 on our web server.

We communicate with our database via our API. The API is written in nodejs. The web server uses port 3200 to communicate with our API. and our API uses port 5432 to communicate with our database. These communications are done with TCP/IP protocols.

Our Database Server is a PostgreSQL. We used pgAdmin locally to interact with the database.