TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Artur Bassov 163940IAPB

WEB APPLICATION FOR SORTING AND FILTERING GAME ICONS FROM GOOGLE PLAY

Bachelor's thesis

Supervisors: Deniss Kumlander
PhD
Viktor Mihhailov
MSc

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Artur Bassov 163940IAPB

VEEBIRAKENDUS GOOGLE PLAY MÄNGUIKOONIDE SORTEERIMISEKS JA FILTREERIMISEKS

Bakalaureusetöö

Juhendajad: Deniss Kumlander
PhD
Viktor Mihhailov
MSc

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Artur Bassov

12.05.2019

Abstract

The objective of this thesis is to create an internal solution for Creative Mobile OÜ to simplify the research for a game icon performance in the application markets, starting with Google Play as a representative. The solution will save time and increase performance of creating a new art.

The result of this work is a full-stack web application, where icons can be sorted and filtered via user interface by country, game category, google inner ranking, colors and tags of objects in the icon. Back-end is written using Node.js, front-end is implemented with Vue.js.

In the work progress several iterations are made, target group with their functional requirements are defined, the application is developed and deployed to the host server.

Solution is mainly focused on artists and creative managers.

This thesis is written in English and is 46 pages long, including 5 chapters, 30 figures and 1 table.

Annotatsioon

Veebirakendus Google Play mänguikoonide sorteerimiseks ja filtreerimiseks

Diplomitöö eesmärgiks on luua sisemise lahenduse Creative Mobile OÜ jaoks. Rakenduse poes, nagu näiteks Google Play, omab iga mobiilimäng enda ikooni. Kunstnikel on segadus, milline ikoon oleks kasutajate jaoks kõige atraktiivsem, meeldejäävam ning mis tooks rohkem klikke ja allalaadimisi. Kunstnikud uurivad olemasolevaid lahendusi rakenduse poes ning baseerivad oma tööd nende põhjal. Selline uuring on manuaalne, seega aeganõudev ega pruugi olla tõhus. Probleemi püstituseks on saada kõige efektiivsem võimalik ikoon mängu jaoks. Ideaalne lahendus on luua automaatne teenus, mis esitaks, milline oleks kõige sobivam ikooni kunstlik teostus rakenduse poes, et kasutajad kohe reageeriksid sellele, vajutaksid ikoonile ning laadiksid mängu alla. Kuid sellist ulatuslikku lahendust ei ole olemas. Et jõuda sellise lahenduseni, peaks alustama millegi lihtsamaga. Põhiline lahendus on teha inimese poolt uuring, et leida trende ikoonide seas mugavamalt ja tõhusamalt.

Selle töö tulemuseks on full-stack veebirakendus, kus ikoone saab sorteerida ja filtreerida riigi, mängukategooria, google sisemise edetabeli, värvide ja ikooni objektide märgistuste alusel kasutajaliidese kaudu. Back-end on kirjutatud kasutades Node.js, front-end on rakendatud Vue.js.

Tööprotsessis on tehtud mitmed iteratsioonid, sihtgrupp koos funktsionaalsete nõudmistega on määratud, rakendus on välja töötatud ning paigaldatud host-serverile.

Lahendus on keskendunud peamiselt kunstnikele ja kreatiivjuhtidele.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 46 leheküljel, 5 peatükki, 30 joonist, 1 tabelit.

List of abbreviations and terms

AJAX	*Asynchronous JavaScript and XML* Is a technology to send or retrieve data in the background for the web application
API	*Application Programming Interface* Is a set of routines, protocols and tools for building software; API specifies how software components should interact
AWS	*Amazon Web Services*
DOM	*Domain Object Model* Is an API for JavaScript to modify HTML objects/elements
EC2	Amazon *Elastic Compute Cloud*
GUI	*Graphical User Interface* Visual elements with which user interact
Google play	Is a digital distribution service operated and developed by Google LLC
HTML	*HyperText Markup Language* Is the standard markup language for creating web pages
HTTP	*HyperText Transfer Protocol* Is an application for distributed systems
IP address	*Internet Protocol address* Is a numerical identifier of each node in the computer network that uses Internet Protocol for communication
Icon	Image that represents a game or application on the application market
JSON	*Java Script Object Notation*
ML	*Machine Learning* Is the science of getting computers to act without being explicitly programmed [1]
MVP	*Minimal Viable Product*
npm	*Node Package Manager* Is a package manager for the JavaScript runtime environment Node.js
S3	Amazon *Simple Storage Service*

SPA	*Single Page Application* Is a web application that dynamically rewrites the current page rather than loading entire new page
SSH	*Secure Shell* Is a cryptographic network protocol
UA	*User Acquisition* Department whose goal is to attract new users for the product
UI	*User Interface*
URL	*Uniform Resource Locator* Web address; is a location of a web resource on a computer network
UX	*User Experience* Person’s emotions and behaviour about using a particular product, system or service

Table of contents

1 Introduction ... 13
  1.1 Problem ... 13
2 Analysis ... 15
  2.1 Target group designation ... 15
  2.2 Defining needs ... 16
  2.3 Iterations ... 16
  2.4 Functional requirements ... 16
  2.5 Non-functional requirements ... 17
3 Used technologies ... 18
  3.1 Single Page Application (SPA) ... 18
  3.2 Model-View-Controller (MVC) ... 19
  3.3 Model - View - View-Model (MVVM) ... 20
  3.4 Frontend with Vue.js ... 21
    3.4.1 Additional used libraries and components ... 22
  3.5 Backend with Node.js ... 22
    3.5.1 Additional used modules ... 23
  3.6 Database with MongoDB ... 23
    3.6.1 MongoDB Atlas ... 23
  3.7 Amazon Web Services ... 24
  3.8 Image classification API - Imagga ... 24
4 Development ... 27
  4.1 Service architecture ... 27
  4.2 Server-side ... 28
    4.2.1 API ... 28
    4.2.2 Obtaining game data ... 30
    4.2.3 Storing game data ... 31
    4.2.4 Processing data ... 32
    4.2.5 Filtering data ... 34
    4.2.6 File structure ... 36

4.3 Client-side 37
4.3.1 Overall structure 37
4.3.2 Components 38
4.3.3 User Interface 40
4.3.4 File structure 41
4.4 Service testing 42
4.5 Future development 43
5 Summary 44

List of figures

Figure 1 Traditional page and SPA Lifecycles 18
Figure 2 The MVC Pattern 19
Figure 3 The MVVM pattern 21
Figure 4 The MVVM patter with Vue.js [10] 21
Figure 5 Sample icon for ML image classification testing 25
Figure 6 AWS Rekognition sample results 25
Figure 7 Google Vision AI results for the sample 26
Figure 8 Imagga results for the sample 26
Figure 9 Client-Server Architecture of the Service 27
Figure 10 Get data from Google Play by query 30
Figure 11 Example of json data fetched 31
Figure 12 mongoose two queries in one-line example 31
Figure 13 Simplified mongoose document scheme for collection 'games' 32
Figure 14 Imagga API request URL to process image for tags 32
Figure 15 Generated tags example 33
Figure 16 Imagga URL for colour processing request 33
Figure 17 Colour data saved about image colours 33
Figure 18 Closest palette colour is determined for shades 34
Figure 19 Generated tags by Imagga are manually grouped 35
Figure 20 MongoDB pipeline pattern in quering example 36
Figure 21 Server project structure 37
Figure 22 Client overall structure 38
Figure 23 Home Component - Desktop view 39
Figure 24 Game Monitor Component - Desktop view 39
Figure 25 Game Detail Component - Desktop view 39
Figure 26 Admin Games Component 40
Figure 27 Filter bar at Game Monitor Component - Mobile view 40
Figure 28 Games result at Game Monitor Component - Mobile view 41
Figure 29 Client project structure 42

Figure 30 Axios get request example 42

List of tables

Table 1 API endpoints 30

1 Introduction

There are lot of services that promise to save time and make specific work easier, most of them are general, but companies need something more specific to solve their problems. Companies are creating their own internal services that can be modified and honed according to company needs, however there are additional costs to this solution.

Companies resort to creating an internal solution if there is enough demand inside the company, there is no suitable solution from the outside and using the solution will save more than the cost of the effort.

The purpose of this paper is to make a web application that allows to increase effectivity of the manual research of the game icons on the market. The research is usually made before creating or updating an icon for the products owned by Creative Mobile OÜ.

Work progress was divided into three main parts. Firstly, a series of interviews were held with people in different positions to define new service scope and requirements and focus on them. Secondly, being developed application was constantly iterated with the potential users. Finally, application was deployed and showed to more people for feedback.

As a result, a web application with possibilities of sorting by country, internal ranking and game category, as well as filtering by colors and tags of object in the icon was created.

1.1 Problem

At the beginning of the creation of the new promo-materials, such as social-media banners, trailers, interactive ads and application icons, there is an uncertainty of selecting an approach of how to do that. There always exist a reference market with major players, who has a great experience in doing that job. Their works ought to have most performance, which means that their art is making users to pay attention for that, attractive to users, is memorable and, as a consequence gains traffic. That is what ‘performance’ means for the promo-materials.

It is useful to analyze market with existing solutions to afterwards create a brand-new one, which complements market by being in a trend.

The problem is that there it takes immense amount of efforts to analyze promo-material market manually. It is not efficient as well, because human could not mention some occasionally appearing pattern just by watching on the unstructured data sets.

For example, until now this kind of analysis is made by visiting pages of games in Application Markets and screenshotting them, trying to find trends in shapes, objects or colors used in the images of the promo-materials.

The focus on icons from all the promo-materials was immediately taken, because they are one of the most available type of promo-art – could be taken from any application market. Also, they are limited in size, that forces to pay attention to one object in general. Patterns could be found in them more easily than in any other promo-material.

In this way, sticking with icons, there could be stated services that will help with analysis for the most performant icon on the market of different levels. The higher the level, the more advanced the service is. Service of the highest level should do something like just outputting the best icon of all the times in the market. A little bit simpler service will define the most performant icon for the current market. Lower service will say which parameters icon should have for the current market to be successful. The first autonomous level is where service defines icon parameters and gives possibilities to manage this data about parameters for the further human manual analysis. Below only manual level, where artists currently are.

This way, service where users could sort and filter applications by icon image parameters, such as objects depicted in them and colours they consist of, is the first step to solve the problem of creation the most performant promotional art.

There is already a solution of common from Game Refinery called visuals explorer, but it is part of an enormous service which is pricy and is not the main focus of the service, thence has a questionable performance (is in beta).

Initially the problem came from the artist from the Research & Development team.

2 Analysis

Initially there were no requirements for the service to be created, therefore they had to be determined. Task was very abstract, and some constraints had to be found. Several statements were known for sure – the service should help to work with data obtained from Google Play. Process of defining the requirement is described in this chapter.

Author held two rounds of interviews.

The purpose of the first interview was to identify the target group of positions for which a service with the ability of obtaining data solely from Google Play (with the possibility of connection of other markets like iTunes) will be the most beneficial.

The purpose of the second one was to interview more people on the earlier defined position and ask them more specific questions about the future service.

2.1 Target group designation

During the first interview a more general questions were asked. Interviewed were given the general description about the possible future service. Service was presented as an entity that will help to make an icon analysis based on data from Google Play. Interviewers were on positions of Producer, Creative Manager, Artist, Analytics and Head of User Acquisition.

Every of these positions had interest in a service, that will state analytical approach to the game icon creation, but positions like Producer, Analytic and UA lacked data that Google Play could provide. They needed more user behaviour data, like gain in downloads, did user came without an ad campaign or was it the same user who didn’t react to the previous icon version. This information can’t be found at Google Play for every application.

The conclusion of the first part of the interview was that the service that can operate with raw image data can be best applied to artists. Depending on the task artists will compare icons available in Google Play to receive better representation of the actual trends for

some specific category, country, etc. Sorting and filtering will help artist to determine abstract trends and patterns.

2.2 Defining needs

The second interview round was aimed to find out are artists interested in the service, what are their expectations and wishes for functionality. Artists showed their interest towards color filtering and filtering by tags of objects depicted in icons. Mainly artists were asked to evaluate different object tags like Person, Building, Animal, Text, etc. on the subject of their importance in the service.

After that stage initial requirements for the service were stated and the development of the service itself was started.

2.3 Iterations

Service development was carried out iteratively. A total of 4 iterations of different scope and duration took place with potential users, who are artists. Full scope was chosen so that MVP is exactly done, and if there are any tasks going out of scope service will be still viable for demonstration.

Every iteration specified and maintained better functional and non-functional requirements that finally can be stated.

2.4 Functional requirements

1. Application data should be collected from the Google Play.

2. First 100 applications from two internal market rankings (Top Free, Top Grossing) should be collected.

3. Applications from the following categories should be collected: Action, Puzzle, RPG, Strategy, Racing, Casual (so further Applications are Games).

4. For every single game should be collected at least the following data: icon image, game name, publisher name, link to the market page, category/genre, ranking, region/country for which game is collected.

5. Game should be able to be sorted by country, ranking and genre/category.
6. Icons should be able to be filtered by tags named after objects depicted in the icon images.
7. Icons should be able to be filtered by colours.
8. Result of the filter query without colour filtering should show colour summary.

2.5 Non-functional requirements

1. Google Play should be parsed for game data.
2. User interaction with the service occurs via GUI of the web application.
3. Data is stored and processed on the server-side.
4. Data is sent and only displayed on the client-side.
5. Icon image classification should be made using Machine Learning.
6. The service should be scalable for different Application Markets.
7. The service should be developed so it is not difficult to connect different Machine Learning models.
8. Database should store other valuable game data, so it could be reused in other service implementations.
9. Web application should be developed with the UX/UI principles.
10. Icons should be sorted by colour volume percentage if colour is used while filtering.

3 Used technologies

In this chapter author describes technologies, frameworks, services used in creation of the service and explains their choice.

3.1 Single Page Application (SPA)

A single page application is a web application that will dynamically rewrite current page according to user interactions rather that load entire new pages. This approach allows to use web application without interruptions, which makes user experience similar to desktop applications. [2]

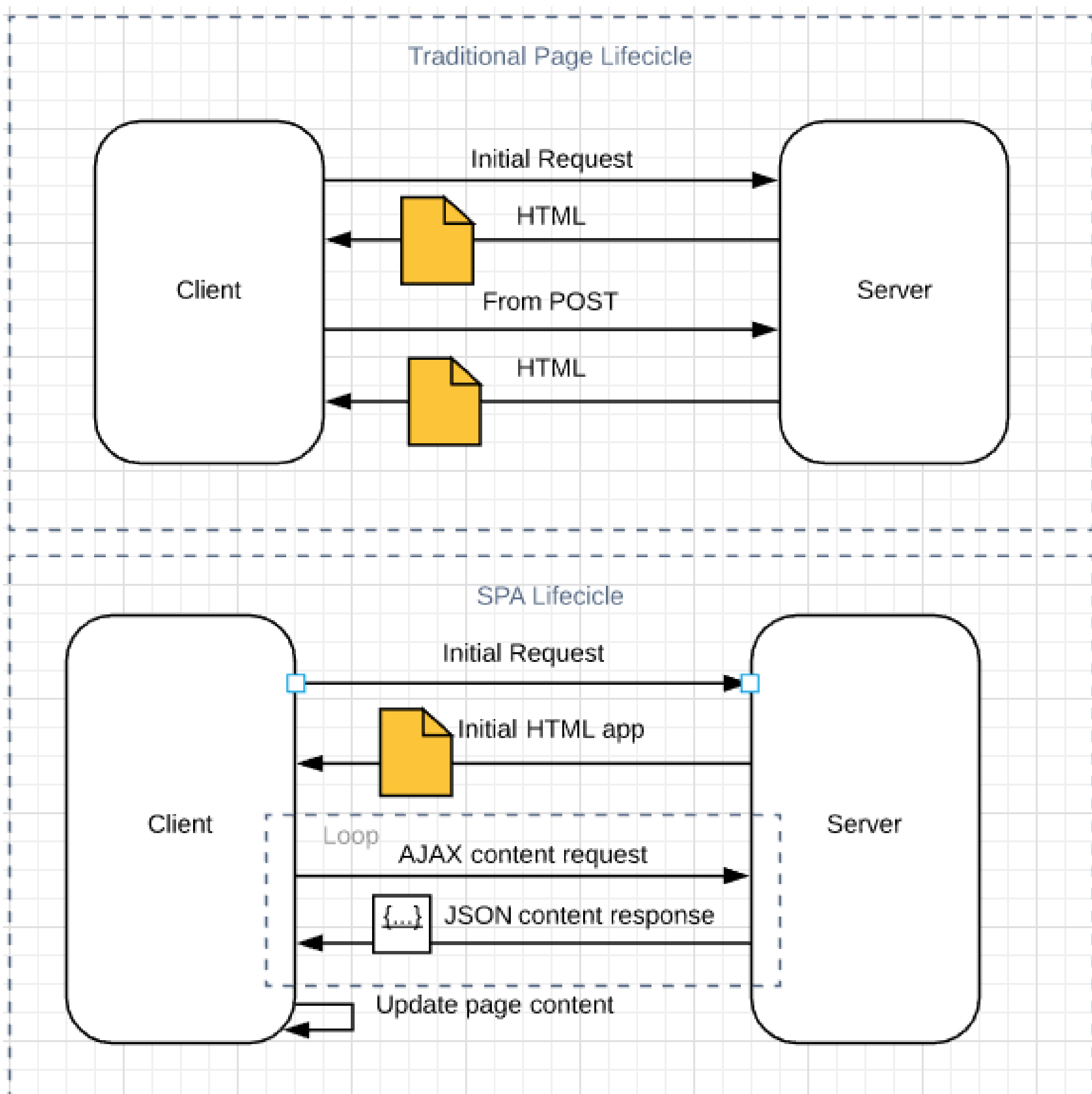

Figure 1 Traditional page and SPA Lifecycles

It is possible due to JavaScript ability to manipulate the DOM elements on the current page. Client sends requests to the server and servers responds with the HTML page only for the first time, then all interactions with the server are via AJAX requests to fetch content from the server to update the current page with it. In the multi-page application server will respond with the whole new page and client reloads it in the browser (Figure 1). Additionally, in the SPA, History API allows to alter browser URL without reloading the page, this allows to provide the perception and navigability of separate logical pages. This means that if redirected to the specific URL the page with the specific content will be loaded, not the initial page. [3]

3.2 Model-View-Controller (MVC)

Model-View-Controller is a software architectural pattern that divides application into three interconnected parts (Figure 2). It is used for applications with the GUI. The Model is the application object the view is representation of that's object information, and the Controller defines the way the user interface reacts to the user input. Designing application with this pattern increases components flexibility and reuse by decoupling them. [4] [5]

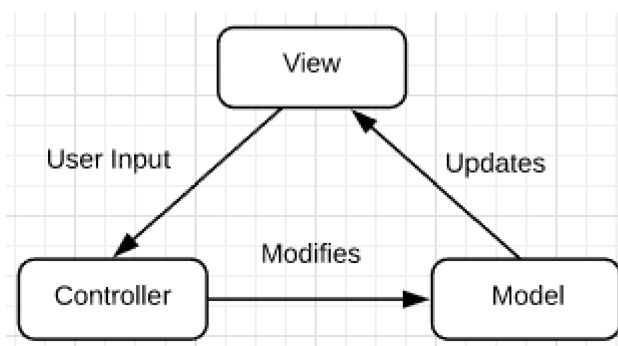

Figure 2 The MVC Pattern

Traditionally this architecture has become popular for designing web applications. Many web application frameworks use this pattern for the development out of the box. [6]

More specifically about every type of object in the pattern:

- Model – is the application's dynamic data structure, independent of the user interface. It directly manages the data and responds to the view about it's state and responds to instructions to change it from the controller

- View – Manages data display for the user. Separating this component allows to create different information representations, for example different charts could represent the same data set.

- Controller – manages the user input to accordingly change model data and consequently it's representation (view).

3.3 Model - View - View-Model (MVVM)

However, MVC pattern seems to be a bit abstract and tends to have different interpretations. It is considered that MVC's object Controller have another meaning because nowadays developers do not think about controlling input itself too much as they did in the days of creation of this pattern, somewhere in 1979 [7]. A more recent variant of MVC is the MVVM pattern.

MVVM is a software architectural pattern that consists of the three objects. It was created to make easier the creation of the user interfaces. It is inspired by another variation of the MVC, the Model-View-Presenter (MVP) pattern. It has popularity on various UI programming platforms.

View has common specification for both patterns. It separates a view from its behavior and state. View merely becomes a rendering of the view-model or Presentation Model of the MVP pattern. [8]

View-model is an abstraction level of a view. The term means "Model of a View". It provides a specialization of the Model that the View can use for data-binding. This is the difference between MVVM and MVP. Unlike the Presenter in MVP, a ViewModel does not have a reference to a view, the view binds to properties on a ViewModel, which exposes data contained in model objects and other state specific to the view. Data bindings allows changed new values to automatically propagate to the view. This maintain both view and model in sync with each other.

One example is when the user clicks a button in the View, an instruction on the ViewModel executes to perform the requested action. The ViewModel, never the View, performs all modification to the model data (Figure 3). [8]

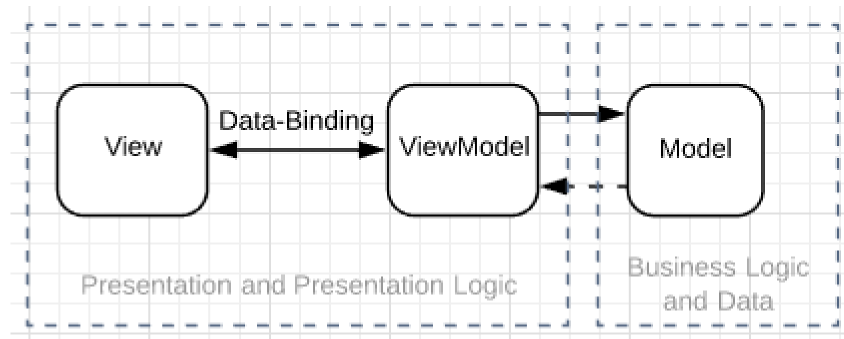

Figure 3 The MVVM pattern

The reason data-binding can't be implemented directly between View and Model is that UI may want to perform complex operation that must be implemented in code which does not fit the strict definition of the View and are too specific to be included in the Model. [7]

3.4 Frontend with Vue.js

Vue is a progressive framework for building user interfaces. [9]

Technically, Vue.js is focused on the ModelView layer in the MVVM architecture pattern and connects Model and View via two-way data bindings. Utilization of this pattern makes Vue capable of powering sophisticated Single-Page Applications. Philosophically, the goal is to provide the benefits of reactive data binding and composable view components with an API that is as simple as possible (Figure 4). [10]

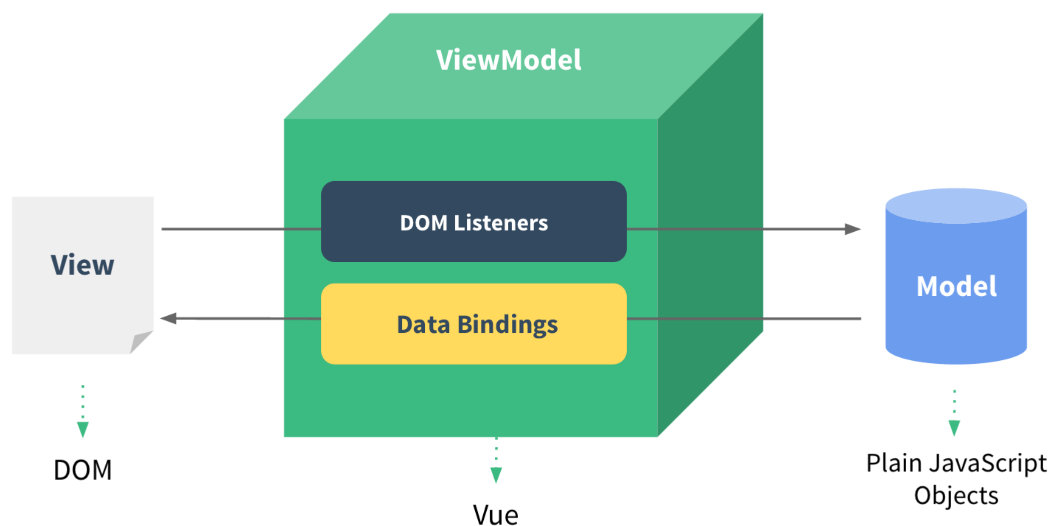

Figure 4 The MVVM patter with Vue.js [10]

Vue manage data inside HTML elements in a reactive manner, which means Vue can automatically detect any property changes and re-render the affected element.

Author has never faced with any frontend frameworks before. Basically Vue.js was chosen because of its simplicity. It has a detailed documentation, small build size and plain learning curve. Official and third-party benchmarks show Vue performance is better than competitors have. [11]

3.4.1 Additional used libraries and components

- Vue-router is the deeply integrated official library for routing to create Single Page Applications.
- Vue-bootstrap is a front-end CSS library wrapper for Vue.js to build responsive web design.
- Vue-multiselect is a solution for select user interface components.
- Axios is a promise-based HTTP client for the browser and node.js, used to execute http request to the server.

3.5 Backend with Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. [12]

It is an open-source, cross-platform JavaScript run-time environment to execute JavaScript code outside of a browser. For example, Node.js allows developers use JavaScript to write server-side scripts to produce dynamic web page content before the page is sent to the user's web browser. This represents a "JavaScript everywhere" paradigm, which is about unifying web application development around a single programming language for both client and server-side scripting. [12]

Node.js has an event-driven architecture capable of asynchronous input/output operations. These design choices make web applications with many input/output operations very scalable and able to sustain high loads.

Author choice of this solution was based on the throughput possibilities, large collection of core libraries for different functionality called modules, possibility to run code on Linux because of utilization of this Operation System on most of the machines of hosting

services, literally simple as three command line command installation on the empty machine, and desire to use one programming language for client and server-side development.

3.5.1 Additional used modules

- google-play-scraper is a module to scrap application data from the Google Play store. It parses data from Google Play web page by requests which are properly documented. Requests return JSON objects with the requested data. It has a very significant part in the whole service. [13]

- Express.js is a framework for building web applications and APIs. It is nearly essential part of the server. It helps to configure server and it handles every request to the server. It has lots of HTTP utility methods that makes API creation quick and easy. [14]

- Mongoose is a MongoDB object modelling tool designed to work in an asynchronous environment. It provides connection to the database and utility methods to manage data with database. Defines models for database documents to define the structure and types of the stored data. [15]

3.6 Database with MongoDB

MongoDB is classified as a NoSQL database. It stores JSON-like documents with schemas. Documents are stored in collections. It supports filed, range queries, regular expressions searches. [16]

Data scraped from Google Play is a JSON object. It is the main entity to store, so NoSQL database that could store just scraped document suites the best. MongoDB is the most popular solution with these prerequisites, therefore it has big community and lots of cases are covered in the internet discussions. Query possibilities and pipeline pattern is a great boon. So, author has chosen it for use in the development.

3.6.1 MongoDB Atlas

MongoDB Atlas is a database as a service created by the experts who design and engineer MongoDB. [17]

Launching an application on any database requires to plan carefully and ensure performance, high availability, security and backup recovery. Atlas handles every of it itself, automating database administration tasks such as configuration, patches, scaling events, backups and more.

For this project it is inestimable time saver, during the development it could be ensured that any of these obligations are under control and configured the best for the chosen size of database. This allows to focus on application development which matters the most in the case.

Another advantage is that is has a free tier and it is easy and fast to setup. So author made choice not to run database on the extended server, but use this server as a great commodity.

3.7 Amazon Web Services

Both client and server side are deployed to AWS in order to share application between workers for the demonstration purposes. Web application is available at http://icon-monitor.s3-website-us-east-1.amazonaws.com/

Web page is hosted using the hosting possibilities of Amazon Simple Storage Service (Amazon S3). Amazon S3 is an object-storage service that offers industry-leading scalability, data availability, security and performance [18].

Server is running on Linux instance of Amazon Elastic Compute Cloud (EC2). Amazon EC2 is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers [19].

Both services are intuitive to use, has lots of possibilities. Author had a free usage period of the services, was familiar with them and wished to acquire more experience in using them, these are the reason of the decision made in favour of AWS.

3.8 Image classification API - Imagga

Creation of the personal solution was not the option. It is time consuming, which affects scope, it is processing time consuming, which is costly, there is not enough icons with

metadata available to train model and no specific algorithm to train it specifically for draw-art as icons.

Selection of the suitable solution was concluded after testing some samples on of them is shown on the Figure 5, which is one of the most detailed images. There are three ready-solution candidates, they are: AWS Rekongition [20], Google Cloud Vision AI [21] and Imagga [22].

Figure 5 Sample icon for ML image classification testing

Amazon utility could not identify anything useful in the picture, even for the most detailed samples (Figure 6). It recognised the drawn fountain as a wedding cake with a 61% of confidence.

Figure 6 AWS Rekognition sample results

Google utility showed better results.

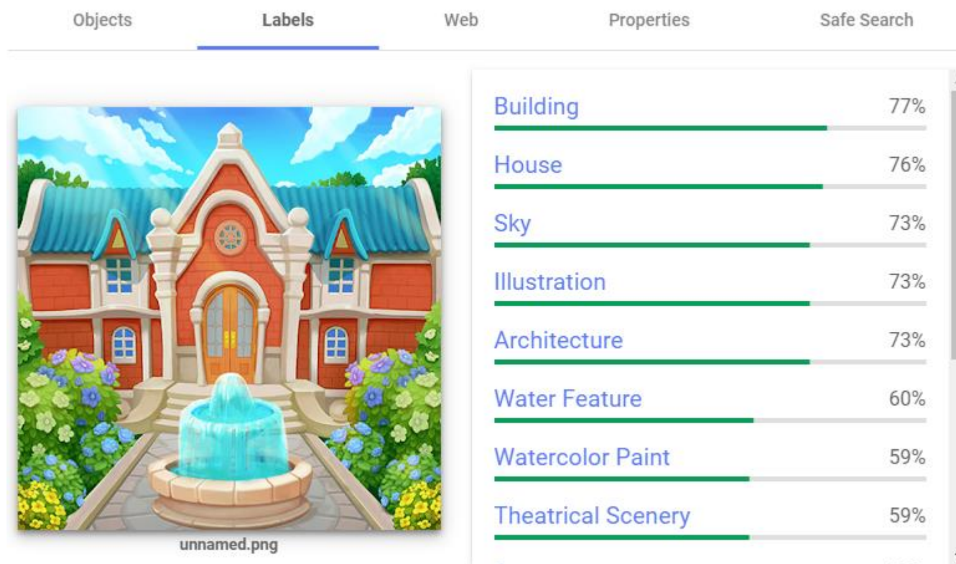

Figure 7 Google Vision AI results for the sample

Tool from Google showed descent results detecting building in the image, realizing the is painted sky, which indicates it is outside, and noticed water pattern. However, Imagga's tags were more specific and seemed to be most suitable, so it was chosen.

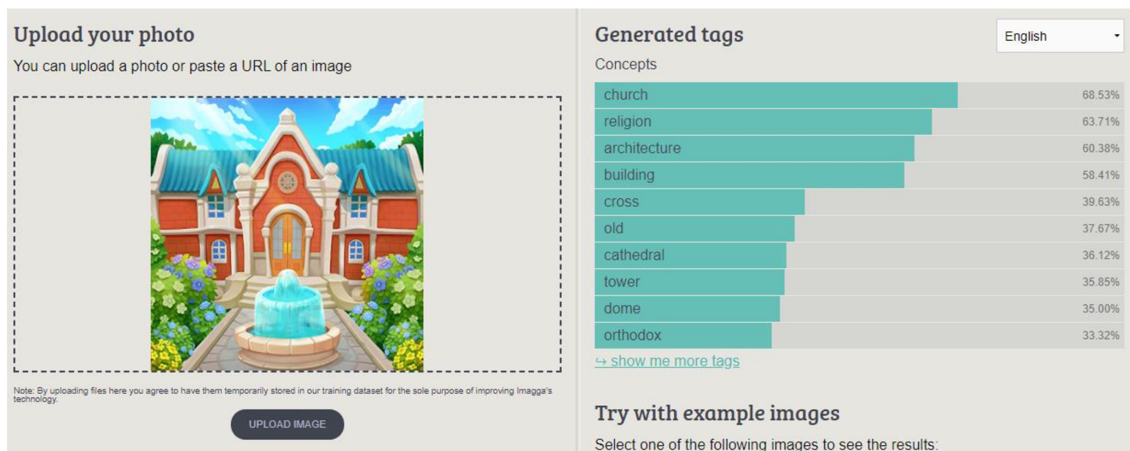

Figure 8 Imagga results for the sample

Imagga offers solutions for many use cases and author decided to use Imagga's Image Recognition services.

4 Development

Service was constantly developed with GitLab Git-repository manager for source code hosting and using issue board to track the progress. Source code is available to the users with permissions at https://gitlab.cs.ttu.ee/artbas/iapb . How to run server and client locally is described in the repository's readme.md file, but database access should be configured due to usage of the IP addresses whitelist for the connections to the database.

Web application is available at http://icon-monitor.s3-website-us-east-1.amazonaws.com/

4.1 Service architecture

Service has basic 3-tier Client-Server Architecture (Figure 9).

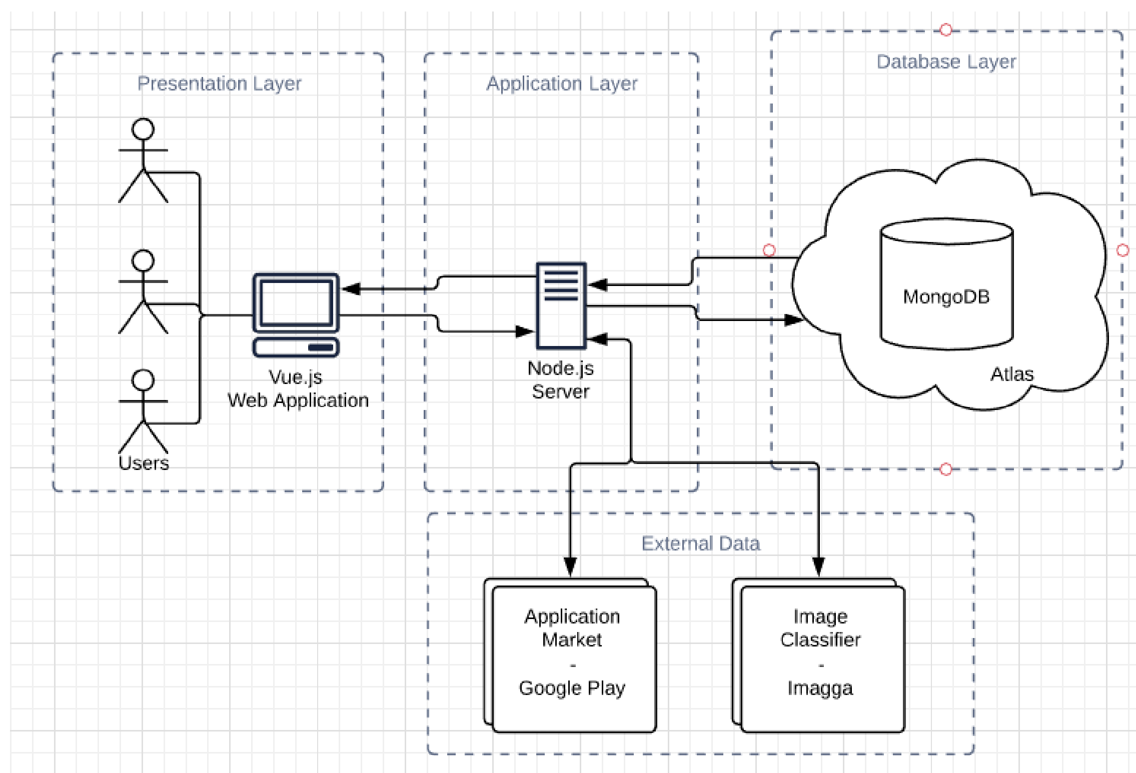

Figure 9 Client-Server Architecture of the Service

It has Presentation Layer of client machine, that hosts web application. users are requesting it via browser and using by user interface. Application Layer is represented by a server, that responds to the client with requested data. Server is able to obtain new data

from the outside, send data for processing to the external API and supervises Database Layer represented by a database server. All communication takes place via HTTP.

Client-side web application is hosted on Amazon S3 (Simple Storage Service), which can be configured to host web page and gives it a public IP address and a domain name. The Vue application is meant to be served by an HTTP server, so basic hosting is sufficient due to accessing its domain name using this protocol.

Server running is more sophisticated. It is running using Amazon EC2 (Elastic Compute Cloud). For this purpose, Linux t2.micro instance was created. Using SSH connection node.js and npm (node package manager) could be installed, source code from git repository downloaded and dependencies installed. With these steps ready node.js server could be started and accessed via instance public IP address, that was configured by AWS. Instance also had a domain name, which is good practice to use so another Amazon internal service could connect directly, as in the case. To constantly run server node package called PM2 was used. It is a process manager to run server process on the background with additional features like logging management.

MongoDB Atlas makes database manual management as easy as possible. Running database instance on one of the three available clouds is just a matter of a few clicks. Author made a decision of using an AWS Cloud due to familiarity and free-tier availability. Database is configured with automated sharding by Atlas, so there are 3 shards located on North America region.

4.2 Server-side

Backend is written using Node.js.

4.2.1 API

Server communication with client is built upon HTTP. First of all, API was implemented on the server so there were endpoints defined (Table 1). Node module Express.js not only helped to configure server on the setup but helps to manage routes for the API. There is a handler mapped to each endpoint, so API knew what to do with the coming request from the client.

Parameters could be sent to some endpoints. Express.js helps in parsing and managing them conveniently too. GET method requests are used where the data is only retrieved, POST method requests are used where data changes are possible, mostly where icons are going to be processed and result will be written into the database.

URL route	Request method	Description
/games	GET	Responds with the list of 100 games lately updated.
/games/filter?	GET	Request has query parameters which are country, ranking, category, tags and colors. Responds with a filtered result of icons according to query parameters.
/games/details/:id	GET	Shows detailed information about the game.
/games/available/tags	GET	Responds with available tags for filtering
/games/available/countries	GET	Responds with available countries for sorting
/games/available/categories	GET	Responds with available categories for sorting
/games/available/rankings	GET	Responds with available rankings for sorting
/games/available/colors	GET	Responds with available colors for filtering
/games/tags/:id	POST	Service endpoint. Updates (sends for processing) tags for the certain game
/games/colors/:id	POST	Service endpoint. Updates (sends for processing) colors for the certain game
/games/a/scrap/?	POST	Service endpoint. Scraps new games by query that includes country, category and ranking.

/games/a/updatetags/?	POST	Service endpoint. Sends for tag processing all the games fitting the query
/games/a/updatecolors	POST	Service endpoint. Sends for color processing all the games fitting the query

Table 1 API endpoints

There are some service endpoints, which means that they are not supposed to be used by user, but they are for development or administration use what '/a' path prefix indicates and should be available only after authentication, for example using 'Basic' authentication in the request header but it was not implemented going out of scope, however should be mentioned for the possible future development.

4.2.2 Obtaining game data

Basically, the whole service is tied up to the data could be obtained from Google Play. As mentioned above there is a npm module called google-play-scraper. It can parse data from Google Play using queries. Best met the functional requirements module method 'list' (Figure 10). It allowed to grab data from all existing Google Play game categories, choose amount of games to grab, set country for the market and grab from internal ranking.

```
scrapGamesByQuery = function(country, ranking, category, amount)
{
    return gplay.list({
        category: category,
        collection: ranking,
        num: amount,
        country: country
    })
}
```

Figure 10 Get data from Google Play by query

The example of data fetched by this module is in json and showed in Figure 11. Response data is an array of json objects. Properties that were actually used are: appId to identify already processed games occurring in different rankings, developer name, game title and URL to Google Play page of the game are used in visualization and most important is the link to the icon. In spite not every field is used it is saved as the whole document with every property from the google-play-scraper response.

```
 [ { url:
'https://play.google.com/store/apps/details?id=com.playappking.b
usrush',
    appId: 'com.playappking.busrush',
    summary: 'Bus Rush is an amazing running game for Android!
Start running now!',
    developer: 'Play App King',
    developerId: '6375024885749937863',
    title: 'Bus Rush',
    icon:
'https://lh3.googleusercontent.com/R6hmyJ6ls6wskk5hHFoW02yEyJpSG
36il4JBkVf-Aojb1q4ZJ9nrGsx6lwsRtnTqfA=w340',
    score: 3.9,
    scoreText: '3.9',
    priceText: 'Free',
    free: false },]
```

Figure 11 Example of json data fetched

4.2.3 Storing game data

For more convenient work with database another node module was used. It is ‘mongoose’. It has some wrapped MongoDB queries wrapped into sole methods (Figure 12). This improves code readability but does not limit MongoDB capabilities – native query structure can be used with mongoose.

```
Tag.find().distinct('tag');
```

Figure 12 mongoose two queries in one-line example

Another mongoose strength is schemes (Figure 13). Scheme defines document structure for the specific collection, where similar documents should be saved.

```
let Game = new Schema({
    appId: {
        type: String
    },
    tags: [{
        name : String,
        confidence : Number
    }],
    country: {
        type: String
    },
    colors: [{
        color_type: String,
        color: String,
        percent: Number,
        code: String
    }]
},{
    collection: 'games'
});
```

Figure 13 Simplified mongoose document scheme for collection 'games'

4.2.4 Processing data

At this point data is scraped and this raw data is saved within scheme. Now icon should be sent to processing to generate tags and to define colours. Flow of sending data for processing is implemented via mentioned earlier application API service endpoints. Hereby, selected game icons are sent to the Machine Learning Imagga API. It is able to set tags for image processed and name image colours.

Imagga API has a specified URL for HTTP request to process image with new tags (Figure 14).

```
https://api.imagga.com/v2/tags?image_url=http://url_to_image
```

Figure 14 Imagga API request URL to process image for tags

Icon image from Google Play is fetched not as a raw data, but URL where it is hosted, and API provided the way to interact with it in this case.

When data is processed it is immediately saved into database and resulting game document is updated with tags array that have resulting structure shown in Figure 15.

```
"tags":[
      {
         "confidence": "38.5370483398438",
         "name": "cartoon"
      },
      {
         "confidence": "28.2365169525146",
         "name": "art"
      }
   ]
```


Figure 15 Generated tags example

Same principle is used to send image for colour processing and there is another URL for this purpose (Figure 16).

```
https://api.imagga.com/v2/colors?image_url= http://url_to_image
```

Figure 16 Imagga URL for colour processing request

Colour data array structure shown in Figure 17.


```
[{
         "color_type":"foreground_colors",
         "color":"blue",
         "percent": "55.7630310058594",
         "code":"#363b7c"
      }]
```


Figure 17 Colour data saved about image colours

Imagga Machine Learning model is able to identify the main object in the image and relatively to it distinguish fore- and background. Thereby it is labelling colours as foreground, background and image colours (whole image; background and foreground together). It depends how percentage colour filling is assigned. For example, background could be 100% black, but if it occupies only 20% of image (main object occupies 80% of image), then percentage field for black will be 20% if there is no more black colour anywhere but on the background.

Colours naming is generalized insofar as will be observed by human and wide range of shades will confuse user. Any kind of shades naming is reduced to the closest more common colour in pallet (Figure 10). Although colour hex code has a shade value, so user will get more specified source image data.

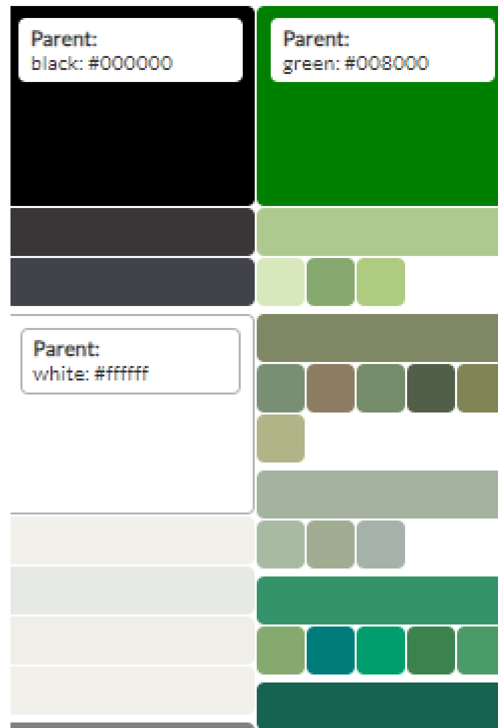

Figure 18 Closest palette colour is determined for shades

4.2.5 Filtering data

Accomplishing the previous steps allows to finally work with data and filter games by icon image metadata.

There are 100 games in every category and three ways to filter them. Filter by tags, by colours and using both. Filtering is working the way if game has no matching element from the user query it is not displaying.

Imagga had fairly specific image to produce tags, thence it generated immense amount of them which is misleading and can not be presented to user as is. Tags had to be configured. First thing what was done to them was that tag assignment confidence by machine, which is one of the output parameters, threshold was set to minimum of 15%. This step lowered tag assignment flaws. Secondly, most common tags were manually combined into groups (Figure 19). These groups are shown to the user in UI and user is choosing from them. If there is any tag match at the game document, this game will be added to the results that will be sent as a response to the client.

```
[
    { group: 'Shape', tags: ['3d', 'circle', 'shape', 'square',
'silhouette', 'template', 'round', 'sphere', 'ball', 'cube', 'box'] },
    { group: 'Bright', tags: ['shiny', 'glossy', 'light', 'bright',
'glow', 'glowing', 'illuminated', 'lights'] },
    { group: 'Happy', tags: ['happy', 'smile', 'fun', 'happiness',
'celebrating', 'cute', 'vibrant', 'confetti', 'cheerful' , 'smiling',
'celebration', 'joy'] },
    { group: 'Person', tags: ['person', 'people', 'male', 'character',
'human', 'silhouette', 'man', 'child', 'boy', 'children', 'baby',
'kid', 'boy', 'adult'] },
    { group: 'Animal', tags: ['animal', 'character', 'bear', 'bird',
'pig', 'animals', 'teddy'] },
    { group: 'Building', tags: ['building', 'house', 'home',
'character'] },
    { group: 'Outdoors', tags: ['park', 'sky', 'outdoor'] },
    { group: 'Text', tags: ['text'] },
    { group: 'Uniform', tags: ['uniform', 'soldier', 'army', 'war',
'military', ''] },
    { group: 'Metal', tags: ['metal'] }
]
```

Figure 19 Generated tags by Imagga are manually grouped

It is more straightforward with colours, due to their naming is already generalized, as described earlier, and thus has a concise range of options. The only feature is that colour separation by type takes place there (fore-, background, image).

In request body user sends list of items he/she wants to use in filtering and they are used to get appropriate items from database. MongoDB has a very convenient way of aggregation (Figure 20). A pipeline pattern is applied there and every step in a search query takes only suitable documents, and even fields, if specified, and passes it to the further pipeline step.

```
function listGamesByTagsAndColors(country, ranking, category,
queryTags, queryColors, res) {
    console.log('Listing by Tags and Colors!')
    Game.aggregate([
        {
            $unwind: "$colors"
        },
        {
            $match: {
                country: country,
                ranking: ranking,
                category: category,
                $and: [{'tags.name': {$in: queryTags}},
                    {'tags.confidence': {$gte: 15}}]
            }
        },
            {
                $unwind: "$colors"
            },
            {
                $match: {
                    'colors.color': { $in:
queryColors.colorNames},
                    'colors.color_type': { $in:
queryColors.colorTypes}
                }
            },
            {
                $sort: { 'colors.percent': -1 }}
        ]
    );
}
```

Figure 20 MongoDB pipeline pattern in quering example

4.2.6 File structure

Server file structure (Figure 21) is divided for the following parts. Configs are modules that exports constants, static data, for example database URL. Handlers are scripts with main API logic, they include methods that process user requests and form responses as well as catching and handling errors. Models are schemes for different collections where json objects are stored in MongoDB. Models are possible because of mongoose npm module. Routes are paths for API where users send request, with Express.js help all these paths are mapped to the handling methods. Services are entity where module is brought to work, in this case is where google-play-scraper module scraps data from Google Play. Main file is app.js, it is where server starts, set upped and configured, mostly with the express module help.

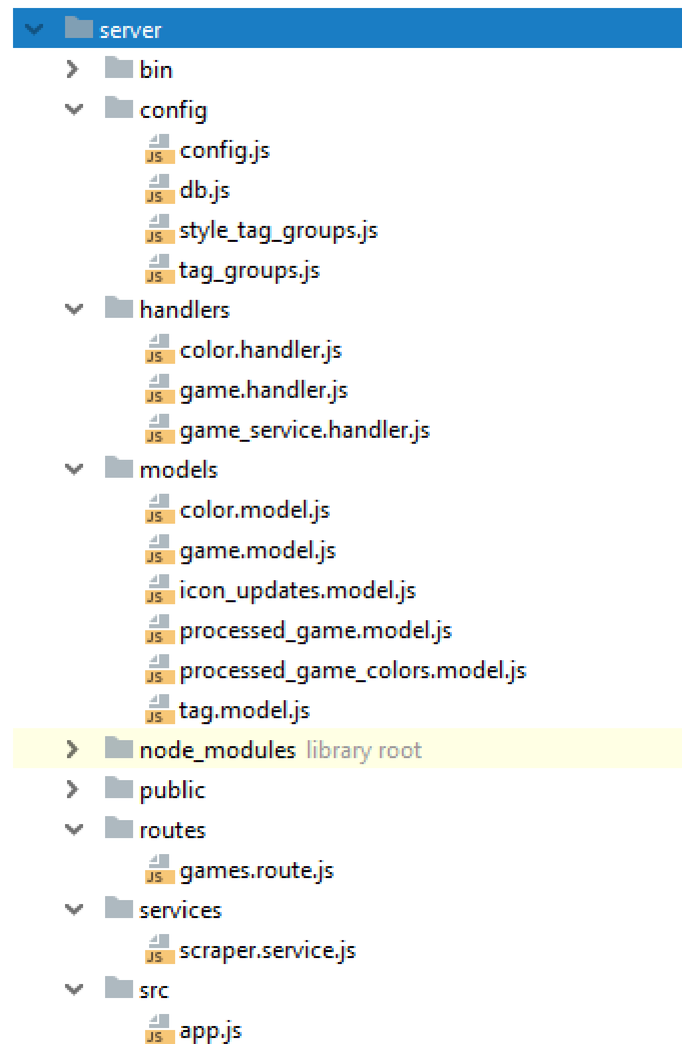

Figure 21 Server project structure

4.3 Client-side

The client-side is represented by a web application implemented using Vue.js framework.

4.3.1 Overall structure

Above all a Vue application has one main component App.js in the case, which will switch with Vue-router to other components and actually show them inside the main component. These components use Axios to perform requests. Axios is promise based HTTP client for the browser and node.js. Structure schematic image is shown at Figure 22.

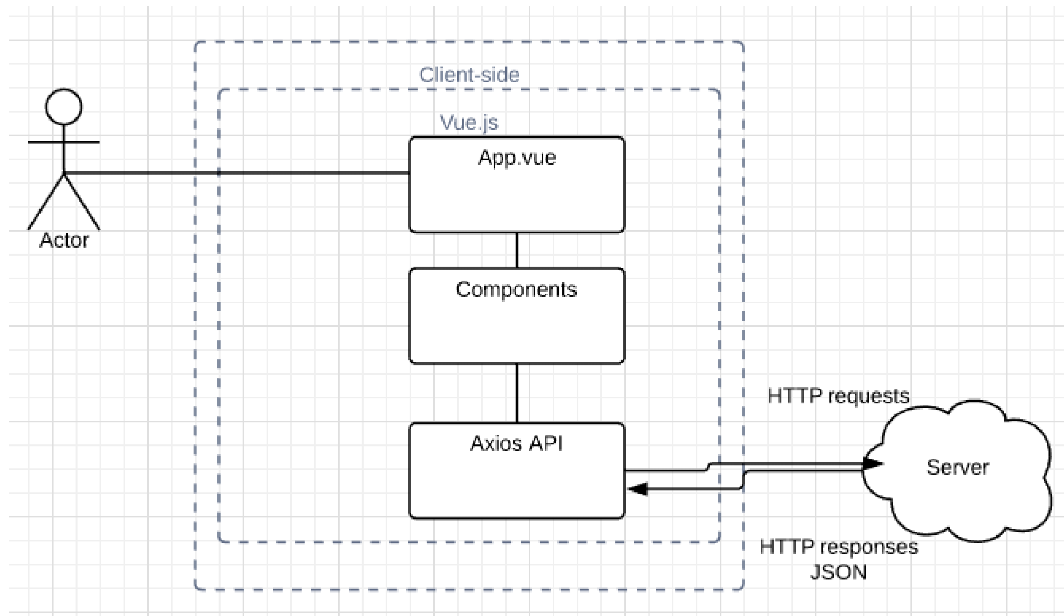

Figure 22 Client overall structure

4.3.2 Components

Navigation bar was introduced into App.vue, i.e. main component, because every other component is wrapped into it, so it is suitable for it perfectly.

Substantially there is a component for each page, however ideally there could be more separation, especially for repeating parts.

Home Component (Figure 23) is a first component user will see, it. It doesn't have much meaning but is foreseen for the future development, so it should show latest icon updates.

Game Monitor Component (Figure 24) is a main functional component. Here are the filters that user interacts with. Results are also displayed here. Ideally this component could have two inner components. One for filters side-bar, second for game result list.

Game Details Component (Figure 25) serves detailed data about application clicked on. First of all, comes colour data separated by types, then are listed generated art style and tags with their confidence level.

And finally, Admin Games Component (Figure 26) there is no way to get to it other than direct link. It is done for development purposes, not for user use, to have some handy buttons to use API service endpoints. It could be disables as the endpoints for the production client and server.

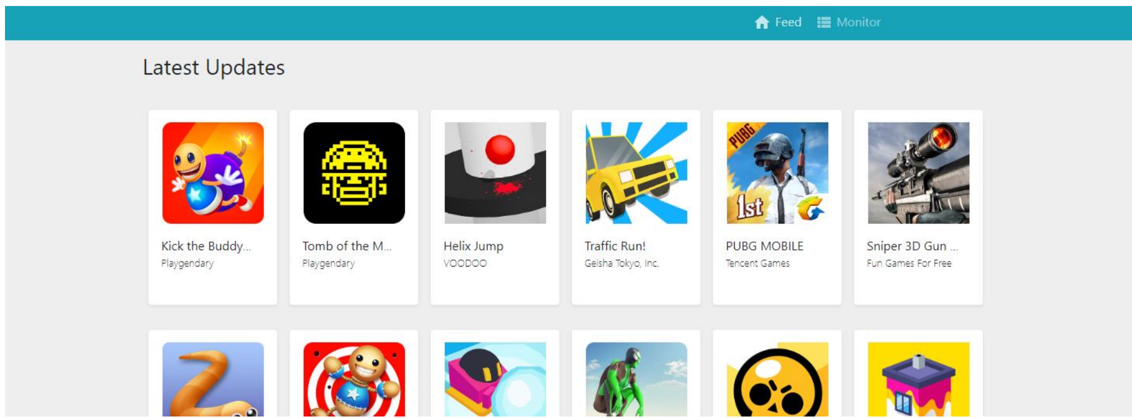

Figure 23 Home Component - Desktop view

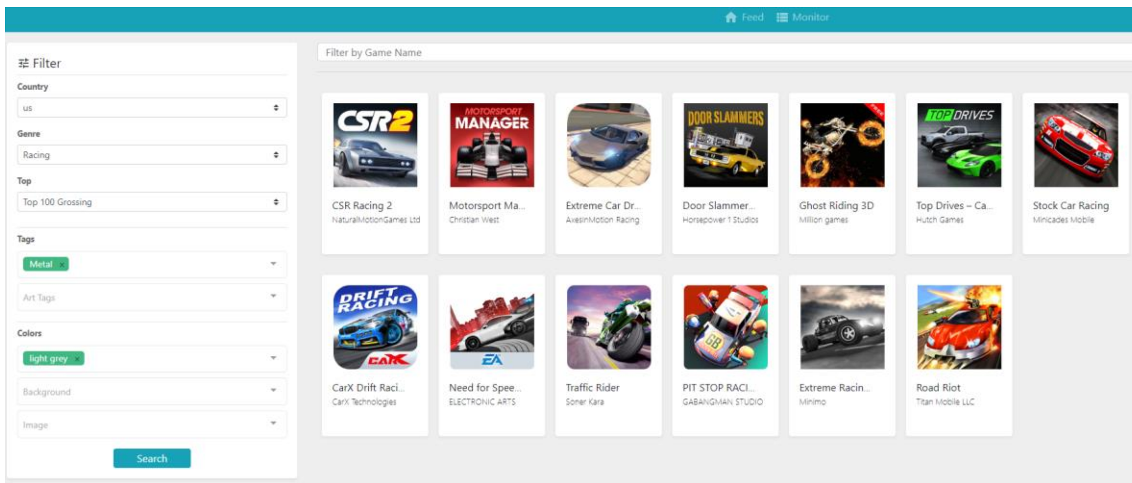

Figure 24 Game Monitor Component - Desktop view

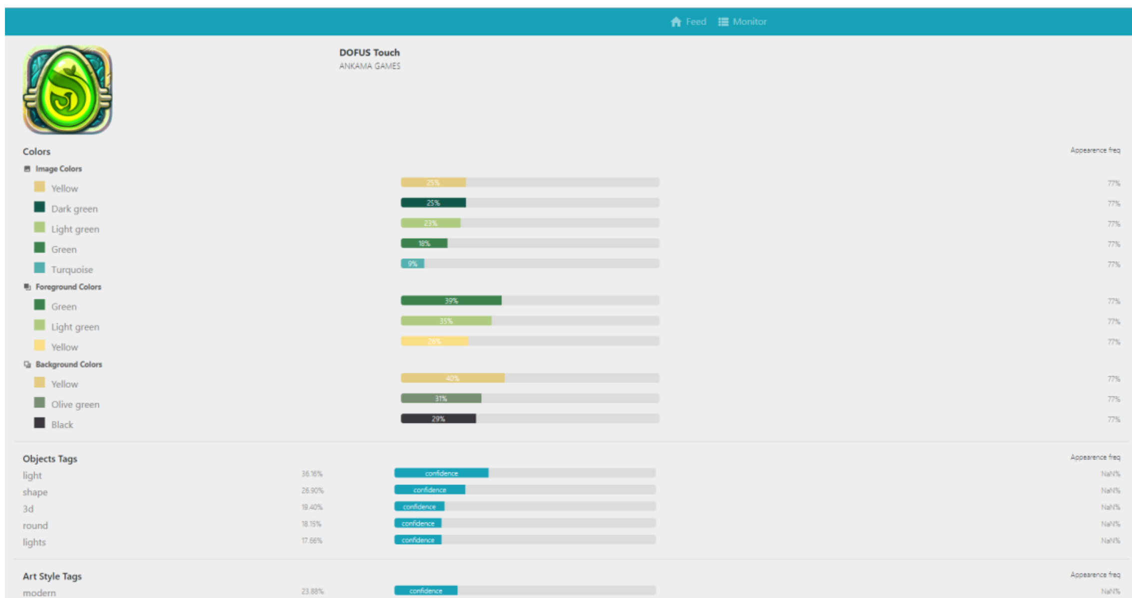

Figure 25 Game Detail Component - Desktop view

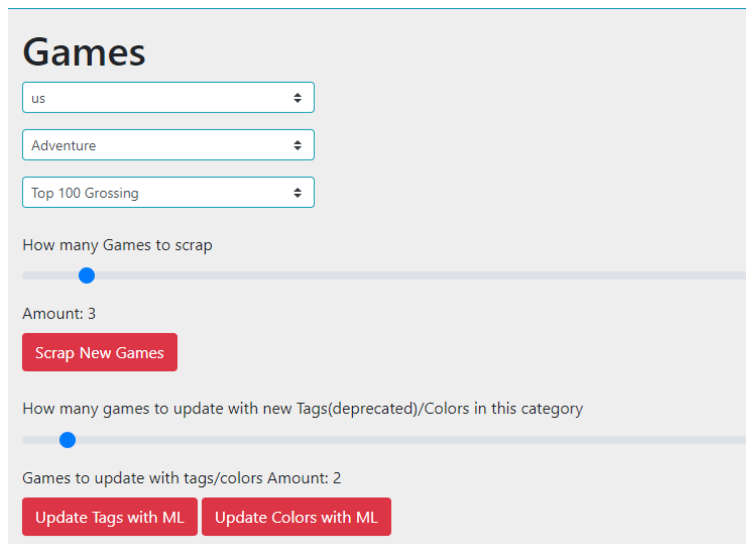

Figure 26 Admin Games Component

4.3.3 User Interface

To achieve representative appearance of the web applicative various external components were used.

Vue-bootstrap is a grid CSS framework with lots of ready UI elements, using which will help to achieve a pleasant look for the web-site. And as an additional advantage makes web application with adaptive-design. Thus, every component looks appropriate for use via mobile device (Figure 27, Figure 28).

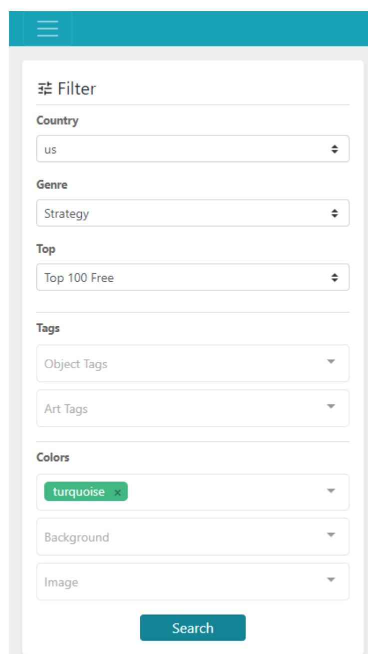

Figure 27 Filter bar at Game Monitor Component - Mobile view

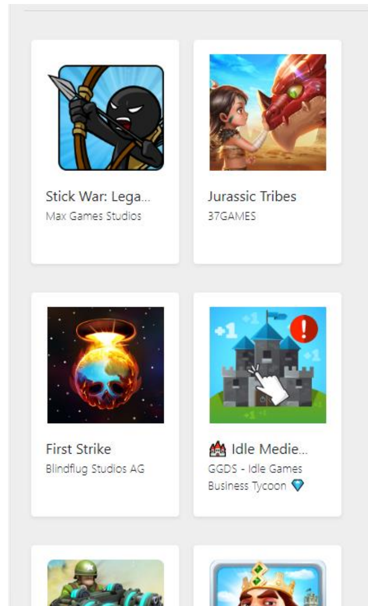

Figure 28 Games result at Game Monitor Component - Mobile view

Vue-multiselect was used to make each filter comfortable as for user as well for development. This component has clear and obvious options data passing for visualization and returning selections made in code.

Vue-material-design-icons were used for different types of icons to make some navigations more intuitive.

4.3.4 File structure

Client file structure (Figure 29) is divided for the following parts. Assets hold static data like images (web-site logo for example) and custom CSS files. Components are for components which are described above. Router keeps all paths to components. Services are for Axios module. It manages HTTP requests to the server. Here is one example at Figure 30. It asks for detail data of selected game from the server. Api() is an Axios class created with server base URL as a property.

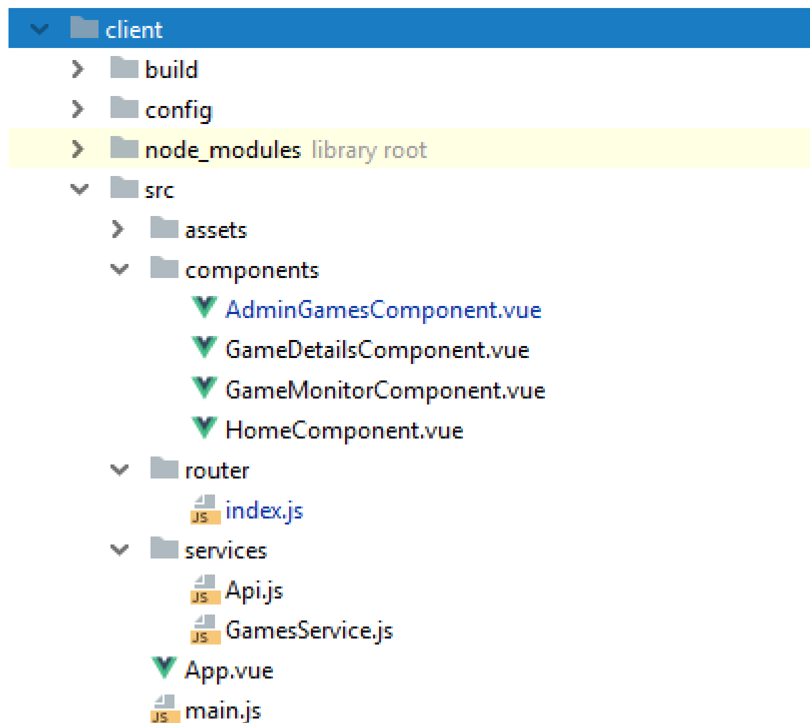

Figure 29 Client project structure

```
fetchGame (id) {
  return Api().get('games/details/' + id)
}
```

Figure 30 Axios get request example

4.4 Service testing

Performance testing is implied by itself due to technology selection. Starting with client, it utilizes Vue.js framework, which is properly tested internally and is famous for it's speed at benchmark tests in comparison with other similar frameworks. Node.js server utilizes Express.js module, which is again well tested and is a thin layer for HTTP requests which is a protocol and designed to be simple and efficient. MongoDB Atlas is automatically configured for the best performance available for that scale. They all are designed to run within high loads. There could be flaws in a user queries, but overall, they are not heavy and does not have excess actions. It is possible to suggest that written queries could be honed for perfection, but specific experience with MongoDB is required.

There is almost no isolated logic on the server, so unit tests are not the matter of concern, but integration tests could take place. Notwithstanding, to write them properly additional

database environment has to be established. Studying new test framework for testing in Node.js is time consuming not to mention staging new test environment, so scope was planned without these. However, for future development and service scaling tests are essential.

4.5 Future development

For the future development first thing which is essential to have is to stage Production, Development and Test environments.

Track icon updates and save icon image history with the data specified. Update checks could be executed daily.

Service could filter by keywords from game description or it's update texts. Tags could be composed from keywords and therefore games could be filtered by features they probably have in game, according to their description.

Features could get weight for the game rank in top list. In this case the service will be of interest for game designers, they could see what features are the most popular and are good to have in the game of the chosen genre.

Another wished feature is to compare games, by icons or other possible features, as keywords from text description.

5 Summary

The aim of this work was to create an internal solution for artists of Creative Mobile OÜ, to mitigate the process of researching and comparing together reference game icons from the application market, Google Play specifically, while creating the brand-new one. In other words, created solution should have to help artists to make icons.

The result is a web application that displays top 100 games of a selected genre and country from Google Play. These games could be filtered by tags and colours, which are defined by machine learning image recognition model.

This solution did not gradually change the process artists create new game icons but is considered to be potentially powerful if future development is applied. Tags that machine was able to generate for icons did not gave sufficient knowledge to be considered as finding out the hidden pattern for the icon images for most genre. Colour filtering is the feature artists payed attention for, but they could not imagine how they could apply it in their work.

The final product is the full-stack application with server written in Node.js and client written on Vue.js. Data is saved in MongoDB. Everything is deployed on AWS.

References

[1] "Machine Learning," [Online]. Available: https://www.coursera.org/learn/machine-learning. [Accessed 14 05 2019].

[2] "ISAM and Single Paged (SPA) Applications," [Online]. Available: https://www.ibm.com/blogs/security-identity-access/isam-and-single-paged-spa-applications/. [Accessed 11 05 2019].

[3] "What Is a Single-Page Application?," [Online]. Available: https://dzone.com/articles/what-is-a-single-page-application. [Accessed 12 05 2019].

[4] "ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET," [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx. [Accessed 10 05 2019].

[5] T. R. a. J. O. Coplien, "The DCI Architecture: A New Vision of Object-Oriented Programming," 20 03 2009. [Online]. Available: https://www.artima.com/articles/dci_vision.html. [Accessed 15 05 2019].

[6] "What Are The Benefits of MVC?," [Online]. Available: http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/. [Accessed 11 05 2019].

[7] J. Gossman, "Introduction to Model/View/ViewModel pattern for building WPF apps," 08 10 2005. [Online]. Available: https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/. [Accessed 17 05 2019].

[8] "Patterns - WPF Apps With The Model-View-ViewModel Design Pattern," [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dd419663.aspx. [Accessed 18 05 2019].

[9] "Introduction," [Online]. Available: https://vuejs.org/v2/guide/. [Accessed 22 04 2019].

[10] "Getting Started," [Online]. Available: https://012.vuejs.org/guide. [Accessed 25 04 2019].

[11] "Comparison with Other Frameworks," [Online]. Available: https://es-vuejs.github.io/vuejs.org/v2/guide/comparison.html. [Accessed 22 04 2019].

[12] "JavaScript Everywhere and the Three Amigos," [Online]. Available: https://www.ibm.com/developerworks/community/blogs/gcuomo/entry/javascript_everywhere_and_the_three_amigos?lang=en. [Accessed 18 05 2019].

[13] "google-play-scraper," [Online]. Available: https://www.npmjs.com/package/google-play-scraper. [Accessed 18 05 2019].

[14] "Express," [Online]. Available: http://expressjs.com/. [Accessed 18 05 2019].

[15] "Mongoose," [Online]. Available: https://www.npmjs.com/package/mongoose. [Accessed 18 05 2019].

[16] "MongoDB," [Online]. Available: https://www.mongodb.com/. [Accessed 18 05 2018].

[17] "MongoDB Atlas," [Online]. Available: https://www.mongodb.com/cloud/atlas. [Accessed 18 05 2018].

[18] "Amazon S3," [Online]. Available: https://aws.amazon.com/s3/. [Accessed 26 04 2019].

[19] "Amazon EC2," [Online]. Available: https://aws.amazon.com/ec2/. [Accessed 26 04 2019].

[20] "Amazon Rekognition," [Online]. Available: https://aws.amazon.com/rekognition/. [Accessed 20 05 2019].

[21] "Vision AI," [Online]. Available: https://cloud.google.com/vision/. [Accessed 20 05 2019].

[22] "Auto-Tagging API," [Online]. Available: https://imagga.com/solutions/auto-tagging.html. [Accessed 20 05 2019].