	School of Computing University of Leeds **Coursework 2 - Report**	**Module Code** **COMP3211**

Web Services Composition
Submission Deadline Date: 17/11/2021

Student Name		**Username**	**ID**
1	Sali Hyusein	sc18s2h	201258162
2	Jasmine Tarbard	Sc18jt	201254554

<u>Composition of Originality (10 marks)</u>

Describe what the Web services composition does

The web service composition allows users to easily navigate and search for latest video game related news, giveaways, reviews and metadata, narrowed down to only the specific genres they select and are interested in. The objective altogether is to provide an easy-to-use and straightforward interface which enhances user-experience in the area of game exploration.

Provide details in the table below

Web Service	**Own or External**	**Input**	**Output**	**Output Parsing / Extract something of interest**
1	**Own**	**A single String parameter specifying the game genre the client is interested in.**	**List of filtered game news or giveaways JSON (determined by the endpoint), matching the input genre.**	**Parses output of external API (Service 3) endpoints before filtering the content matching a client-specified genre.**
2	**Own**	**A plain text input of the game's name/title. Not case sensitive.**	**JSON of the game's reviews and information such as description.**	**Parses a collection of databases to extract rows matching the client-specified game.**
3	**External**	**No inputs; Only correct header and endpoint links are required to access resources.**	**List of game/game news/giveaway objects**	**N/A**

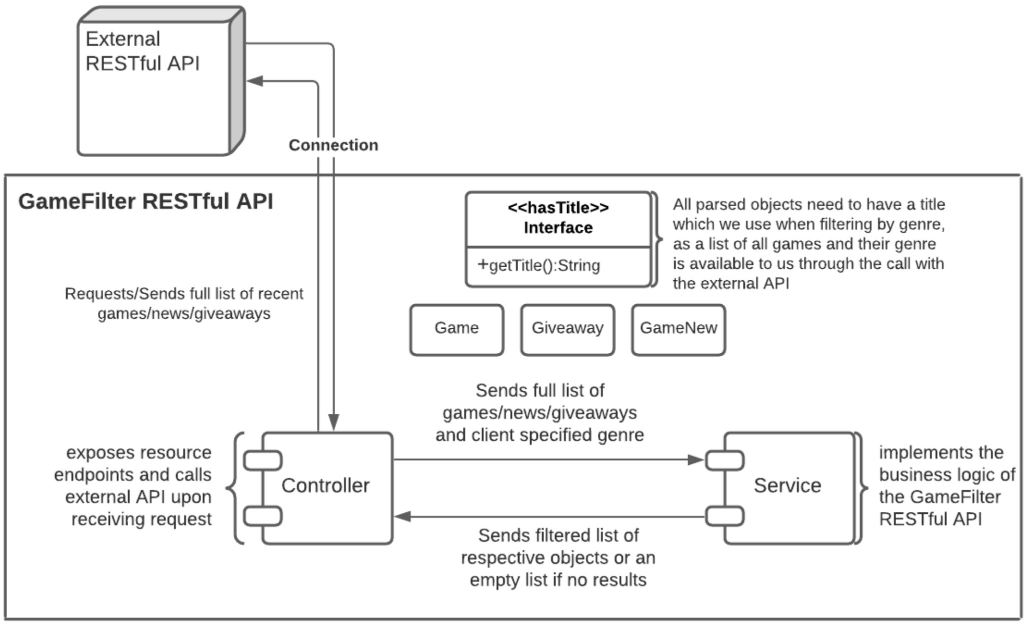
1st Web Service (20 marks)

Name of student in charge: Sali Hyusein

1st Web service	*Fill in this table*
Name of service	**game-filter**
SOAP-based or RESTful	**RESTful**
Brief description	**The service provides clients the ability to get a list of recent news or giveaways matching only a specific genre of interest. The list returned is dynamic as it uses an external service in its logic to get the full list of news or giveaways for video games, before filtration.**

Server design

The UML diagram below describes the main components of the server and their interaction. Namely, the Server constitutes of two main components – a Controller and a Service class. The Controller exposes all RESTful endpoints which conform to standard naming conventions and also establishes connection with the External service upon receiving a request from a client. After successfully querying and getting data from the External API, the Controller sends the full list of games, news or giveaways to the Service class for processing. The Service class then filters out the information and returns a list of only relevant data objects, that belong to one of the three class definitions {Game, Giveaway, GameNew} all conforming to the hasTitle interface. The collection of filtered items is then sent back to the Client who made the call.


External RESTful API
Connection
GameFilter RESTful API
<<hasTitle>> Interface
+getTitle():String
All parsed objects need to have a title which we use when filtering by genre, as a list of all games and their genre is available to us through the call with the external API
Requests/Sends full list of recent games/news/giveaways
Game
Giveaway
GameNew
Sends full list of games/news/giveaways and client specified genre
exposes resource endpoints and calls external API upon receiving request
Controller
Service
implements the business logic of the GameFilter RESTful API
Sends filtered list of respective objects or an empty list if no results

Server implementation

The server is implemented using the framework Spring Boot Web(RESTful) for Java, built and managed using latest version of Apache Maven. The implementation follows standard practices for Model-View-Controller approach and declares highly modular classes with functionality limited to their scope. There is one master class GameFilterApplication which serves as the starting point for the server and wires all other components altogether. The port the server is listening to is defined in application.properties file and is set to 8082. A Controller class is implemented with 3 unique endpoints, all of which are of type GET. The first endpoint does not require any input parameters and returns a list of genres available for querying in the External Service. The second request takes a String parameter specifying a genre. Upon receiving the request, the server first gets the list of all games and news from the External API. It then calls the business logic of our server described in the Service class to filter the news list, given the genre parameter. This is done by cross-matching news titles with game titles of that genre (since news do not have a parameter defining the genre it relates to). Similar implementation follows the third endpoint of the Server with the exception that giveaways are considered instead of news.

Explain how the service is invoked. You may include relevant snippet of source code

Step-by-step instruction on how to run the server:

- Download latest version of Maven from *https://maven.apache.org/download.cgi*
- You can find installation instructions at *https://maven.apache.org/install.html*
- Navigate to the service folder game-filter/
- Execute the following command in command line: **mvn spring-boot:run**

The web service should now be running on localhost:8082

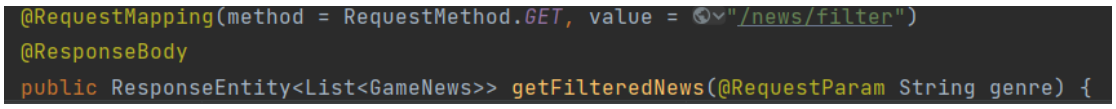
Endpoints:

http://localhost:8082/all/genres
method: GET
Parameters: None
Returns: List<Text> of all available genres for filtration

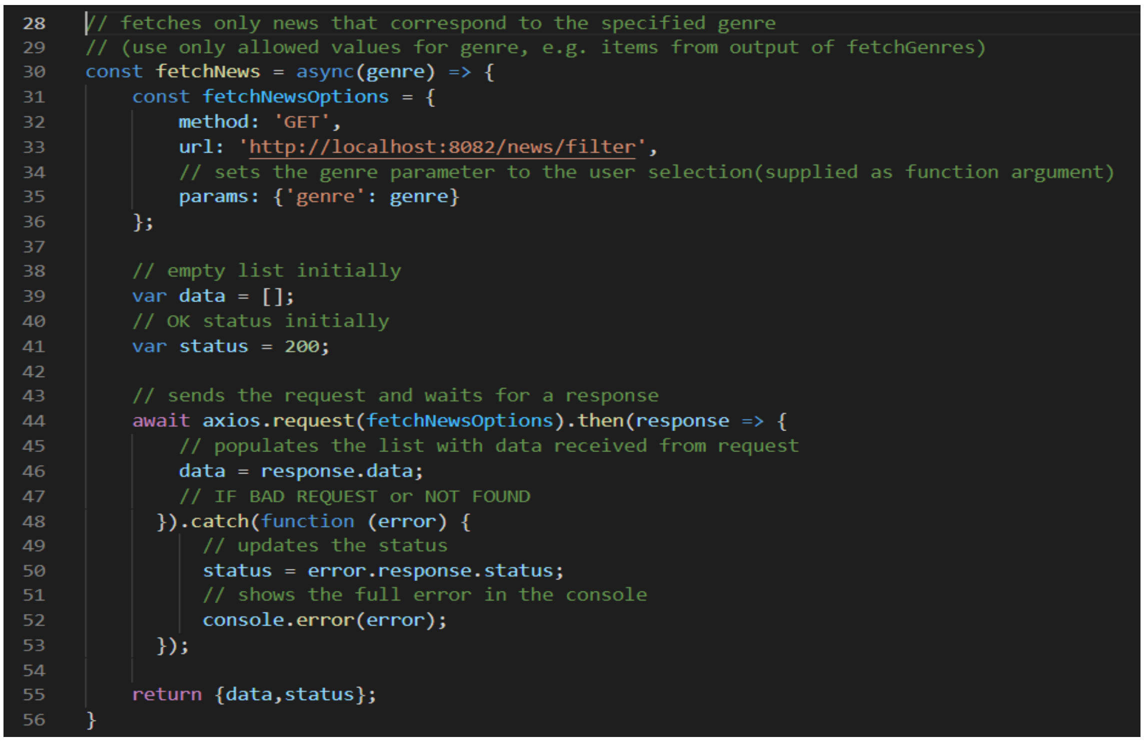
http://localhost:8082/news/filter
method: GET
Parameters: genre <Text>
Returns: List<Object> of latest news on games corresponding to the genre parameter or Empty list if none match the filter.

http://localhost:8082/giveaways/filter
method: GET
Parameters: genre <Text>
Returns: List<Object> of latest giveaways on games corresponding to the genre parameter or empty list if none match the filter.

The first snippet of code below showcases an endpoint definition in Java on the Server side (Spring Boot RESTful) and the second snippet shows how it is accessed using Javascript and Axios.



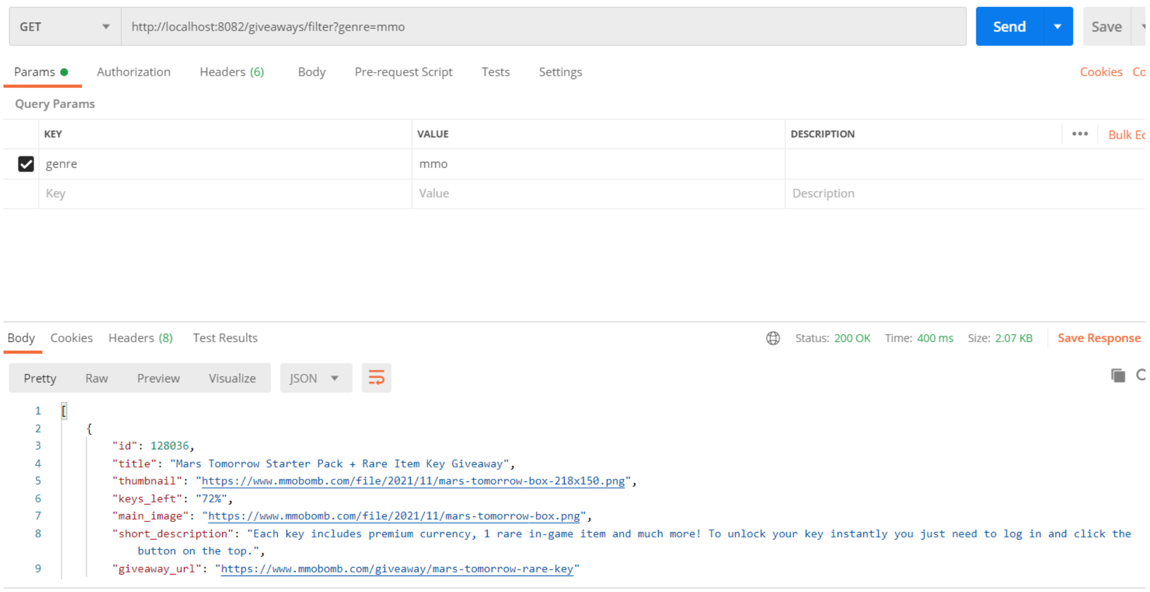
```
@RequestMapping(method = RequestMethod.GET, value = "/news/filter")
@ResponseBody
public ResponseEntity<List<GameNews>> getFilteredNews(@RequestParam String genre) {
```



```javascript
// fetches only news that correspond to the specified genre
// (use only allowed values for genre, e.g. items from output of fetchGenres)
const fetchNews = async(genre) => {
    const fetchNewsOptions = {
        method: 'GET',
        url: 'http://localhost:8082/news/filter',
        // sets the genre parameter to the user selection(supplied as function argument)
        params: {'genre': genre}
    };

    // empty list initially
    var data = [];
    // OK status initially
    var status = 200;

    // sends the request and waits for a response
    await axios.request(fetchNewsOptions).then(response => {
        // populates the list with data received from request
        data = response.data;
        // IF BAD REQUEST or NOT FOUND
      }).catch(function (error) {
          // updates the status
          status = error.response.status;
          // shows the full error in the console
          console.error(error);
      });

    return {data,status};
}
```

Include evidence of its execution through a client, e.g. screen shot

The result of running the GET request to fetch giveaways for games corresponding to the ‘mmo’ genre parameter using POSTMAN serving as the Client side is showcased below.



http://localhost:8082/giveaways/filter?genre=mmo

KEY	VALUE	DESCRIPTION
genre	mmo	
Key	Value	Description

Status: 200 OK Time: 400 ms Size: 2.07 KB

```json
[
    {
        "id": 128036,
        "title": "Mars Tomorrow Starter Pack + Rare Item Key Giveaway",
        "thumbnail": "https://www.mmobomb.com/file/2021/11/mars-tomorrow-box-218x150.png",
        "keys_left": "72%",
        "main_image": "https://www.mmobomb.com/file/2021/11/mars-tomorrow-box.png",
        "short_description": "Each key includes premium currency, 1 rare in-game item and much more! To unlock your key instantly you just need to log in and click the
            button on the top.",
        "giveaway_url": "https://www.mmobomb.com/giveaway/mars-tomorrow-rare-key"
```

Measure the service invocation time. You are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected.

Run No.	Service Invocation time
1	306ms

2	296ms
3	283ms
4	188ms
5	313ms
Average	277.2ms
Standard Deviation	45.727016

Explain how you have obtained these measurements

These measurements are obtained by running the endpoint used as evidence 5 times through POSTMAN which outputs the response time in milliseconds(ms) in addition to the response data and status. The Standard Deviation is calculated using the formula for population:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}}$$

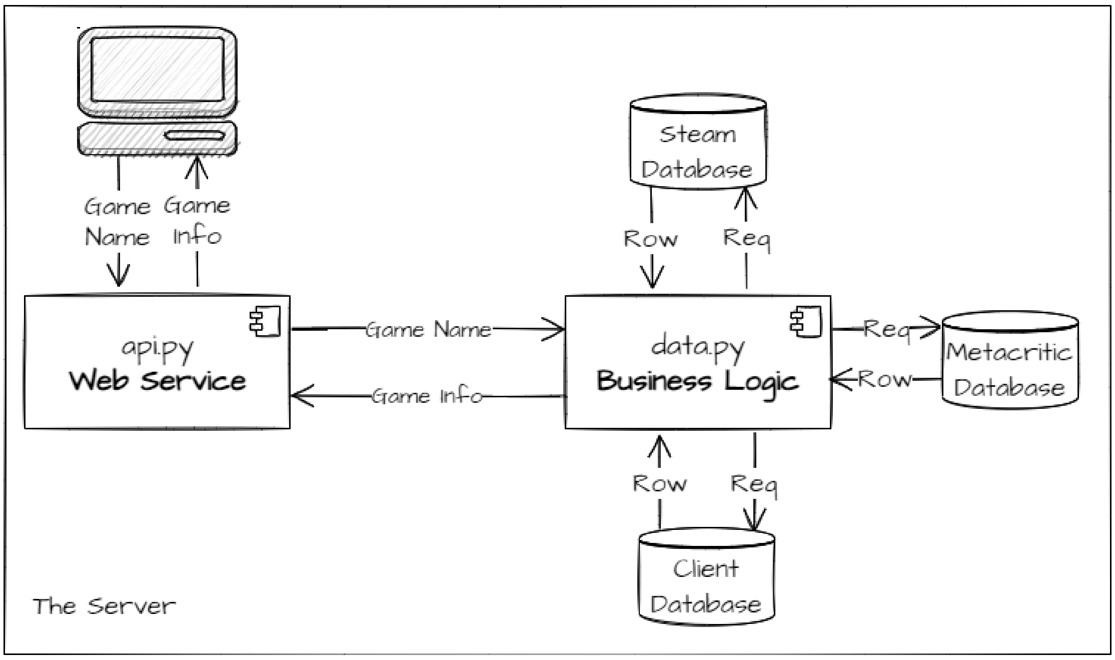
2nd Web Service (20 marks)

Name of student in charge: Jasmine Tarbard

2nd Web service	*Fill in this table*
Name of service	**reviews**
SOAP-based or RESTful	**RESTful**
Brief description	**The reviews web service allows users of our client to create and upload reviews for the games displayed on our client as well as providing access to data from a snapshot of Metacritic and Steam.**

Server design

The server is comprised of two components: the web service which hosts and provides access to the endpoints for the client and the business logic which is responsible for obtaining and processing the data from the databases. There are 3 databases: client_games.csv, metacritic_games.csv and steam_games.csv. The Metacritic and Steam databases provide a large selection of reviews and complimentary information to supplement the sparse client dataset which was created to store reviews sent by the clients.



The Metacritic and Steam databases were obtained from Kaggle and are both held under CC0: Public Domain licenses. I have modified the Steam database by removing some unnecessary columns which was done through Pandas.

Server implementation

The server has been implemented via Python and the libraries Flask and Flask_Restful. The web service component is designed to receive url requests in the format of *localhost:7777/`service`/`game name`* where service is one of steam, metacritic or client. The server returns its data in JSON format, which is natively supported by Flask_Restful.

Explain how the service is invoked. You may include relevant snippet of source code

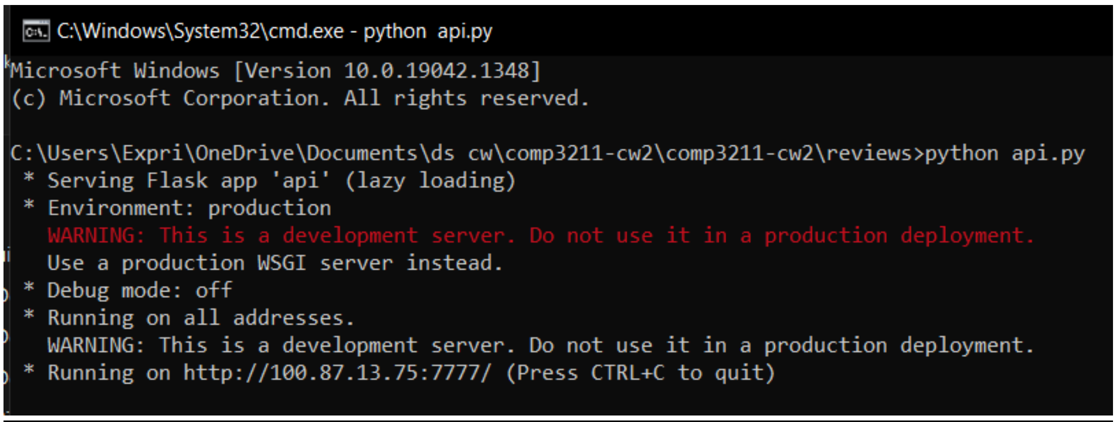
The web service was built in Windows, programmed in Python and uses two libraries: flask and flask_restful. If prerequisites are already installed skip to step 3.

1. Download and install Python 3 - https://www.python.org/downloads/release/python-3100/
2. Run the following commands after Python's installation to install the required libraries:
   a. `pip install flask`
   b. `pip install flask_restful`
3. Navigate to the web services folder: `comp3211-cw2\reviews`
4. Run the following command: `python api.py`

The reviews web service should now be running on localhost with a port of 7777.

http://localhost:7777/client/DOOM

Include evidence of its execution through a client, e.g. screen shot



```
C:\Windows\System32\cmd.exe - python  api.py
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Expri\OneDrive\Documents\ds cw\comp3211-cw2\comp3211-cw2\reviews>python api.py
 * Serving Flask app 'api' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://100.87.13.75:7777/ (Press CTRL+C to quit)
```

Measure the service invocation time. You are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected.

Run No.	**Service Invocation time**
1	524 ms
2	527 ms
3	540 ms
4	522 ms
5	526 ms
Average	527.8 ms
Standard Deviation	40.16

Explain how you have obtained these measurements

These measurements were obtained by sending requests to the server via POSTMAN. The same request was run five times and each time the response time, given by POSTMAN, was documented. The Standard Deviation was calculated using the formula for population.

3rd Web Service - External Service (15 marks)

3rd Web service	*Fill in this table*
Name	**MMO Games API – By MMOBomb!**
SOAP-based or RESTful	**RESTful**
Name of publisher, e.g. Google, Twitter …	**Digiwalls Media**
Brief description	**Programmatic access to the latest multiplayer online games, news and giveaways.**
URL	**https://rapidapi.com/digiwalls/api/mmo-games**

Explain how it is invoked. You may include relevant snippet of source code

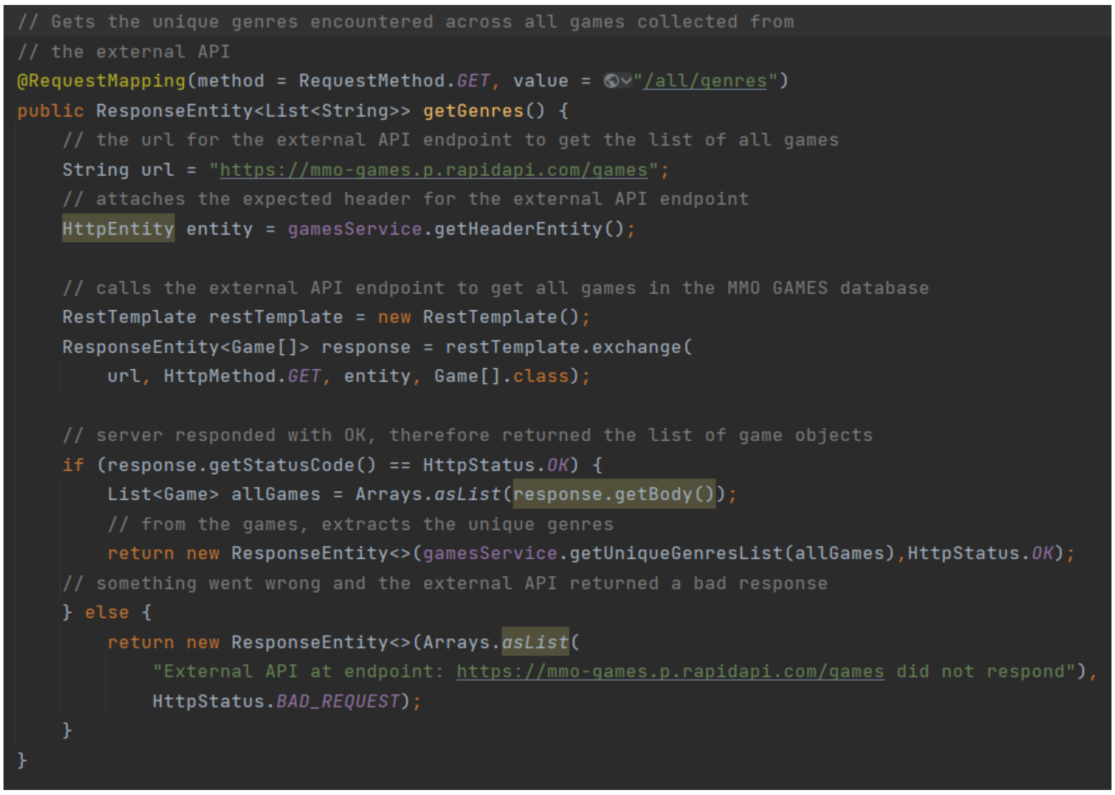
In order to successfully make requests to the External Service the following header key-value pairs must be attached to the header

'x-rapidapi-host': 'mmo-games.p.rapidapi.com',
'x-rapidapi-key': 'daa42aca34msh04b68d62611460dp193a8ejsn93d9f47b24c6'

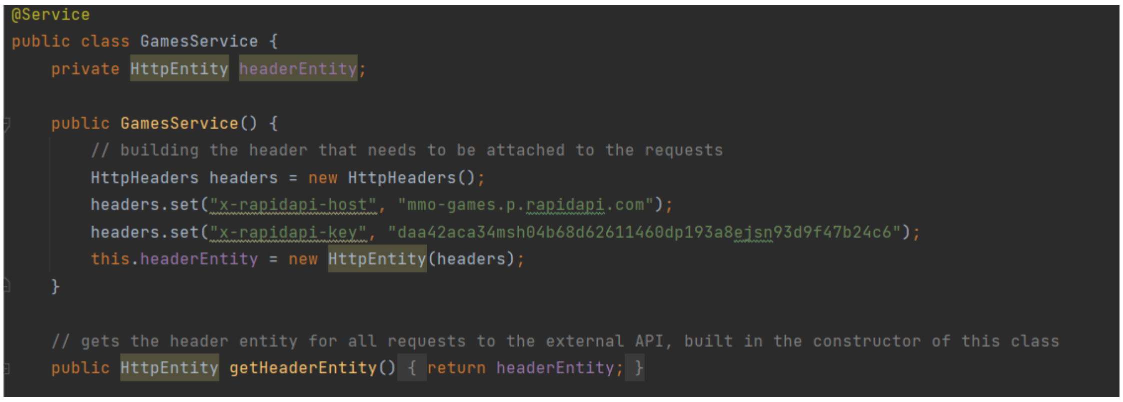
Documentation of all available endpoints can be found at:

https://rapidapi.com/digiwalls/api/mmo-games

Description of how it is invoked in our implementation: Upon a request from a Client, Service 1 invokes the External Service to get the latest list of games, provided by the endpoint: https://mmo-games.p.rapidapi.com/games .Similarly using the endpoint https://mmo-games.p.rapidapi.com/giveaways to get all recent giveaways and https://mmo-games.p.rapidapi.com/latestnew for the news. An example of such invocation is shown in the code snippets below.



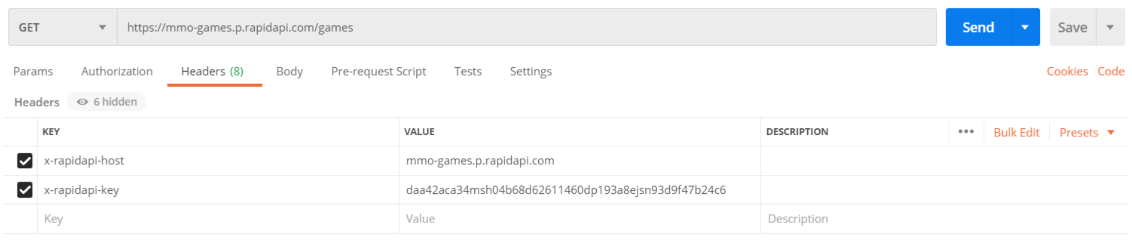
```
// Gets the unique genres encountered across all games collected from
// the external API
@RequestMapping(method = RequestMethod.GET, value = "/all/genres")
public ResponseEntity<List<String>> getGenres() {
    // the url for the external API endpoint to get the list of all games
    String url = "https://mmo-games.p.rapidapi.com/games";
    // attaches the expected header for the external API endpoint
    HttpEntity entity = gamesService.getHeaderEntity();

    // calls the external API endpoint to get all games in the MMO GAMES database
    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<Game[]> response = restTemplate.exchange(
        url, HttpMethod.GET, entity, Game[].class);

    // server responded with OK, therefore returned the list of game objects
    if (response.getStatusCode() == HttpStatus.OK) {
        List<Game> allGames = Arrays.asList(response.getBody());
        // from the games, extracts the unique genres
        return new ResponseEntity<>(gamesService.getUniqueGenresList(allGames),HttpStatus.OK);
    // something went wrong and the external API returned a bad response
    } else {
        return new ResponseEntity<>(Arrays.asList(
            "External API at endpoint: https://mmo-games.p.rapidapi.com/games did not respond"),
            HttpStatus.BAD_REQUEST);
    }
}
```



```java
@Service
public class GamesService {
    private HttpEntity headerEntity;

    public GamesService() {
        // building the header that needs to be attached to the requests
        HttpHeaders headers = new HttpHeaders();
        headers.set("x-rapidapi-host", "mmo-games.p.rapidapi.com");
        headers.set("x-rapidapi-key", "daa42aca34msh04b68d62611460dp193a8ejsn93d9f47b24c6");
        this.headerEntity = new HttpEntity(headers);
    }

    // gets the header entity for all requests to the external API, built in the constructor of this class
    public HttpEntity getHeaderEntity() { return headerEntity; }
```

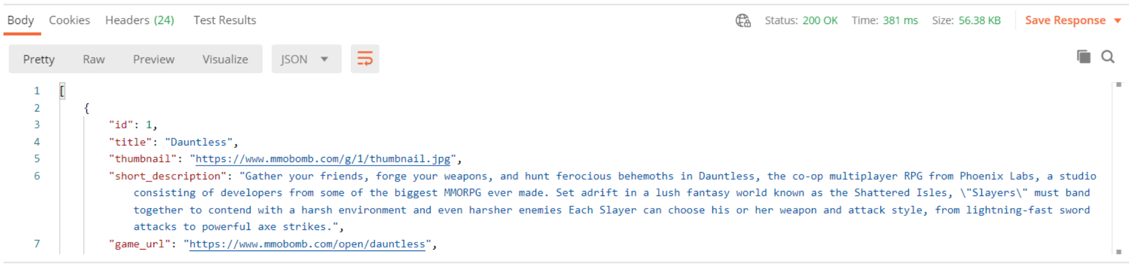
Include evidence of its execution through a client, e.g. screen shot

The result of running the GET request to fetch all games from the External Service using POSTMAN serving as the Client side is showcased below.



GET https://mmo-games.p.rapidapi.com/games

	KEY	VALUE	DESCRIPTION
✓	x-rapidapi-host	mmo-games.p.rapidapi.com	
✓	x-rapidapi-key	daa42aca34msh04b68d62611460dp193a8ejsn93d9f47b24c6	
	Key	Value	Description



Status: 200 OK Time: 381 ms Size: 56.38 KB

```json
[
    {
        "id": 1,
        "title": "Dauntless",
        "thumbnail": "https://www.mmobomb.com/g/1/thumbnail.jpg",
        "short_description": "Gather your friends, forge your weapons, and hunt ferocious behemoths in Dauntless, the co-op multiplayer RPG from Phoenix Labs, a studio
            consisting of developers from some of the biggest MMORPG ever made. Set adrift in a lush fantasy world known as the Shattered Isles, \"Slayers\" must band
            together to contend with a harsh environment and even harsher enemies Each Slayer can choose his or her weapon and attack style, from lightning-fast sword
            attacks to powerful axe strikes.",
        "game_url": "https://www.mmobomb.com/open/dauntless",
```

Measure the service invocation time. You are expected to run the experiments n times (e.g. n = 5). A statistical analysis (average, standard deviation) is expected.

Run No.	**Service Invocation time**
1	381ms
2	130ms
3	109ms
4	106ms
5	112ms
Average	167.6ms
Standard Deviation	107.02635

Explain how you have obtained these measurements

These measurements were obtained by sending requests to the server via POSTMAN. The same request was run five times and each time the response time, given by POSTMAN, was documented. The Standard Deviation was calculated using the formula for population.

Web Services Integration (10 marks)

Provide details of the Web services integration

HTML, CSS, JavaScript and underlying libraries are used to build the Presentation Layer which serves as the Client for all three services, thus bringing together three independent components into a single coherent and transparent system.

This is done in the following way:

List of available genres is extracted and displayed in the landing page, using Service 1 (which in turn uses Service 3 to get list of games and their genres).

Upon clicking on a specific genre, the user selection is saved in session and can be used at any point to request filtered news and giveaways matching the filter. The appropriate endpoints are invoked when the user selects the wildcard corresponding to their point of interest. The response is then rendered on the user's screen, possibly with an error message if the services responded with a different status than OK.

Additionally, users have the ability to see or add reviews to games (filtered to the genre selection) of their choice. Clicking on a game invokes Service 2 and the corresponding response is rendered on the screen.

At any point in time the user can choose to select a different genre and/or observe a different aspect of the application in a transparent way, without having to know that there are in fact 3 services used as resource, how they are invoked or in what order. Therefore, achieving the purpose of modern Distributed Systems.

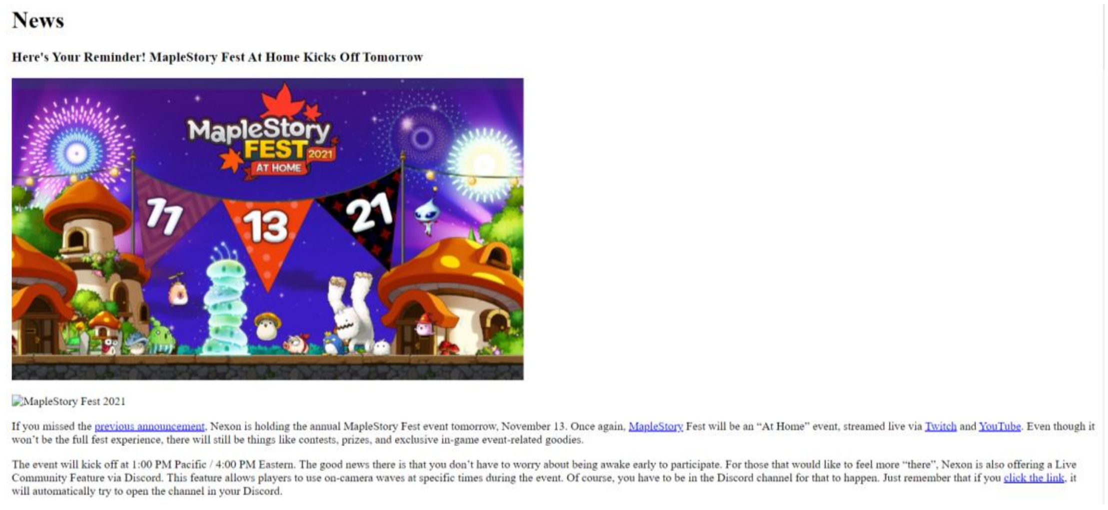
Web User Interface (10 marks) – if attempted

Provide details of your Web-based application (Servlets/JSP/Other Frameworks)

The services are invoked using a promised-based HTTP Client 'Axios' for JavaScript runtime. The request options are defined as JavaScript variable and sent to the server through an Axios call. When parameters need to be specified from the user, in our case the genre, this is done by an event handler function, which registers the user's selection through a click, assigns the respective genre parameter to the request options and performs the axios call to the relevant RESTful endpoint.

Successful Execution (10 marks)

Include evidence of the Web services integration execution, e.g. screen shot



News

Here's Your Reminder! MapleStory Fest At Home Kicks Off Tomorrow

MapleStory Fest 2021

If you missed the previous announcement, Nexon is holding the annual MapleStory Fest event tomorrow, November 13. Once again, MapleStory Fest will be an "At Home" event, streamed live via Twitch and YouTube. Even though it won't be the full fest experience, there will still be things like contests, prizes, and exclusive in-game event-related goodies.

The event will kick off at 1:00 PM Pacific / 4:00 PM Eastern. The good news there is that you don't have to worry about being awake early to participate. For those that would like to feel more "there", Nexon is also offering a Live Community Feature via Discord. This feature allows players to use on-camera waves at specific times during the event. Of course, you have to be in the Discord channel for that to happen. Just remember that if you click the link, it will automatically try to open the channel in your Discord.

Code Submission

The assumption is that your code has been submitted on Minerva.
If you have made your code available on Git instead, please provide details here:

Video Submission

The assumption is that your video demonstrating your integrated client has been submitted on Minerva.
If you have made your video available elsewhere, e.g. on YouTube, please provide details here: