# Project 4 – Wizard World App

**by Xiao Li (AndrewID: xiaol4)**

## Description:

My application allows users to enter a search keyword (e.g., Expelliarmus, Charm, Fire) in Harry Potter and retrieves spell information from the WizardWorld API.

Here is how my application meets the task requirements

## 1. Implement a native Android application

The name of my native Android application project in Android Studio is: Project4Android

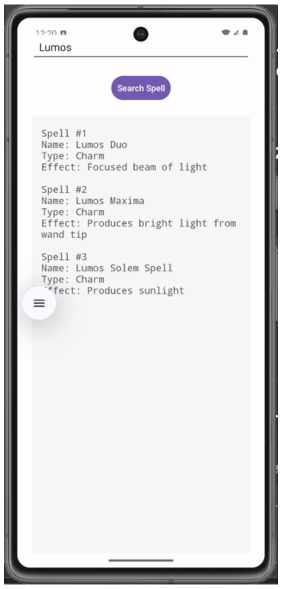
### a. Has at least three different kinds of views in your Layout (TextView, EditText, ImageView, etc.)

My application uses TextView, EditText, Button, and ImageView in its main layout.

These views are organized inside a LinearLayout to create a simple and responsive interface for entering a spell keyword and displaying the result.

All view definitions can be found in content_main.xml, where the TextView provides instructions, the EditText captures user input, the Button triggers the query, and the ImageView (or TextView, depending on your UI) displays the returned spell information.

Here is a screenshot of the layout before the picture has been fetched.



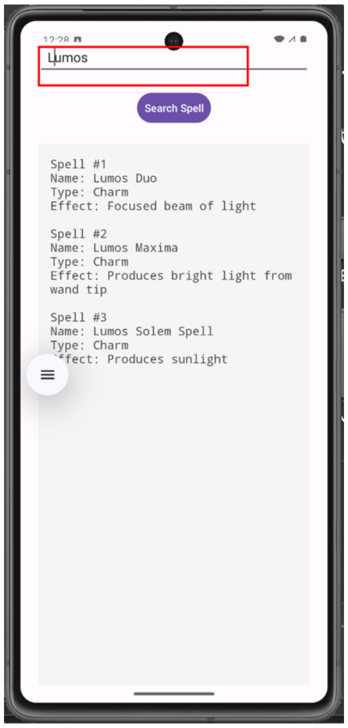
```
Spell #1
Name: Lumos Duo
Type: Charm
Effect: Focused beam of light

Spell #2
Name: Lumos Maxima
Type: Charm
Effect: Produces bright light from
wand tip

Spell #3
Name: Lumos Solem Spell
Type: Charm
ffect: Produces sunlight
```

## b. Requires input from the user

My application requires the user to enter a spell keyword (for example, fire, charm, or a specific spell name). The user types the keyword into the EditText field and presses the Submit button to trigger the query.



```
Spell #1
Name: Lumos Duo
Type: Charm
Effect: Focused beam of light

Spell #2
Name: Lumos Maxima
Type: Charm
Effect: Produces bright light from
wand tip

Spell #3
Name: Lumos Solem Spell
Type: Charm
Effect: Produces sunlight
```

## c. Makes an HTTP request (using an appropriate HTTP method) to your web service

My application performs an HTTP GET request in SpellFetcher.java.

The HTTP request is: " https://<codespace url>/spell?keyword=" + keyword where keyword is the user's search term.

The searchSpell() method sends this request to my web application, waits for the JSON response, parses the returned spell information (such as name, type, and effect), and then updates the UI with the retrieved data.

## d. Receives and parses an XML or JSON formatted reply from the web service

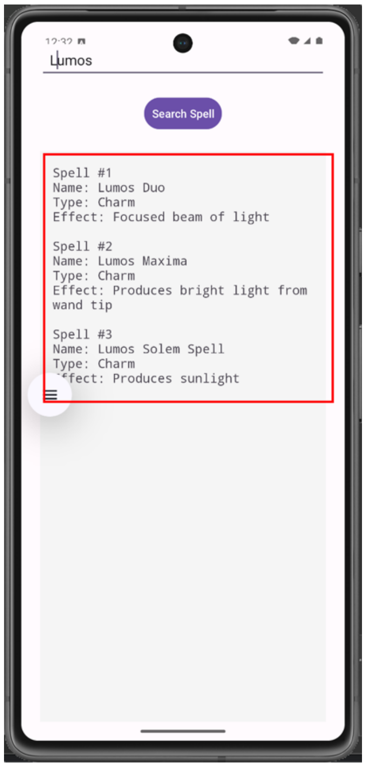
My application receives a JSON-formatted reply from my web service containing the spell information requested by the user. The response includes fields such as the spell's name, type, and effect.

An example of the JSON reply is:

```
[
  {
    "name": "Lumos Duo",
    "type": "Charm",
    "effect": "Focused beam of light"
  },
  {
    "name": "Lumos Maxima",
    "type": "Charm",
    "effect": "Produces bright light from wand tip"
  },
  {
    "name": "Lumos Solem Spell",
    "type": "Charm",
    "effect": "Produces sunlight"
  }
]
```

e. Displays new information to the user

Here is the screen shot after the picture has been returned.



```
Spell #1
Name: Lumos Duo
Type: Charm
Effect: Focused beam of light

Spell #2
Name: Lumos Maxima
Type: Charm
Effect: Produces bright light from
wand tip

Spell #3
Name: Lumos Solem Spell
Type: Charm
ffect: Produces sunlight
```

2. Implement a web application, deployed to Heroku

My web application is deployed and running inside GitHub Codespaces, which provides a cloud-based development and hosting environment. The web service is hosted on an Apache Tomcat server running within my Codespace container.

The URL of my web service deployed in GitHub Codespaces is:

https://crispy-trout-jj4xqw67qwqw3gvq-8080.app.github.dev/

https://crispy-trout-jj4xqw67qwqw3gvq-8080.app.github.dev/spell?name={your_words}

https://crispy-trout-jj4xqw67qwqw3gvq-8080.app.github.dev/dashboard

## a. Using an HttpServlet to implement a simple (can be a single path) API

In my web application project, I follow the standard Model–View–Controller (MVC) structure required by the assignment. My API is implemented as a simple HTTP GET endpoint exposed through a servlet.

- **Model:** WizardWorldAPIClient.java and SpellParser.java

These classes handle the business logic, including making the external API call to the WizardWorld API and parsing the JSON response into Java objects.

- **View:** dashboard.jsp

This JSP page displays the stored logs and operational analytics.

- **Controller:** SpellServlet.java

This servlet receives the incoming HTTP request from the Android client, calls the model classes to fetch and parse the spell information, and returns a JSON-formatted response to the mobile application.

The servlet path is defined in web.xml and mapped to a single endpoint (for example, /spell), which the Android application uses to request spell data.

## b. Receives an HTTP request from the native Android application

SpellServlet.java receives the HTTP GET request sent by the Android application.

The request includes the query parameter “keyword”, which contains the user’s search term.

The servlet extracts this parameter and passes it to the model classes (WizardWorldAPIClient and SpellParser) to retrieve and process the spell information.

## c. Executes business logic appropriate to your application

The business logic is implemented in the model classes WizardWorldAPIClient.java and SpellParser.java.

WizardWorldAPIClient makes an HTTP request to the WizardWorld API using a URL of the form the original api https://wizard-world-api.herokuapp.com/swagger/index.html

The JSON response returned by the external API is then passed to SpellParser.java, which extracts the spell details needed by the application, including the spell’s name, type, and effect. These parsed values are returned to the servlet so that the Android application can display them to the user.

d. Replies to the Android application with an XML or JSON formatted response.

My application replies to the Android client using a simple JSON structure.

Instead of using a JSP to format XML, my servlet (SpellServlet.java) directly writes a JSON-formatted response to the output stream.

An example of the JSON response returned by my web service is:

```
[
 {
  "name": "Lumos Duo",
  "type": "Charm",
  "effect": "Focused beam of light"
 },
 {
  "name": "Lumos Maxima",
  "type": "Charm",
  "effect": "Produces bright light from wand tip"
 },
 {
  "name": "Lumos Solem Spell",
  "type": "Charm",
  "effect": "Produces sunlight"
 }
]
```

This JSON format is generated in SpellServlet.java after retrieving and parsing the spell information from the WizardWorld API.

The Android application receives this JSON and displays the spell results to the user.

**To document the rest of the requirements:**

3. Handle error conditions - Does not need to be documented.
4. Log useful information - Itemize what information you log and why you chose it.

My web application records detailed information for every request processed by the SpellServlet.

These logs are stored using the LogEntry and LogDAO classes and displayed on the web-based dashboard.

The following fields are logged:

- **Timestamp**

Records the exact time each request was received.

Useful for ordering logs and analyzing traffic patterns over time.

- **Input keyword**

Stores the user's search term (e.g., "Lumos").

This allows the dashboard to compute "Top 10 Searched Spells" and understand user behavior.

- **Type and Incantation parameters (if provided)**

These optional search parameters allow analysis of how users filter spell results beyond the basic name keyword.

- **Found flag (true/false)**

Indicates whether the WizardWorld API returned at least one valid spell for the given query.

Useful for monitoring search effectiveness and identifying common failed queries.

- **Latency (in milliseconds)**

Measures total processing time for the request, including external API latency.

This value is aggregated to compute the "Average API Latency" displayed on the dashboard.

- **API URL called**

Logs the exact WizardWorld API request generated by the backend.

This helps with debugging, verifying parameter formatting, and diagnosing mismatches between input and API call.

- **HTTP status code returned from WizardWorld API**

Allows monitoring API reliability and quickly identifying when the external service is failing.

These logs are later aggregated and presented in the dashboard to show metrics such as the most frequently searched spells, API performance, and recent activity.

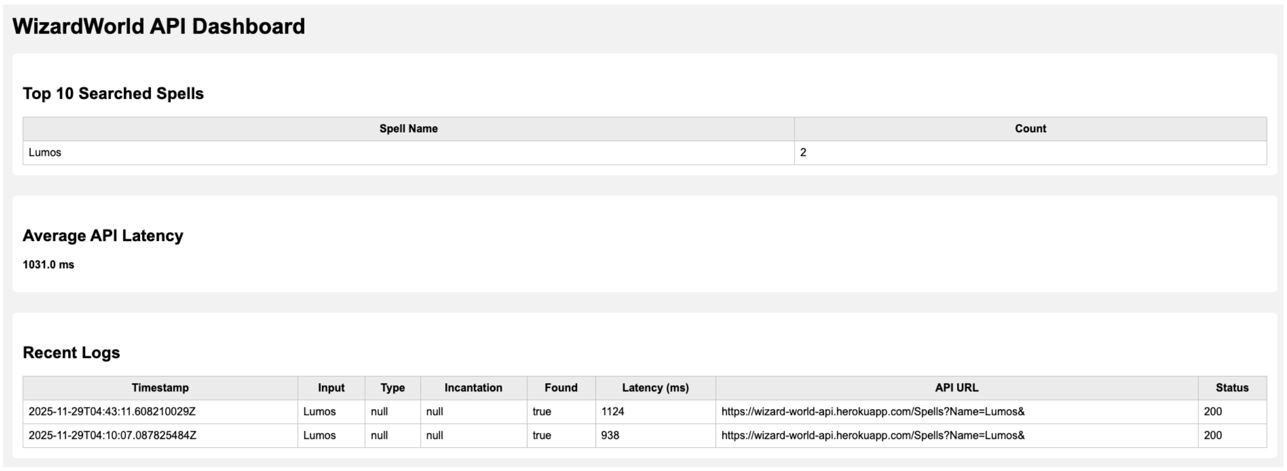
5. Store the log information in a database - Give your Atlas connection string with the three shards

The application uses the LogDAO class to insert each LogEntry into a collection named logs inside my Atlas cluster.

My MongoDB Atlas connection string is (I still use the srv connection, it works well on my codespace):

mongodb+srv://xiaol4_db_user:1CvEvAdAPIeTqBRK@wizardworld.p6rgfx6.mongodb.net/?appName=WizardWorld

6. Display operations analytics and full logs on a web-based dashboard - Provide a screen shot.



**WizardWorld API Dashboard**

**Top 10 Searched Spells**

| Spell Name | Count |
|---|---|
| Lumos | 2 |

**Average API Latency**

1031.0 ms

**Recent Logs**

| Timestamp | Input | Type | Incantation | Found | Latency (ms) | API URL | Status |
|---|---|---|---|---|---|---|---|
| 2025-11-29T04:43:11.608210029Z | Lumos | null | null | true | 1124 | https://wizard-world-api.herokuapp.com/Spells?Name=Lumos& | 200 |
| 2025-11-29T04:10:07.087825484Z | Lumos | null | null | true | 938 | https://wizard-world-api.herokuapp.com/Spells?Name=Lumos& | 200 |

My web application includes a web-based dashboard (dashboard.jsp) that displays operational analytics derived from the log data stored in MongoDB Atlas. The dashboard presents three main sections:

- Top 10 Searched Spells

This table shows the most frequently searched spell names along with their corresponding query counts.

In the example shown, “Lumos” appears as the most common search term.

- Average API Latency

This section displays the mean round-trip time (in milliseconds) for calls made to the external WizardWorld API, allowing me to monitor backend performance.

- Recent Logs

This table lists the most recent log entries, including the timestamp, input parameters, parsing results, latency, API URL invoked, and the returned HTTP status code.

These logs provide full visibility into the behavior of the system and assist with debugging and performance monitoring.