Distributed Systems for Information Systems Management
Name: Santiago Bolaños Vega
Email: sbolaosv@andrew.cmu.edu
Date: November 23, 2024

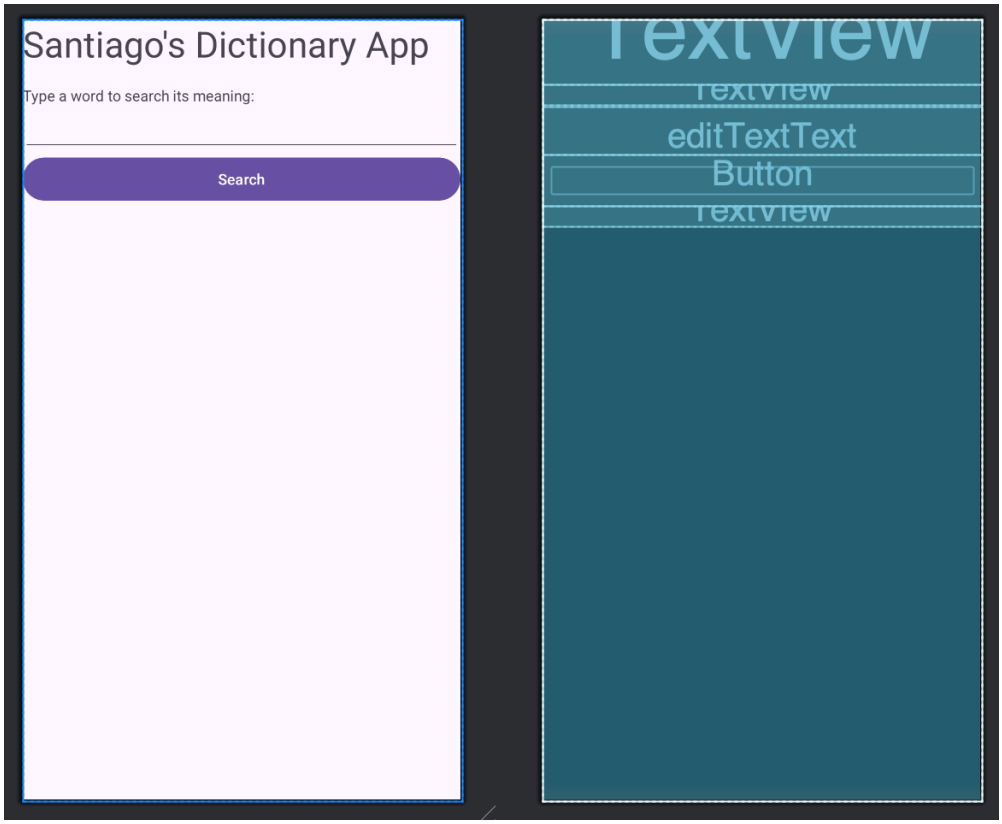
Project 4 Task 2 – Distributed Application Requirements

1. Implement a Native Android Application

The name of my native Android application project in Android Studio is: DictionaryApp. My mobile app will be a pocket dictionary. It will prompt the user for a string and then use the API to return the definition(s) associated with it.

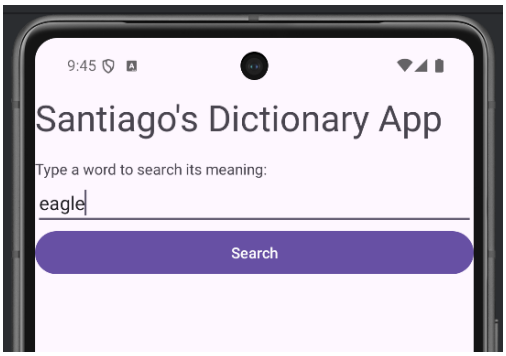
a. **Has at least three different kinds of Views in your Layout (TextView, EditText, ImageView, or anything that extends android.view.View).**

My application uses TextView, Edit Text, and Button. See content_main.xml for details of how they are incorporated into the LinearLayout. Please see a screenshot of the layout.



b. Requires input from the user

Here is a screenshot of the user searching for the definition of an eagle.



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

My application does an HTTP GET request in GetWord.java. The HTTP request is: https://fluffy-lamp-wrgrvwvxxv5xf5vjq-8080.app.github.dev/word/ + search_word where search_word is the user's search term.

The search method makes this request to my Web Application, parses the returned JSON, and returns de possible definitions of the word.

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

An example if the JSON reply is:

{ status": "ok",
"body":
"{\"word\":\"eagle\",\"phonetic\":\"/ˈiːgəl/\",\"meanings\":[{\"partOfSpeech\":\"noun\",\"definition\":\"Any of several large carnivorous and carrion-eating birds in the family Accipitridae, having a powerful hooked bill and keen vision.\"},{\"partOfSpeech\":\"noun\",\"definition\":\"A gold coin with a face value of ten dollars, formerly used in the United States.\"},{\"partOfSpeech\":\"noun\",\"definition\":\"A 13th-century coin minted in Europe and circulated in England as a debased sterling silver penny, outlawed under Edward I.\"},{\"partOfSpeech\":\"noun\",\"definition\":\"A score of two under par for a hole.\"},{\"partOfSpeech\":\"verb\",\"definition\":\"To score an eagle.\"}]}"
}

e. Displays new information to the user

Here is the screenshot after the possible definitions have been returned.

f. Is repeatable (I.e. the user can repeatedly reuse the application without restarting it.)

The user can type in another search term and hit the Search button. Here is an example of having typed in hat.

2. Implement a web service

The project directory name is DictionaryWebService

a. Implement a simple (can be a single path) API.

The URL of my web service deployed in Codespaces is https://fluffy-lamp-wrgrvwvxxv5xf5vjq-8080.app.github.dev/ . It has 2 types of services:

- word: Returns the possible definitions of a word (max 7).

- dashboard: Displays some analytics metrics and logs.

In my web server project:
- Model: DictionaryModel.java
- Controller: DictionaryServlet.java
- View: dashboard.jsp (just for the analytics and logs)

b. Receives an HTTP request from the native Android application

My DictionaryServlet.java receives the HTTP request with the parameter (path info) containing the word the user wants to know its possible definitions. It passes the word as a string to the model.

c. Executes business logic appropriate to your application. This includes fetching XML or JSON information from some 3rd party API and processing the response.

DictionaryModel.java makes an HTTP request to https://api.dictionaryapi.dev/api/v2/entries/en/. Then it parses the JSON response and extracts the parts of the answer it needs to respond to the Android application.

d. Replies to the Android application with an XML or JSON formatted response. The schema of the response can be of your own design.

The adjustResponse method adjusts the word in the required format. Then the Gson libraries convert it into JSON format following the attributes of the Word and Meaning classes.

3. Handle error conditions

No documentation required

4. Log Useful Information

The information that is stored is the following:

Request-Related Attributes

- **word (Searched Word):** This represents the specific word or term entered by the user in the application. It's critical for tracking user behavior, analyzing search patterns, and debugging search-related issues.

- **appRequestTime (App Request Time):** The timestamp when the server receives a request from the app. It marks the beginning of the request lifecycle, essential for tracking end-to-end response time.

- **deviceType (Device Type):** Indicates the type of device used to make a call to the web server (e.g., mobile, desktop, tablet). This could be useful for understanding user demographics and optimizing the exposed services for different devices.

API-Related Attributes

- **apiRequestTime (API Request Time):** The timestamp when the app sends a request to the external API. It's used to measure the time taken to initiate communication with the API.

- **apiReplyTime (API Reply Time):** The timestamp when the external API responds to the app's request. This helps measure API latency and diagnose issues related to slow API responses.

- **apiResponseCode (API Response Code):** A numeric code representing the status of the API response. It provides a quick overview of the API's health and response quality.

- **apiResponseCodeDescription (API Response Message):** A textual description of the apiResponseCode, giving more context about the response. Useful for logging and debugging API interactions.

Reply-Related Attributes

- **appReplyTime (App Reply Time):** The timestamp when the app sends a reply back to the user. This marks the end of the request lifecycle and helps calculate the total request-response duration.

- **appResponseCode (App Response Code):** A numeric code that indicates the result of the app's processing of the user's request. This is crucial for debugging issues at the application layer.

- **appResponseCodeDescription (App Response Message):** A textual description of the appResponseCode, providing additional details about the outcome of the request. This enhances the log's readability and aids in debugging.

5. Store the log information in a database

- **Connection String:** "mongodb+srv://jsantiagobv:SHIRO123@cluster.y4hq5.mongodb.net/?retryWrites=true&w=majority&appName=Cluster"
- **Primary Shard:** cluster-shard-00-01.y4hq5.mongodb.net:27017
- **Secondary Shard:** cluster-shard-00-00.y4hq5.mongodb.net:27017
- **Secondar Shard (III):** cluster-shard-00-02.y4hq5.mongodb.net:27017

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

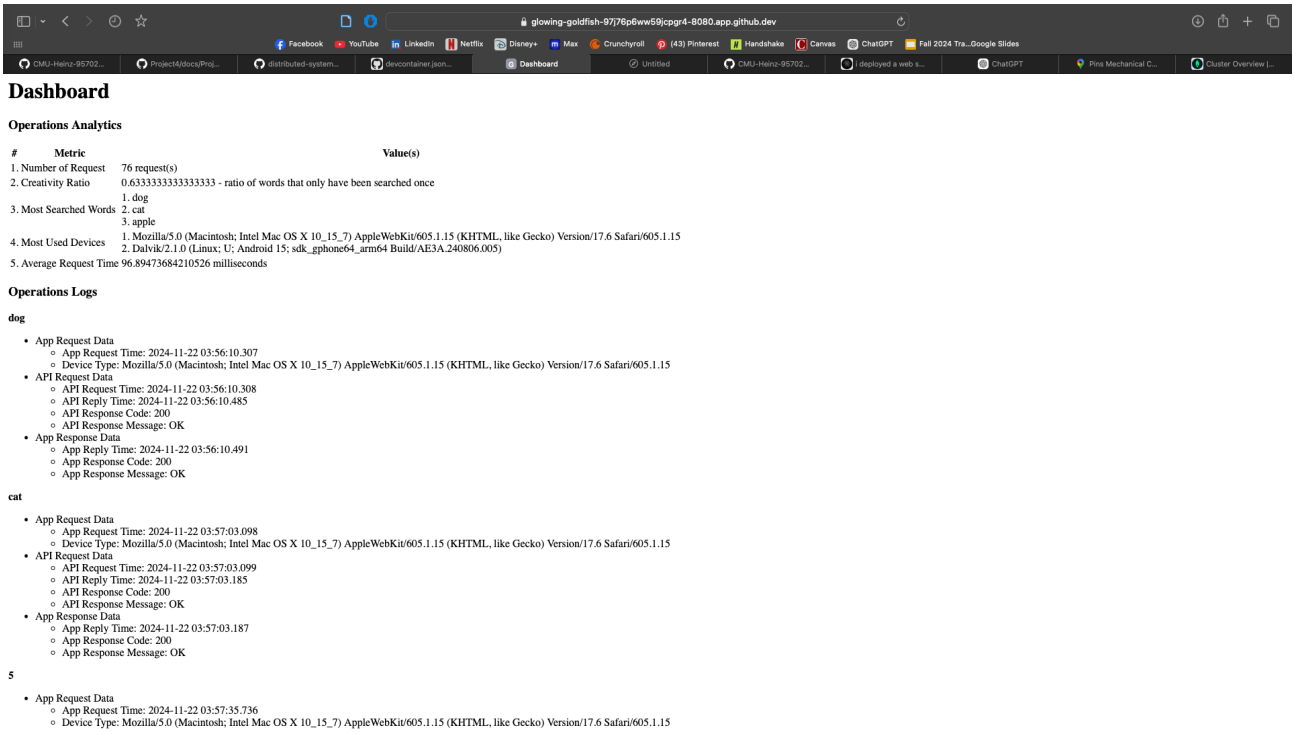
a. **A unique URL addresses a web interface dashboard for the web service.**

   The URL for the dashboard is: https://fluffy-lamp-wrgrvwvxxv5xf5vjq-8080.app.github.dev/dashboard

b. The dashboard displays at least 3 interesting operations analytics.

c. The dashboard displays formatted full logs.

For literals b and c, see image below:



7. Deploy the web service to GitHub Codespaces

All the steps have been followed. Please find in Github the following files:

- .devcontainer.json
- Dockerfile
- ROOT.war

References

For this project I used the following sources:

- Lab 2: InterestingPicture
- Lab 3: Containers and Cloud
- Lab 9: AndroidInterestingPicture
- ChatGPT
- Blackbox.ai