Name - Neharika
AndrewID- nneharik

PROJECT- 4

CatBreedApplication

Description:
CatBreedApplication is a mobile app that allows users to search for cat breeds and instantly view detailed information including origin, life span, temperament traits, and a breed image. The app sends a query to a Java servlet, which in turn queries TheCatAPI to fetch breed and image data. The app displays the fetched data in a clean UI and maintains a backend log of mobile user interactions.

Android App Overview:
The Android app features a search interface where users can enter a cat breed and retrieve associated details from a web service. Each request is sent to a Java servlet deployed on GitHub Codespaces. The servlet handles API communication with TheCatAPI and sends a formatted JSON response back to the app, which then updates the UI. The app also sends metadata such as the phone model and user-agent in headers, enabling server-side logging for operational analytics.

Dashboard Overview:
A separate web-based operations dashboard provides insights into the app's usage. It displays:

- Top 3 most searched cat breeds
- Average API response latency
- Most popular temperament traits
- Full logs of mobile-originated requests (excluding desktop/browser-based traffic)

This logging data is stored in MongoDB Atlas, and only requests originating from mobile devices are recorded per project requirements.

While the /cat endpoint is meant only for mobile requests, the /dashboard.jsp view is a browser-accessible page to review logs and analytics.

1. Implement a Native Android Application

The name of my native Android application project in Android Studio is: **CatBreedApplication**
The package name is : **com.example.catbreedapplication**

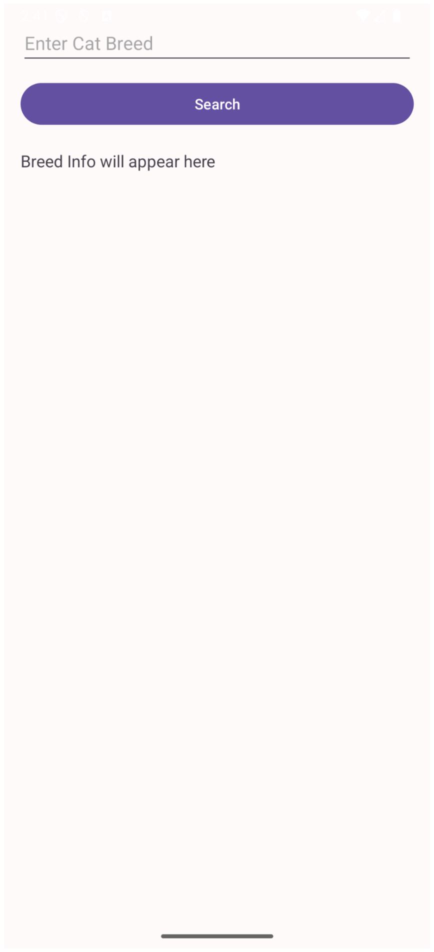
a. Has at least three different kinds of views in your Layout (TextView, EditText, ImageView, etc.)

My app uses the following views in its layout:

- **TextView** – Displays labels like breed name, origin, temperament, and life span.
- **EditText** – Lets the user enter the name of a cat breed.
- **ImageView** – Displays the image of the fetched cat breed from the API.
- **Button** – A single **Search** button triggers the backend query.

The views are organized vertically using a LinearLayout. The layout includes padding for spacing and uses centerCrop for proper scaling of the image within the ImageView.
See res/layout/activity_main.xml for implementation details.

Below is a screenshot of the layout before fetching a cat breed:

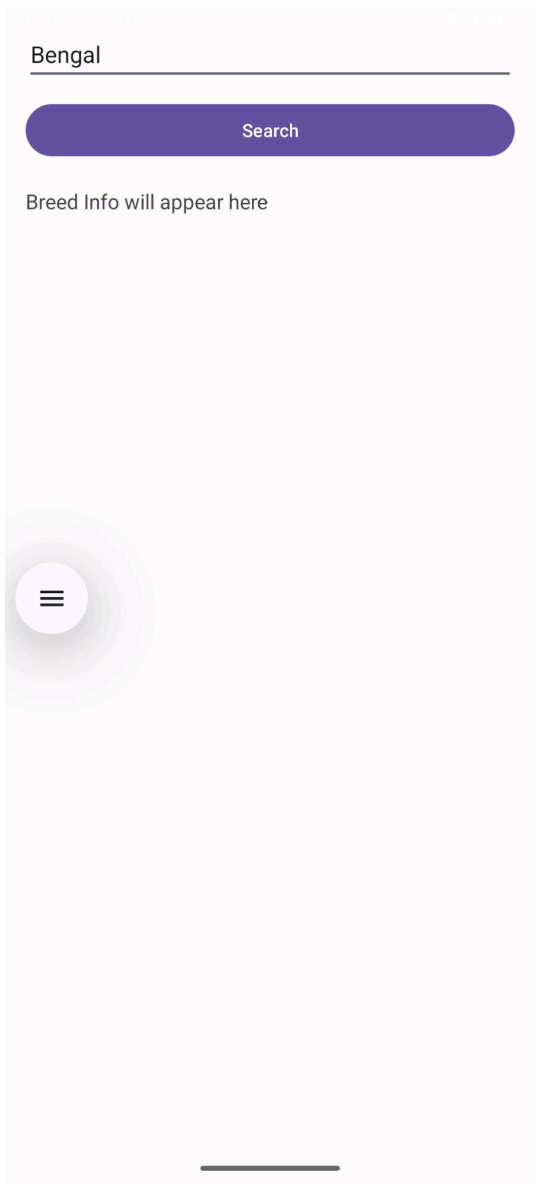


Breed Info will appear here

b. Requires input from the user

The user is required to input a cat breed name into the EditText field. This input is used to query the backend servlet, which in turn fetches breed-related information (such as origin, temperament, life span, and an image) from a third-party API.

Below is a screenshot of the user searching for a cat breed:



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

My application makes an HTTP GET request in MainActivity.java using an AsyncTask named FetchBreedTask. The request is made to the servlet hosted on GitHub Codespaces:

https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev/cat?breed=<userInpu
t>

Here, <userInput> is the cat breed name entered by the user in the Android app’s EditText field.

Although the servlet URL can be accessed from a browser, it only logs data when a mobile-originated request (with headers) is detected.

The FetchBreedTask opens a connection to the above URL, adds headers including the phone model and user agent, and reads the JSON response from the servlet. This JSON is then parsed and displayed in the app, including the cat’s breed name, origin, temperament, lifespan, and image.

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

The servlet endpoint returns a JSON response in the format:

```
{
  "name": "Bengal",
  "origin": "United States",
  "temperament": "Alert, Agile, Energetic, Demanding, Intelligent",
  "life_span": "12 - 15",
  "image_url": "https://cdn2.thecatapi.com/images/..."
}
```

In the FetchBreedTask class, the app uses the org.json.JSONObject class to parse the JSON response, extracting fields such as name, origin, temperament, life_span, and image_url. These values are then displayed in the UI using TextView for breed details and an ImageView to display the fetched image. If the response contains an error or is invalid, it is handled gracefully and an error message is shown in the app.

e. Displays new information to the user

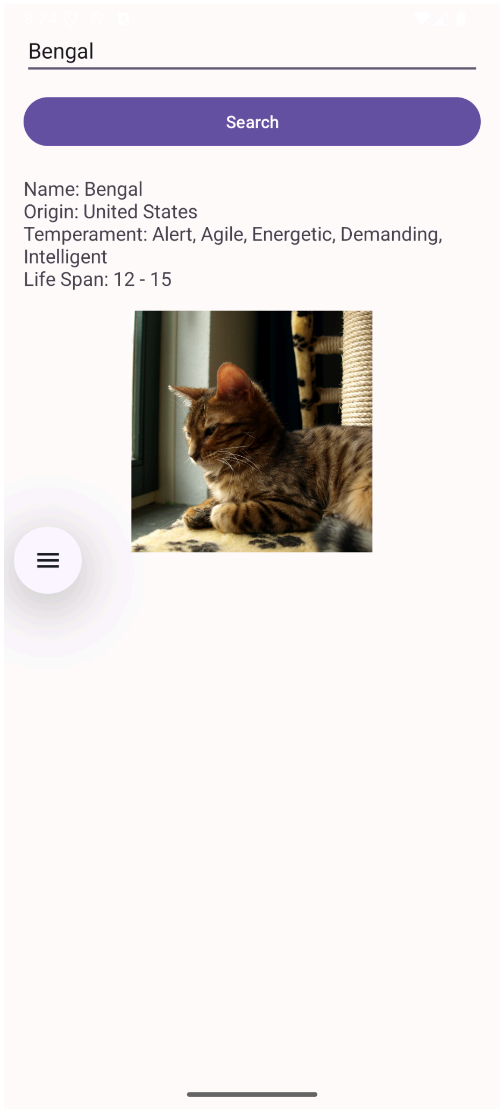
After fetching the cat breed information from the web service, the app displays:

- The cat's breed image in an ImageView.
- The breed name, origin, temperament, and life span in TextViews.

- These details appear only after a successful response and are dynamically updated based on the user’s input.

The TextView displays the structured cat information (e.g., "Name: Bengal", "Origin: United States", etc.), and the ImageView loads and renders the cat image fetched from the provided image_url in the JSON response.

Below is a screenshot of the app after fetching breed “Bengal”:



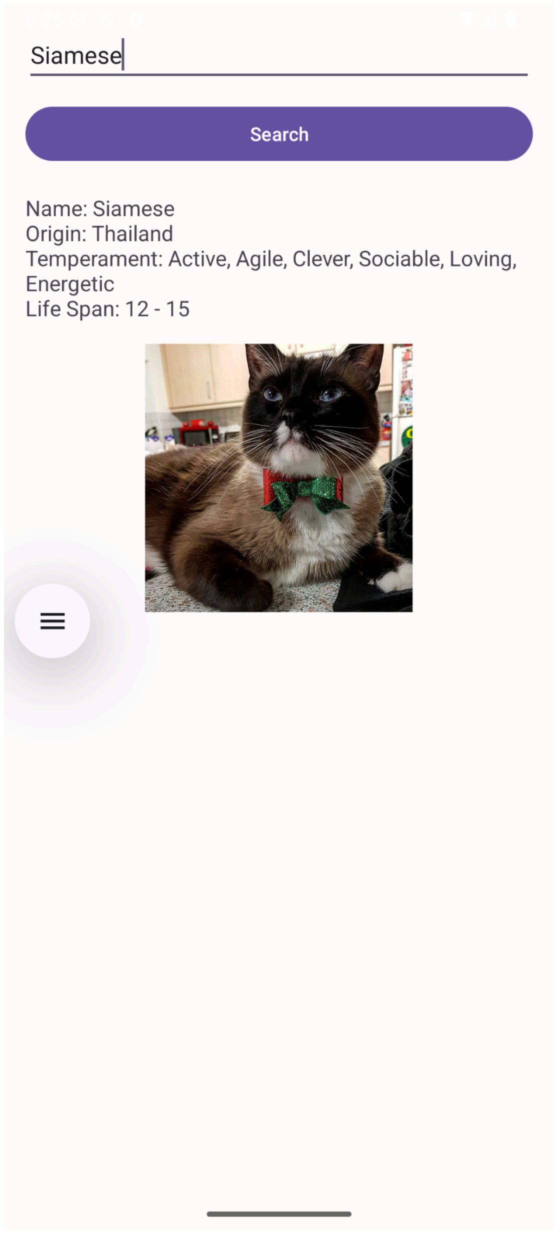
Name: Bengal
Origin: United States
Temperament: Alert, Agile, Energetic, Demanding, Intelligent
Life Span: 12 - 15

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

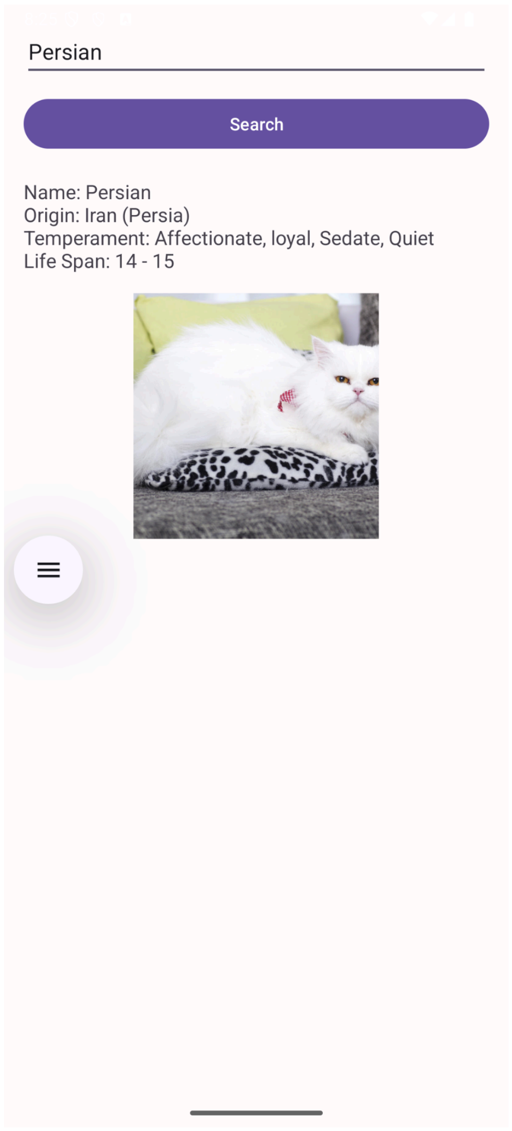
The user can repeatedly enter new cat breeds into the EditText and tap the "Search" button to fetch information about different breeds. Each request triggers a fresh API call to the server, and the app dynamically updates the TextView and ImageView with the latest breed details and image.

The previously displayed results are cleared or overwritten with each new search, allowing the user to continue exploring different breeds without needing to restart the app. This makes the experience seamless and highly repeatable.

The user can type in another breed to search and hit Search. Below is an example:



Name: Siamese
Origin: Thailand
Temperament: Active, Agile, Clever, Sociable, Loving, Energetic
Life Span: 12 - 15



Name: Persian
Origin: Iran (Persia)
Temperament: Affectionate, loyal, Sedate, Quiet
Life Span: 14 - 15

2. Implement a Web Application, Deployed to GitHub Codespaces

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

https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev/

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

In my web app project:

- **Model:** Not explicitly separated; the business logic (API call to TheCatAPI, latency tracking, and logging to MongoDB) is embedded directly in the servlet.

- **View:** The servlet returns a structured JSON response containing cat breed information, which is rendered by the Android app. Analytics such as most searched breeds and fetch latency are displayed via dashboard.jsp.

- **Controller:** CatBreedServlet.java handles the /cat endpoint. It processes the GET request from the Android app, fetches breed data, logs mobile requests, and returns the response.

The servlet is annotated with @WebServlet("/cat") and does not require explicit mapping in web.xml.

b. Receives an HTTP request from the native Android application

CatBreedServlet.java receives the HTTP GET request from the Android application with a query parameter breed. This parameter is extracted using request.getParameter("breed"). The servlet also collects header information such as phone-model and User-Agent to log device-specific data. The breed parameter is then used to process the request and respond with the corresponding cat breed information.

c. Executes business logic appropriate to your application

Inside CatBreedServlet.java, the servlet performs the business logic directly. It:

- Sends an HTTP request to TheCatAPI using the breed provided.
- Parses the JSON response to extract the cat's origin, temperament, lifespan, and image URL.
- Logs analytics (such as latency, device type, and breed popularity) into a MongoDB collection.
- Returns a structured JSON object containing the extracted breed information and metadata (e.g., fetch time).

This logic is embedded directly in the servlet and handles both external API interaction and data transformation.

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

The servlet (CatBreedServlet.java) sends the response back to the Android application in **JSON** format. The response includes key information such as:

```
{

  "name": "Bengal",
  "origin": "United States",

  "temperament": "Alert, Agile, Energetic, Demanding, Intelligent",

  "life_span": "12 - 15",

  "imageURL": "https://cdn2.thecatapi.com/images/image123.jpg"

}
```

The JSON is built directly in the servlet using a JsonObject or string construction and returned via response.getWriter().print(jsonString); with the appropriate Content-Type set to application/json. This allows the Android app to parse and display the relevant breed information dynamically.

3. Handle Error Conditions

- The Android app handles network errors using a toast message, such as:
  "Error: Network issue" when the connection fails or times out.

- It detects and manages invalid JSON responses by catching JSONException and showing an error like:
  "Error: Invalid response (not JSON)".

- On the server-side (CatBreedServlet.java), the web service responds with a 400 Bad Request if:
  The breed parameter is missing or invalid.
  Example JSON:
  { "error": "Breed not supported: <breed>", "latency_ms": <latency> }

- If the external API (TheCatAPI) fails to respond or throws an exception, the servlet catches it and returns:
  { "error": "Failed to fetch breed info: HTTP <code>", "latency_ms": <latency> }

- If no breed data is found (empty response), the server returns:
  { "error": "No information found for breed: <breed>", "latency_ms": <latency> }

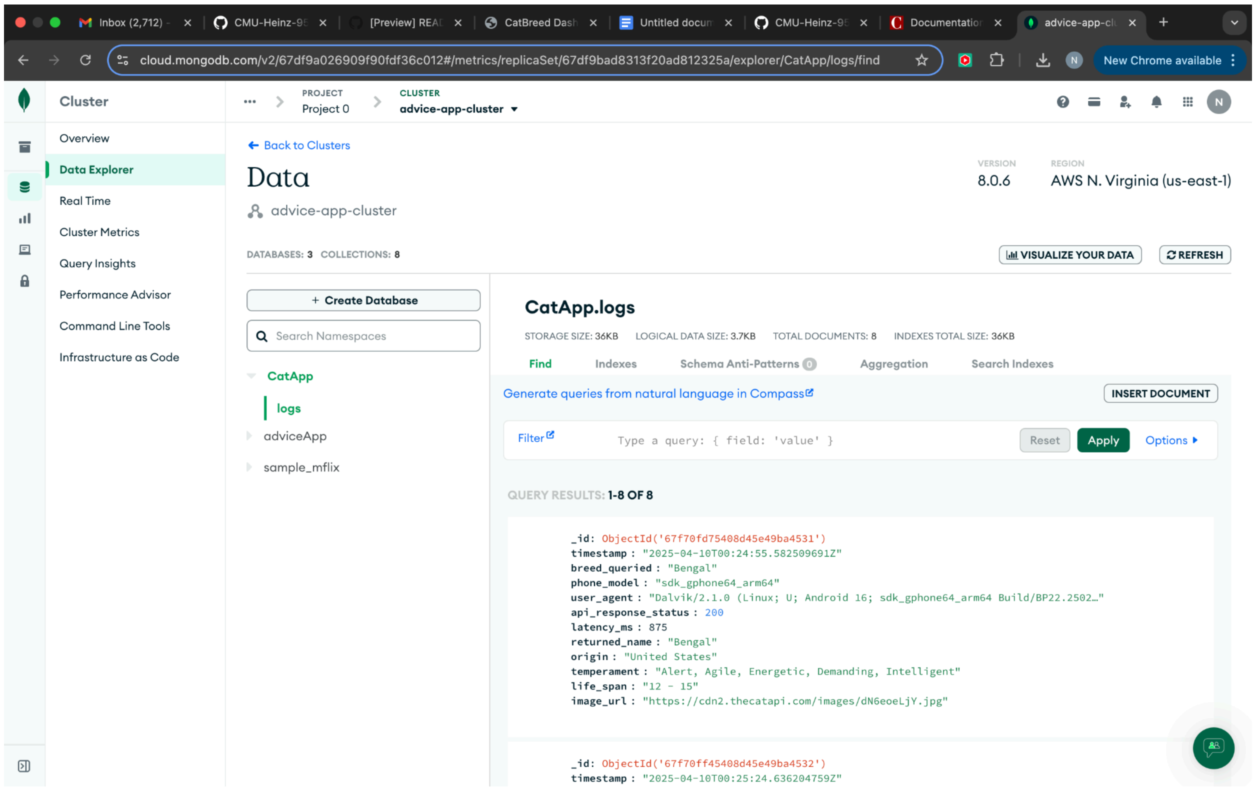
- On the Android app, if the image fails to load or URL is null/invalid, a toast like "Failed to load image" is shown to the user.

4. Log Useful Information

I log the following information for each request in MongoDB Atlas (in **CatBreedServlet.java**):

- **Timestamp**: When the request was made.
- **Breed Name**: The cat breed entered by the user via the Android app.
- **Image URL**: The image URL returned by TheCatAPI.
- **Breed Info**: A brief description or characteristics of the cat breed.
- **Latency (ms)**: The time taken to process and respond to the request.
- **User Agent**: The user agent string passed by the Android device (retrieved from the request header).
- **API Request URL**: The full endpoint used to query TheCatAPI.
- **API Response**: The raw JSON string received from TheCatAPI for debugging and analytics.

These logs are written to the **"logs"** collection in the **"CatApp"** database on MongoDB Atlas. I chose these fields to monitor API reliability, performance, and user query patterns. Logging is also done to the server console for real-time debugging.



```
_id: ObjectId('67f70fd75408d45e49ba4531')
timestamp : "2025-04-10T00:24:55.582509691Z"
breed_queried : "Bengal"
phone_model : "sdk_gphone64_arm64"
user_agent : "Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.2502…"
api_response_status : 200
latency_ms : 875
returned_name : "Bengal"
origin : "United States"
temperament : "Alert, Agile, Energetic, Demanding, Intelligent"
life_span : "12 - 15"
image_url : "https://cdn2.thecatapi.com/images/dN6eoeLjY.jpg"

_id: ObjectId('67f70ff45408d45e49ba4532')
timestamp : "2025-04-10T00:25:24.636204759Z"
```

5. Store the Log Information in a Database

The logs are stored in MongoDB Atlas using the following connection string:

mongodb+srv://nneharik:testing098123@advice-app-cluster.j4xd0.mongodb.net/?retryWrites=true&w=majority&appName=advice-app-cluster

- The database used is hosted on the advice-app-cluster MongoDB Atlas cluster.
- The logs are inserted into the Logs collection inside the CatApp database (based on your servlet code).
- The connection uses the MongoClient from the MongoDB Java driver.
- The appName=advice-app-cluster parameter is used to label the application in MongoDB's internal monitoring tools – it does not affect the database name or functionality.

This configuration enables remote logging and monitoring of all incoming API requests, helping debug issues and analyze usage.

6. Display Operations Analytics and Full Logs on a Web-Based Dashboard

A web-based dashboard is implemented to display operations analytics and full logs, accessible via the web application deployed on GitHub Pages. The dashboard is implemented in a JSP file named dashboard.jsp, which queries MongoDB and renders analytics using HTML. It is accessible via the root welcome file configured in web.xml and provides the following features:

- **Top 3 Most Searched Cat Breeds**: Displays the three most frequently queried cat breeds along with the number of times each breed was searched.
  (e.g., "Siamese — 2 searches", "Bengal — 1 search", "Ragdoll — 1 search").

- **Average Cat Info Fetch Time (ms)**: Shows the average time taken to fetch cat breed data from the Cat API.
  (e.g., "744.0 ms")

- **Most Popular Temperament Traits**: Aggregates and displays the most commonly appearing temperament traits across search results.
  (e.g., "Agile — 3 mentions", "Intelligent — 3 mentions", etc.)

- **Recent API Logs**: A detailed table displaying each request made by users with the following fields:

  - **Timestamp**: When the request was made (e.g., `"2025-04-10T00:27:36.110062514Z"`).

  - **Breed**: The name of the cat breed requested (e.g., `"Bombay"`).

  - **User Agent**: The full user agent string of the Android client.

  - **Status**: HTTP status code returned (e.g., `200`).

  - **Latency (ms)**: Time taken to fulfill the request (e.g., `717 ms`).

  - **Origin**: The country of origin of the cat breed (e.g., `"United States"`).

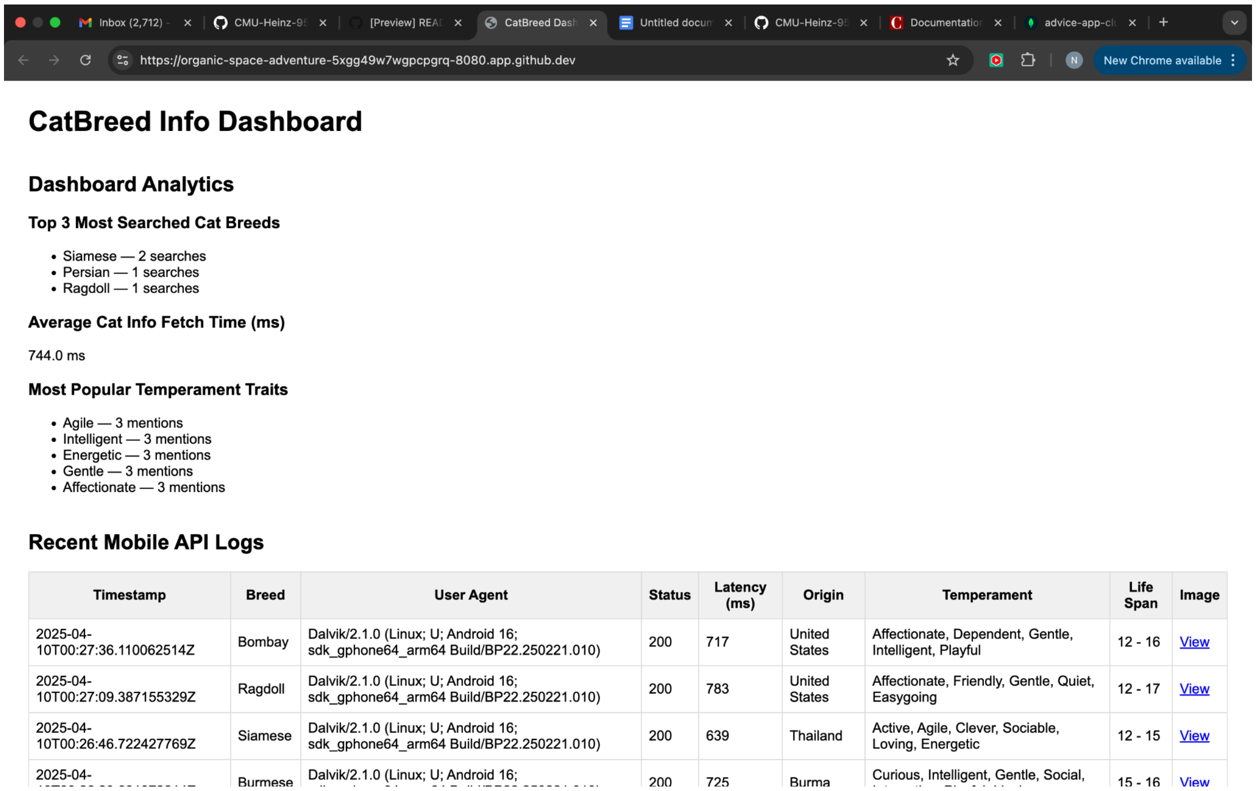
  - **Temperament**: A comma-separated list of behavioral traits.

  - **Life Span**: Expected lifespan of the breed (e.g., `"12 - 16"`).

  - **Image**: A thumbnail preview that links to the full image from the Cat API.

The dashboard uses **Bootstrap** for layout and responsive design, **Font Awesome** for icons, and **custom CSS** for enhanced styling (e.g., column spacing, colored badges, hover effects).

It is accessible at:
https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev/



https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev

CatBreed Info Dashboard

Dashboard Analytics

Top 3 Most Searched Cat Breeds

- Siamese — 2 searches
- Persian — 1 searches
- Ragdoll — 1 searches

Average Cat Info Fetch Time (ms)

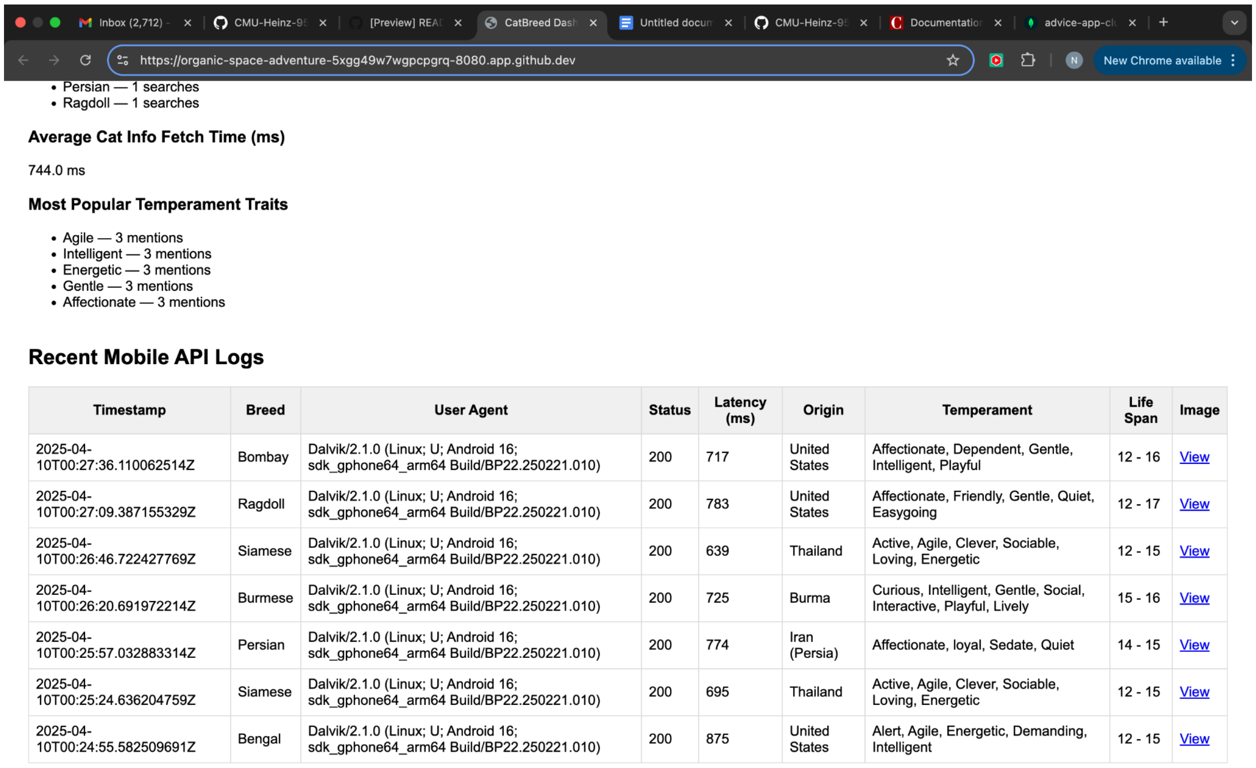
744.0 ms

Most Popular Temperament Traits

- Agile — 3 mentions
- Intelligent — 3 mentions
- Energetic — 3 mentions
- Gentle — 3 mentions
- Affectionate — 3 mentions

Recent Mobile API Logs

Timestamp	Breed	User Agent	Status	Latency (ms)	Origin	Temperament	Life Span	Image
2025-04-10T00:27:36.110062514Z	Bombay	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	717	United States	Affectionate, Dependent, Gentle, Intelligent, Playful	12 - 16	View
2025-04-10T00:27:09.387155329Z	Ragdoll	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	783	United States	Affectionate, Friendly, Gentle, Quiet, Easygoing	12 - 17	View
2025-04-10T00:26:46.722427769Z	Siamese	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	639	Thailand	Active, Agile, Clever, Sociable, Loving, Energetic	12 - 15	View
2025-04-	Burmese	Dalvik/2.1.0 (Linux; U; Android 16;	200	725	Burma	Curious, Intelligent, Gentle, Social,	15 - 16	View



https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev

- Persian — 1 searches
- Ragdoll — 1 searches

Average Cat Info Fetch Time (ms)

744.0 ms

Most Popular Temperament Traits

- Agile — 3 mentions
- Intelligent — 3 mentions
- Energetic — 3 mentions
- Gentle — 3 mentions
- Affectionate — 3 mentions

Recent Mobile API Logs

Timestamp	Breed	User Agent	Status	Latency (ms)	Origin	Temperament	Life Span	Image
2025-04-10T00:27:36.110062514Z	Bombay	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	717	United States	Affectionate, Dependent, Gentle, Intelligent, Playful	12 - 16	View
2025-04-10T00:27:09.387155329Z	Ragdoll	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	783	United States	Affectionate, Friendly, Gentle, Quiet, Easygoing	12 - 17	View
2025-04-10T00:26:46.722427769Z	Siamese	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	639	Thailand	Active, Agile, Clever, Sociable, Loving, Energetic	12 - 15	View
2025-04-10T00:26:20.691972214Z	Burmese	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	725	Burma	Curious, Intelligent, Gentle, Social, Interactive, Playful, Lively	15 - 16	View
2025-04-10T00:25:57.032883314Z	Persian	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	774	Iran (Persia)	Affectionate, loyal, Sedate, Quiet	14 - 15	View
2025-04-10T00:25:24.636204759Z	Siamese	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	695	Thailand	Active, Agile, Clever, Sociable, Loving, Energetic	12 - 15	View
2025-04-10T00:24:55.582509691Z	Bengal	Dalvik/2.1.0 (Linux; U; Android 16; sdk_gphone64_arm64 Build/BP22.250221.010)	200	875	United States	Alert, Agile, Energetic, Demanding, Intelligent	12 - 15	View

7. Setup Instructions

7.1 Prerequisites

- **GitHub Codespaces**: Used to run the backend in a cloud environment.
- **Docker** *(optional)*: Can be used to build and run the backend locally if preferred.
- **Apache Tomcat**: Required only for local deployment (not needed for Codespaces).
- **Android Studio**: Required to build and test the native Android client app.
- **MongoDB Atlas**: Used to store API logs. The connection string is provided in CatBreedServlet.java.
- **TheCatAPI Key**: Required to fetch breed information. The API key is there in CatBreedServlet.java.

7.2 Backend Setup (GitHub Codespaces)

1. Clone the Repository:
https://github.com/CMU-Heinz-95702/distributed-systems-project-04-neharikasrivastav

Verify the Android Endpoint:

The backend is not intended for browser use. Instead, it is designed to handle requests exclusively from the Android application, which sends an HTTP GET request with the cat breed name as a query parameter (e.g., ?breed=siamese).

Expected Request (from Android app):

GET /cat?breed=siamese

Expected JSON Response:
```
{
  "origin": "Thailand",
  "temperament": "Active, Agile, Clever, Sociable, Loving, Energetic",
  "life_span": "12 - 15",
  "image": "https://cdn2.thecatapi.com/images/ai6Jps4sx.jpg",
  "latency_ms": 639
}
```

Test the Analytics Dashboard
Although search is Android-only, the dashboard is web-based and can be accessed to view analytics:

https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev/

The dashboard displays:

- Most searched cat breeds
- Average fetch time
- Most common temperament traits
- Complete API logs (timestamp, breed, latency, origin, temperament, etc.)

7.3 Backend Setup (Local Docker)

1.Install Apache Tomcat

  - Tomcat is not automatically included in the Docker image. Install it manually and ensure it's available at /usr/local/tomcat.
  - You may install it via a previous local setup or within a Codespaces container.

2.Build the WAR File

```
mvn clean package
```

This generates the `ROOT.war` file in the `target/` directory.

3.Deploy the WAR File

```
cp target/ROOT.war /usr/local/tomcat/webapps/
```

4.Start Tomcat
Once Tomcat starts, it automatically deploys the WAR file. The backend will be ready to receive GET requests from the Android app, like:

```
GET /cat?breed=siamese
```

Note: The local backend does not support browser-based testing. Use the Android app to send breed queries and check logs via MongoDB Atlas or your web-based dashboard.

7.4 Android App Setup

1.Open in Android Studio

- Open the CatBreedApplication directory in Android Studio.
- Let Gradle sync complete automatically or trigger a sync manually if prompted.

2.Update the Endpoint URL

- Navigate to MainActivity.java inside app/src/main/java/com/example/catbreedapplication/.
- Locate and update the BASE_URL string for backend connection.

For Codespaces Deployment:

```
private static final String BASE_URL =
"https://organic-space-adventure-5xgg49w7wgpcpgrq-8080.app.github.dev/
";
```

For Local Docker/Tomcat Setup (with emulator):

```
private static final String BASE_URL = "http://10.0.2.2:8080/cat";
```

3.Run the App

- Use an Android emulator or connect a physical Android device.
- Click **Run** or use the terminal to build and deploy the app.
- The app sends a GET request with the selected breed name and displays breed info from your servlet.

8. Challenges and Solutions

Port Visibility in Codespaces:

- **Challenge:** GitHub Codespaces occasionally set port 8080 as private, which prevented access to the web app endpoint.
- **Solution:** Manually updated the devcontainer.json to expose port 8080 publicly by default for accessing the CatBreed servlet and dashboard.

Browser-Based Access Not Needed:

- **Challenge:** Unlike traditional projects that support browser-based queries, this app is designed solely for mobile (Android) access.
- **Solution:** Omitted desktop user interface support in the servlet and focused exclusively on parsing User-Agent for Android devices.

JSON Parsing Errors (Android Side):

- **Challenge:** Android app failed to parse malformed or unexpected responses (e.g., missing fields or HTML errors).
- **Solution:** Added validation to ensure the response starts with { before parsing and used error-handling blocks to display meaningful toast messages.

Cat API Key Rate Limits:

- **Challenge:** Hitting the limit on the Cat API when fetching breed data frequently during testing.
- **Solution:** Implemented fallback error messages and retry logic with logging to MongoDB when rate limits are triggered.

MongoDB Connection Issues:

- **Challenge:** Initial MongoDB setup failed due to misconfigured connection URI and credentials.

- **Solution:** Updated the MONGO_URI in CatBreedServlet.java with proper credentials and ensured the advice-app-cluster database was reachable.

Tomcat Not Automatically Installed in Docker:

- **Challenge:** Docker setup did not include Tomcat by default, which prevented local WAR file deployment.
- **Solution:** Documented a manual Tomcat setup in /usr/local/tomcat, and added startup instructions using catalina.sh.

GitHub File Size Limit for Android APK:

- **Challenge:** Final .apk or zipped Android Studio project exceeded GitHub's 25MB direct upload limit.
- **Solution:** Used Git Large File Storage (Git LFS) to track and upload the .apk file or .zip bundle. Added .gitattributes and committed using LFS.