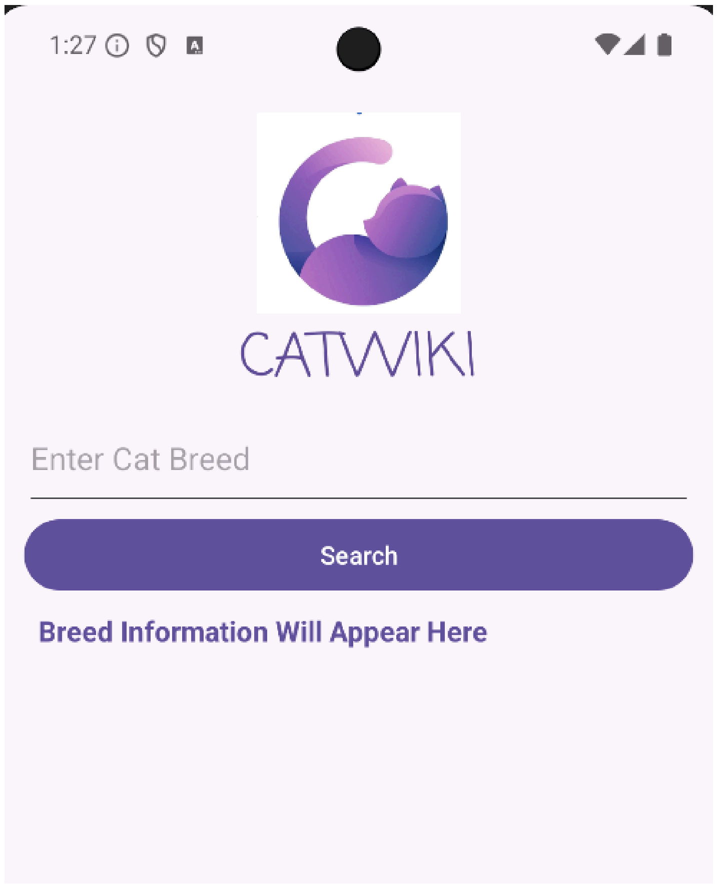
Project 4 Task 2 – CatWiKi

Yunqing Xiao (yunqingx)
The name of the API: The Cat API
URL: https://thecatapi.com/

1. Implement a native Android application

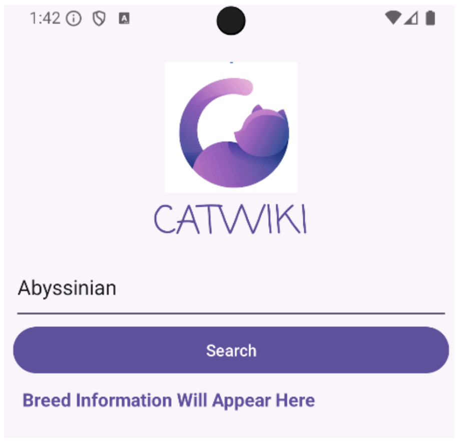


a. At least three different kinds of views:

- My Android application uses TextView, EditText, Button, and ImageView. These are implemented in the layout file activity_cat_picture.xml.

b. Requires user input:

- The application prompts the user to input a search term to fetch relevant information from the web service.

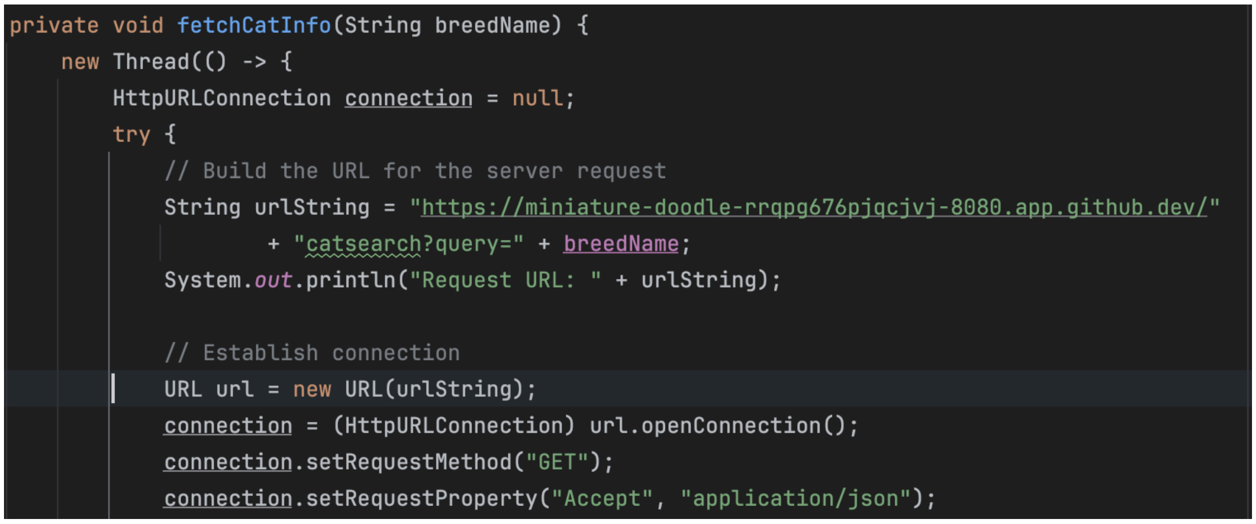


c. Makes an HTTP request:

- The application makes an HTTP GET request to the web service. The request URL includes the user's input as a query parameter.

https://miniature-doodle-rrqpg676pjqcjvj-8080.app.github.dev/catsearch?query=<user_input>

A task is executed to send an HTTP GET request to the `/catsearch` web service for retrieving cat breed information based on user input. Utilizing Java's `URL` and `HttpURLConnection` classes, it constructs a URL with the breed name as a query parameter, establishes the connection, and verifies the response code to ensure the request was successful. Upon success, it reads the input stream, parses the JSON response into a `JsonObject` using Gson, and updates the UI thread with the retrieved breed details and associated image or an error message if the operation fails. This asynchronous approach allows the application to fetch data from the remote web service in the background, preventing the UI thread from being blocked and delivering a seamless user experience with timely results for the user's query. Below is the relevant code:



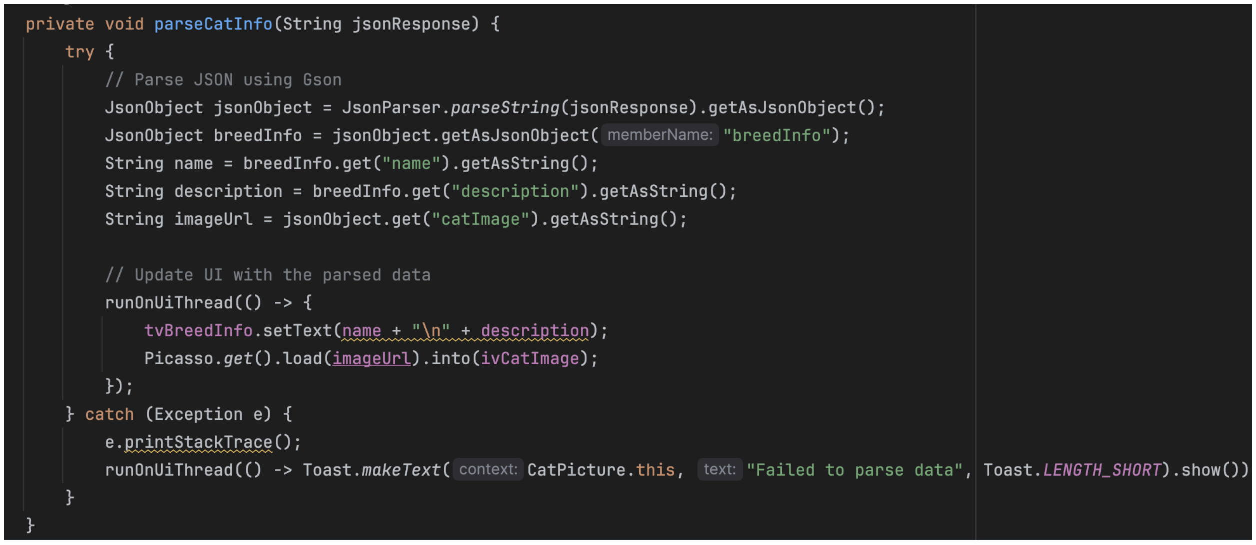
```java
private void fetchCatInfo(String breedName) {
    new Thread(() -> {
        HttpURLConnection connection = null;
        try {
            // Build the URL for the server request
            String urlString = "https://miniature-doodle-rrqpg676pjqcjvj-8080.app.github.dev/"
                    + "catsearch?query=" + breedName;
            System.out.println("Request URL: " + urlString);

            // Establish connection
            URL url = new URL(urlString);
            connection = (HttpURLConnection) url.openConnection();
            connection.setRequestMethod("GET");
            connection.setRequestProperty("Accept", "application/json");
```

d. Parses an XML or JSON response:

- The application receives a JSON response and uses the Gson library to parse it into Java objects.

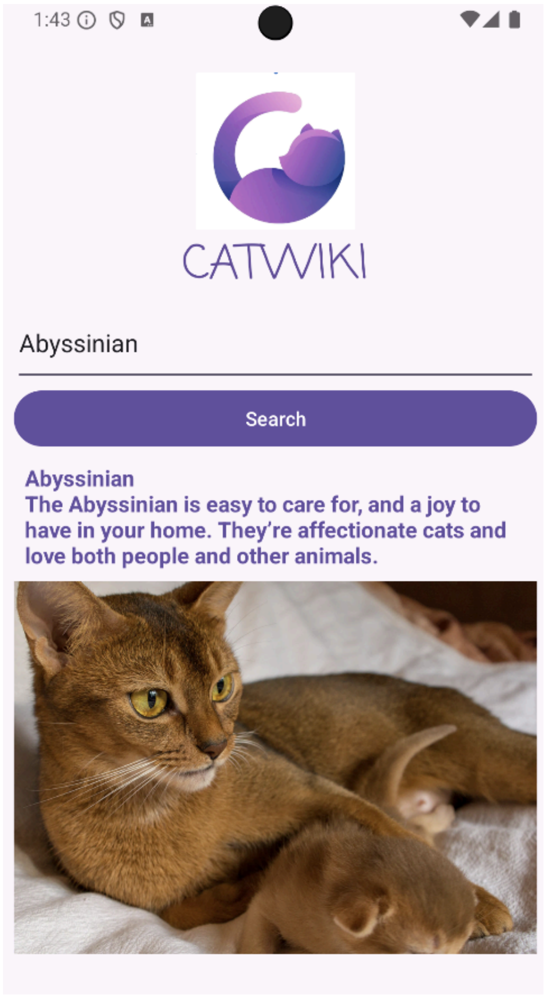
In the Android application, the received JSON response from the `/catsearch` web service is parsed into a `JsonObject` using the Gson library. This process converts the JSON string, which contains detailed information about the requested cat breed and its associated image, into a structured format that the application can easily process. After parsing, the application checks if the response contains valid data. If valid data is present, the application extracts the breed's details, such as its name and description, as well as the URL of the associated image. It then displays this information, loading the image using the Picasso library and updating the text view with the breed's details. If no valid data is found, the application updates the text view to notify the user that no information is available for their query.

This process highlights the application's ability to handle JSON responses dynamically, providing users with detailed, query-specific content fetched from a remote service while ensuring a seamless and engaging experience.



```
private void parseCatInfo(String jsonResponse) {
    try {
        // Parse JSON using Gson
        JsonObject jsonObject = JsonParser.parseString(jsonResponse).getAsJsonObject();
        JsonObject breedInfo = jsonObject.getAsJsonObject( memberName: "breedInfo");
        String name = breedInfo.get("name").getAsString();
        String description = breedInfo.get("description").getAsString();
        String imageUrl = jsonObject.get("catImage").getAsString();

        // Update UI with the parsed data
        runOnUiThread(() -> {
            tvBreedInfo.setText(name + "\n" + description);
            Picasso.get().load(imageUrl).into(ivCatImage);
        });
    } catch (Exception e) {
        e.printStackTrace();
        runOnUiThread(() -> Toast.makeText( context: CatPicture.this,  text: "Failed to parse data", Toast.LENGTH_SHORT).show())
    }
}
```

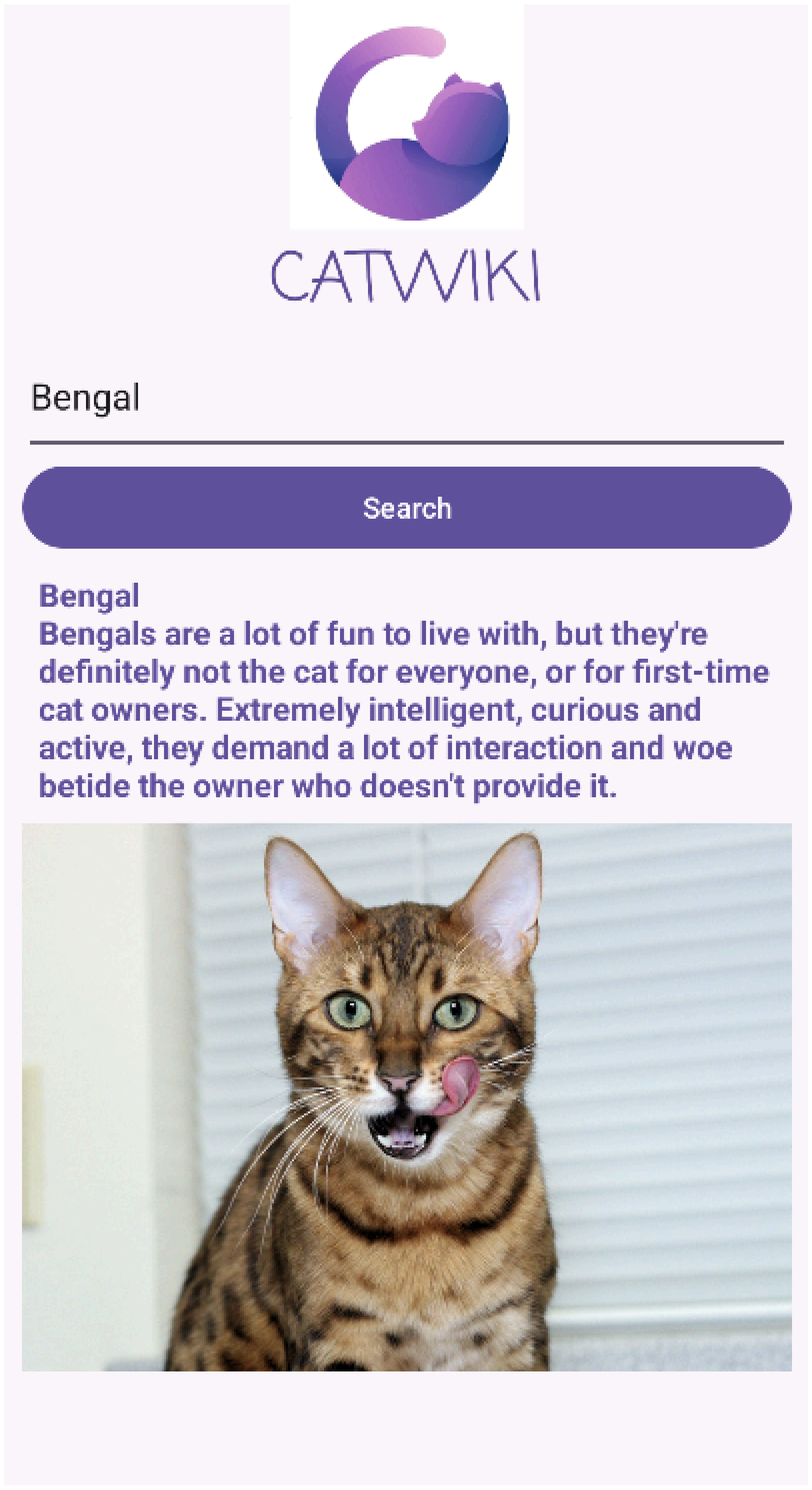
e. Displays new information to the user:



Abyssinian
The Abyssinian is easy to care for, and a joy to have in your home. They're affectionate cats and love both people and other animals.

f. Is repeatable:

- Users can input new terms and retrieve updated results without restarting the application.

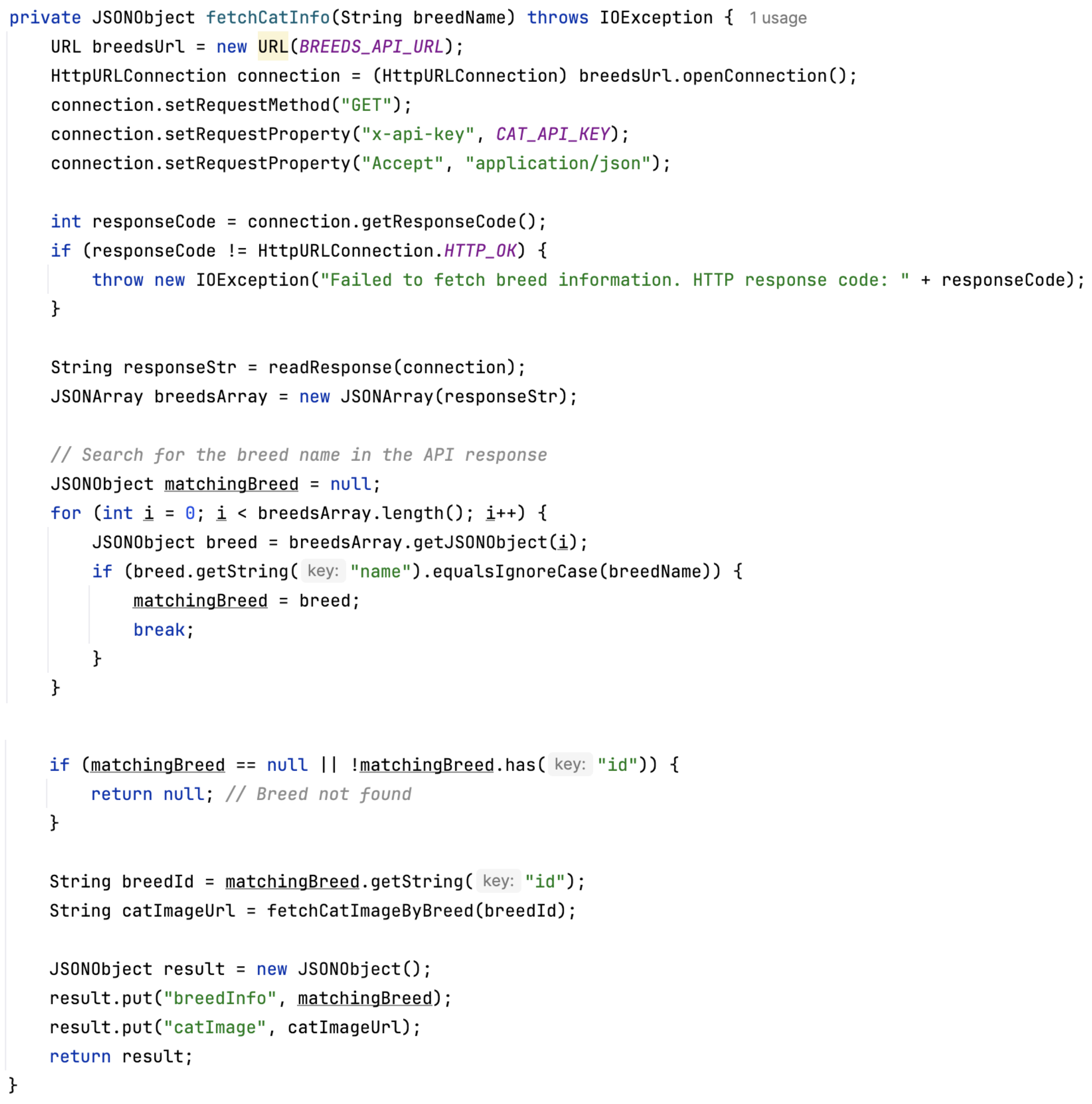


Bengal
Bengals are a lot of fun to live with, but they're definitely not the cat for everyone, or for first-time cat owners. Extremely intelligent, curious and active, they demand a lot of interaction and woe betide the owner who doesn't provide it.

2. Implement a web service

a. Simple API:

- The web service exposes a single endpoint /search. It accepts HTTP GET requests with a query parameter query.



```
private JSONObject fetchCatInfo(String breedName) throws IOException {  1 usage
    URL breedsUrl = new URL(BREEDS_API_URL);
    HttpURLConnection connection = (HttpURLConnection) breedsUrl.openConnection();
    connection.setRequestMethod("GET");
    connection.setRequestProperty("x-api-key", CAT_API_KEY);
    connection.setRequestProperty("Accept", "application/json");

    int responseCode = connection.getResponseCode();
    if (responseCode != HttpURLConnection.HTTP_OK) {
        throw new IOException("Failed to fetch breed information. HTTP response code: " + responseCode);
    }

    String responseStr = readResponse(connection);
    JSONArray breedsArray = new JSONArray(responseStr);

    // Search for the breed name in the API response
    JSONObject matchingBreed = null;
    for (int i = 0; i < breedsArray.length(); i++) {
        JSONObject breed = breedsArray.getJSONObject(i);
        if (breed.getString( key: "name").equalsIgnoreCase(breedName)) {
            matchingBreed = breed;
            break;
        }
    }

    if (matchingBreed == null || !matchingBreed.has( key: "id")) {
        return null; // Breed not found
    }

    String breedId = matchingBreed.getString( key: "id");
    String catImageUrl = fetchCatImageByBreed(breedId);

    JSONObject result = new JSONObject();
    result.put("breedInfo", matchingBreed);
    result.put("catImage", catImageUrl);
    return result;
}
```

b. Receives HTTP requests:

- Requests from the Android application are received and logged by the CatInfoServlet.java servlet.

The servlet demonstrates its capability to handle incoming HTTP GET requests from a native Android application. Upon receiving a request, it extracts the `query` parameter from the request URL, along with the client's IP address and device information from the `User-Agent` header. If the `query` parameter is missing, empty, or invalid, the servlet responds with a `400 Bad Request` error. Otherwise, it performs a search for cat breed information using the provided query.

The servlet interacts with The Cat API to fetch breed details and associated images, implementing retry logic to handle potential API failures. The request details—including the timestamp, search query, client IP address, device information, and API response time—are logged into MongoDB for future analytics. This ensures that all interactions are recorded for further insights and operational monitoring. Below is an example of how the servlet handles these requests:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

   String breedName = request.getParameter("query");

   String ipAddress = request.getRemoteAddr();

   String userAgent = request.getHeader("User-Agent");


   // Validate input query

   if (breedName == null || breedName.trim().isEmpty()) {

       response.sendError(HttpServletResponse.SC_BAD_REQUEST, "The breed name
cannot be empty or null.");

       return;

   }

   if (!breedName.matches("[a-zA-Z ]+")) { // Validate alphabetic characters
and spaces

       response.sendError(HttpServletResponse.SC_BAD_REQUEST, "Invalid breed
name format. Only alphabetic characters are allowed.");

       return;

   }


   long startTime = System.currentTimeMillis();

   JSONObject catInfoResponse;
```

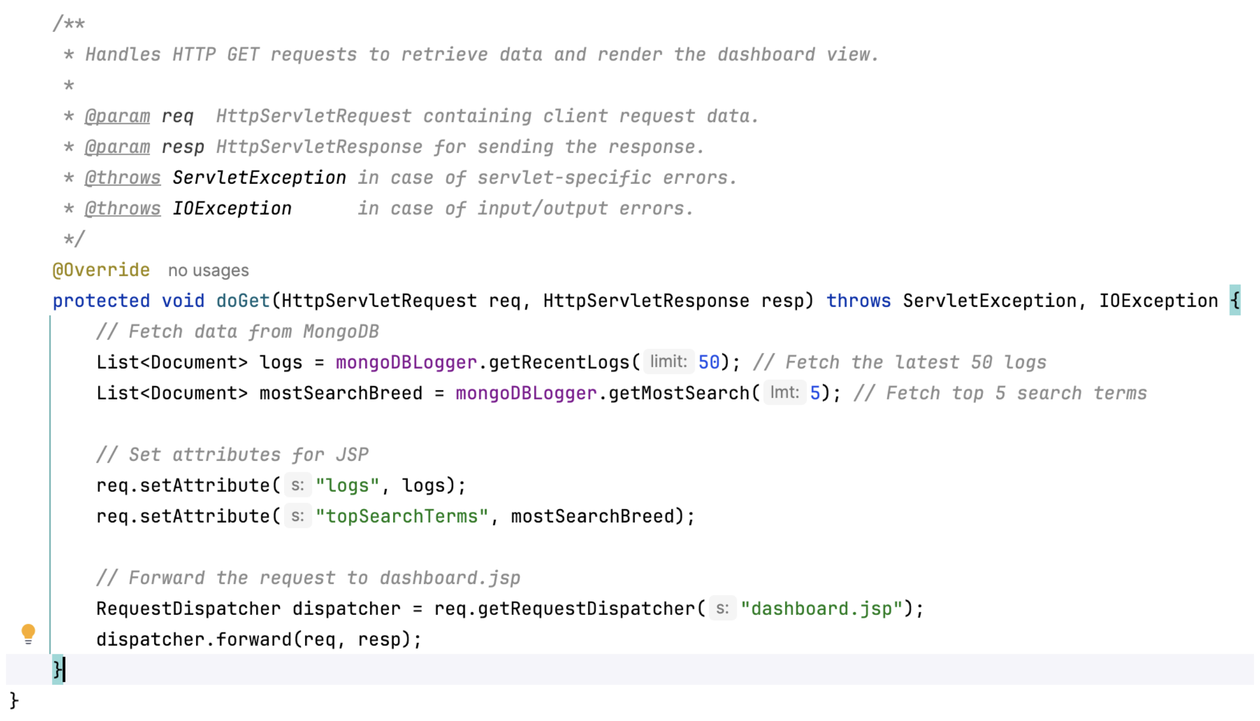
```
    try {
        // Fetch cat breed information with retry logic
        catInfoResponse = fetchCatInfoWithRetry(breedName, 3);
    } catch (IOException e) {
        // Log error and send internal server error response
        mongoDBLogger.logRequest(
                new Document("requestTimestamp", new Date())
                        .append("clientDevice", userAgent)
                        .append("clientIpAddress", ipAddress)
                        .append("queryParameter", breedName)
                        .append("error", e.getMessage())
        );
        response.sendError(HttpServletResponse.SC_INTERNAL_SERVER_ERROR, "The
server encountered an error. Please try again later.");
        return;
    }

    if (catInfoResponse == null) {
        // Respond with a 404 if no breed information is found
        response.sendError(HttpServletResponse.SC_NOT_FOUND, "No cat information
found for the given breed.");
        return;
    }

    long endTime = System.currentTimeMillis();

    // Log successful request and response details
    mongoDBLogger.logRequest(
            new Document("requestTimestamp", new Date())
                    .append("clientDevice", userAgent)
                    .append("clientIpAddress", ipAddress)
                    .append("queryParameter", breedName)
                    .append("apiRequestTimestamp", new Date())
                    .append("apiLatencyMs", endTime - startTime)
                    .append("responseStatus", HttpServletResponse.SC_OK)
                    .append("responseData", catInfoResponse.toString())
    );

    // Send JSON response
    response.setContentType("application/json");
    response.setCharacterEncoding("UTF-8");
    PrintWriter out = response.getWriter();
    out.print(catInfoResponse.toString());
    out.flush();
}
```

c. Executes business logic:

- The web service fetches data from the Cat API, processes it, and formats the response.

```
long endTime = System.currentTimeMillis();

// Log successful request and response details
mongoDBLogger.logRequest(
        new Document("requestTimestamp", new Date())
                .append("clientDevice", userAgent)
                .append("clientIpAddress", ipAddress)
                .append("queryParameter", breedName)
                .append("apiRequestTimestamp", new Date())
                .append("apiLatencyMs", endTime - startTime)
                .append("responseStatus", HttpServletResponse.SC_OK)
                .append("responseData", catInfoResponse.toString())
);

// Send JSON response
response.setContentType("application/json");
response.setCharacterEncoding("UTF-8");
PrintWriter out = response.getWriter();
out.print(catInfoResponse.toString());
out.flush();
```

d. Replies with JSON:



```
   /**
    * Handles HTTP GET requests to retrieve data and render the dashboard view.
    *
    * @param req  HttpServletRequest containing client request data.
    * @param resp HttpServletResponse for sending the response.
    * @throws ServletException in case of servlet-specific errors.
    * @throws IOException      in case of input/output errors.
    */
   @Override  no usages
   protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
       // Fetch data from MongoDB
       List<Document> logs = mongoDBLogger.getRecentLogs( limit: 50); // Fetch the latest 50 logs
       List<Document> mostSearchBreed = mongoDBLogger.getMostSearch( lmt: 5); // Fetch top 5 search terms

       // Set attributes for JSP
       req.setAttribute( s: "logs", logs);
       req.setAttribute( s: "topSearchTerms", mostSearchBreed);

       // Forward the request to dashboard.jsp
       RequestDispatcher dispatcher = req.getRequestDispatcher( s: "dashboard.jsp");
       dispatcher.forward(req, resp);
   }
}
```

4. Log useful information
Information logged includes:

1. Timestamp of the request.

2. User query term.

3. Response time.

4. HTTP status code.

5. Data fetched from the 3rd Party API.

6. Response sent back to the Android application.

5. Store log information in a database

1. **Timestamp of Request:** Records the precise date and time when the mobile application made the request. This information is essential for identifying usage trends, such as peak activity hours, and for maintaining a chronological log of interactions. This is logged using `new Date()` in the MongoDB document.
2. **Search Query Parameter:** Captures the specific breed name entered by the user as the search query. Logging this parameter helps analyze user interests, identify popular breeds, and guide future feature improvements or promotional efforts. It is extracted from the `query` parameter in the HTTP request.
3. **Client Device Information:** Extracts details about the user's device, such as its make and model, using the `User-Agent` header from the HTTP request. This information is valuable for optimizing the app's performance on frequently used devices and identifying device-specific issues.
4. **Request IP Address:** Logs the IP address of the device initiating the request, obtained using `request.getRemoteAddr()`. This data can be leveraged for geolocation analysis to understand user distribution and customize the app's offerings based on regional preferences.
5. **Response Code from The Cat API:** Logs the HTTP response code received from The Cat API during the external API call. This helps in diagnosing issues with the third-party service and monitoring the stability and reliability of the integration.
6. **API Response Latency:** Measures and logs the time taken by The Cat API to respond to a request, calculated as the difference between request initiation and response completion times. Tracking latency enables the identification of performance bottlenecks and helps enhance the app's responsiveness.

Here is an example of how these parameters are logged in the MongoDB document:

```
mongoDBLogger.logRequest(

    new Document("timestamp", new Date())
```

```
            .append("query", breedName)

            .append("ipAddress", ipAddress)

            .append("userAgent", userAgent)

            .append("responseCode", responseCode)

            .append("apiLatencyMs", endTime - startTime)

);
```

By systematically capturing and analyzing this data, the application gains insights into user behavior, device usage, and system performance, enabling informed decisions for future enhancements and better user experiences.

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

a. Unique URL:

  - Dashboard accessible at: https://miniature-doodle-rrqpg676pjqcjvj-8080.app.github.dev/dashboard

b. Operations analytics:

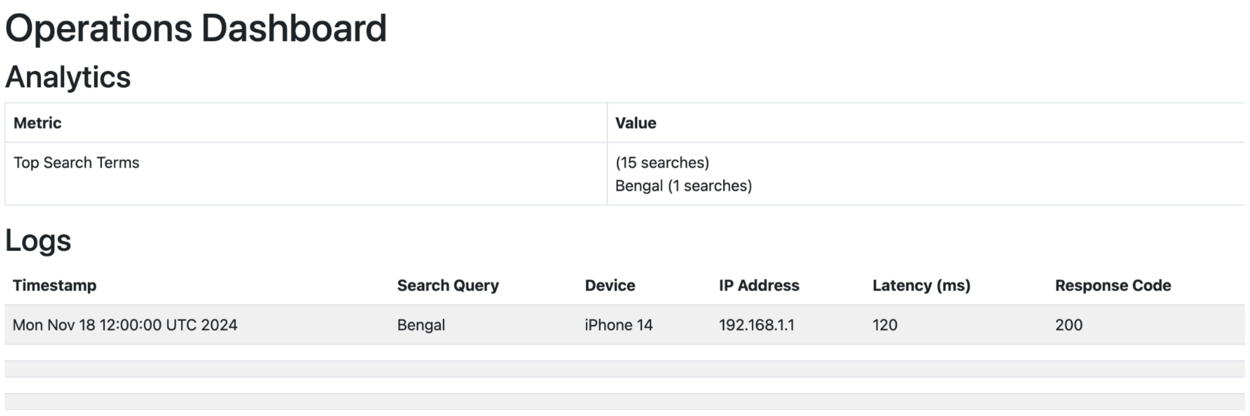
  - Top 5 search queries.

  - Average response time.

  - Number of successful vs. failed requests.

c. Formatted logs:

  - Logs are displayed in a paginated HTML table on the dashboard using JSP and Bootstrap.


Operations Dashboard
Analytics

Metric	Value
Top Search Terms	(15 searches) Bengal (1 searches)

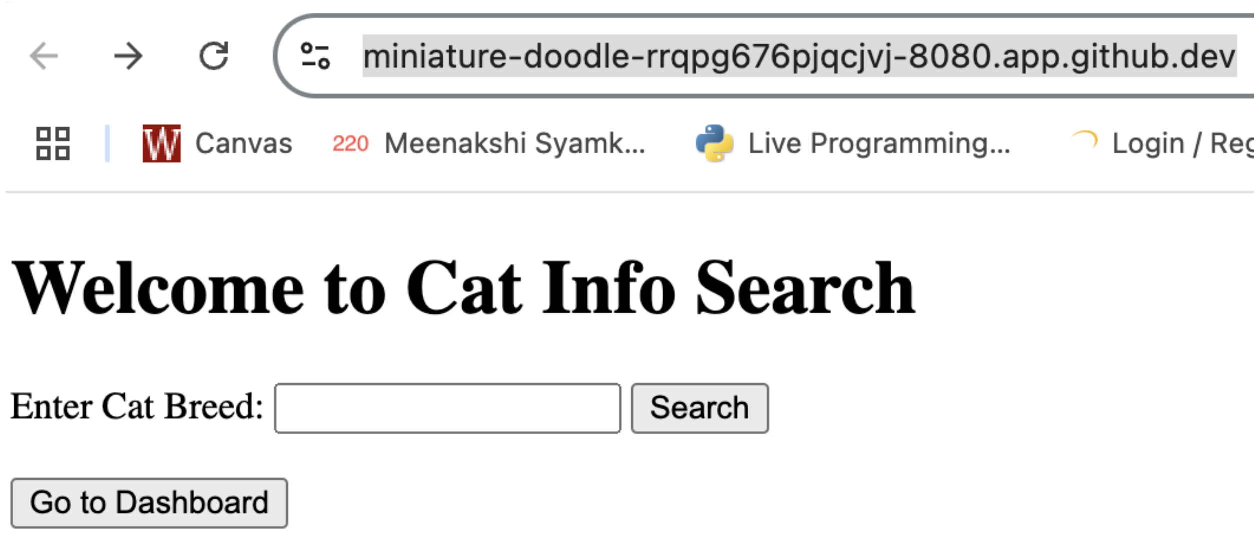
Logs

Timestamp	Search Query	Device	IP Address	Latency (ms)	Response Code
Mon Nov 18 12:00:00 UTC 2024	Bengal	iPhone 14	192.168.1.1	120	200

7. Deploy the web service to GitHub Codespaces
Web service deployed using GitHub Codespaces at:

https://miniature-doodle-rrqpg676pjqcjvj-8080.app.github.dev/.

Screenshot of deployed service working in an incognito browser:



Welcome to Cat Info Search

Enter Cat Breed:

Class: CatDashboard

The CatDashboard servlet provides an interface for displaying analytics and logs related to cat breed searches.
It retrieves recent search logs and the most frequently searched breeds from a MongoDB database and forwards
this data to a JSP page for rendering a dashboard view.

Key Features:
- Fetches the latest 10 log entries for display.
- Retrieves the top 5 most frequently searched cat breeds.
- Forwards the data to dashboard.jsp for presentation.

Mapped to the URL pattern /dashboard.

Class: CatInfoServlet

The CatInfoServlet handles HTTP GET requests for retrieving detailed information about cat breeds.
It integrates with The Cat API to fetch breed details and an associated image based on a client-provided 'query' parameter.

Key Features:
- Validates input (non-empty, alphabetic query).
- Fetches breed details and an image from The Cat API.
- Logs requests and responses to MongoDB.
- Includes retry logic for API calls.

Mapped to the URL pattern /catsearch.

Class: MongoDBLogger

The MongoDBLogger class manages logging and retrieving data from MongoDB.
Key Features:
- Logs request metadata as MongoDB documents.
- Retrieves the most frequently searched queries.
- Fetches the most recent log entries.
- Utilizes the MongoDB Java Driver for database operations.

Class: CatPicture

The CatPicture Android activity allows users to search for cat breed information by entering a breed name.
Key Features:
- Fetches breed details from the web service.
- Displays breed information and an associated image using a user-friendly interface.
- Provides input validation and error handling.

Class: CatDashboard

The CatDashboard servlet provides an interface for displaying analytics and logs related to cat breed searches.
Key Features:
- Fetches the latest 50 log entries for display.
- Retrieves the top 5 most frequently searched cat breeds.
- Uses MongoDBLogger for database operations, fetching logs and aggregated search terms.
- Forwards the data to dashboard.jsp for rendering the dashboard.

Class: CatInfoServlet

The CatInfoServlet handles HTTP GET requests to fetch detailed information about cat breeds.
Key Features:
- Integrates with The Cat API to retrieve breed details and associated images.
- Validates input query for correct format and non-empty values.
- Implements retry logic for handling API failures (up to 3 attempts).
- Logs requests and responses using MongoDBLogger, capturing metadata like timestamps, client details, and response data.
- Returns JSON responses to the client, including breed details and an image URL.

Class: MongoDBLogger

The MongoDBLogger class provides functionality for logging requests and retrieving analytics data from MongoDB.
Key Features:
- Logs request metadata as MongoDB documents, including timestamps, client details, and search parameters.
- Aggregates search terms to identify the most frequently searched breeds.
- Retrieves the most recent log entries for dashboard display.
- Utilizes the MongoDB Java Driver for database operations.

Class: CatPicture

The CatPicture Android activity allows users to search for cat breed information by entering a breed name.

Key Features:
- Provides an intuitive interface with input fields, buttons, and displays for breed details and images.
- Sends HTTP GET requests to the web service /catsearch.
- Parses JSON responses to extract and display breed information and an image.
- Utilizes the Picasso library for loading and displaying images.
- Handles user input validation and error feedback gracefully.