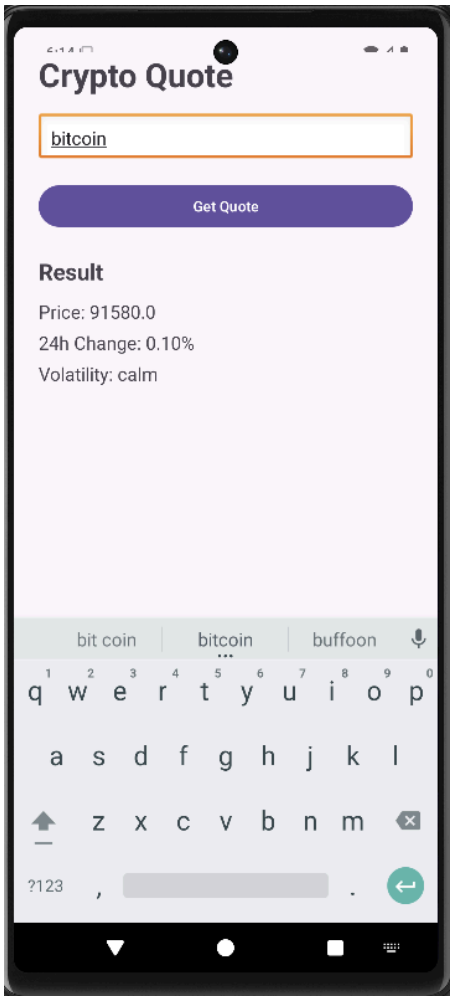
1 Implement a native Android application

a. Has at least three different kinds of Views in your Layout (TextView, EditText, ImageView, or anything that extends android.view.View). In order to figure out if something is a View, find its API. If it extends android.view.View then it is a View.
b. Requires input from the user
c. Makes an HTTP request (using an appropriate HTTP method) to your web service
d. Receives and parses an XML or JSON formatted reply from your web service
e. Displays new information to the user
f. Is repeatable (I.e. the user can repeatedly reuse the application without restarting it.)




```
<EditText
    android:id="@+id/editCoin"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Enter coin (e.g., bitcoin)"
    android:padding="12dp"
    android:background="@android:drawable/edit_text" />

<!-- Button -->
<Button
    android:id="@+id/buttonQuote"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Get Quote"
    android:layout_marginTop="18dp" />

<!-- Results Section -->
<TextView
    android:id="@+id/textResultTitle"
```

1b. Requires input from the user

The app requires the user to input:

- coin (e.g., bitcoin, dogecoin)

The quote request only runs after the user has entered these values and taps the button.

1c. Makes an HTTP request to the web service

When the user taps “Get Quote”, the app builds a URL pointing to my deployed web service in GitHub Codespaces:

GET "https://improved-meme-5gpj6g5v4gvrhv9jp-8080.app.github.dev/"

```
public class MainActivity extends AppCompatActivity {

    // 🔥 Localhost for Android emulator (10.0.2.2 = host machine)
    1 usage
    private static final String BASE_URL =
            "https://improved-meme-5gpj6g5v4gvrhv9jp-8080.app.github.dev/";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

The request is done using a background networking component (e.g., HttpURLConnection / OkHttp) to avoid blocking the UI thread.

The app also sets custom headers like:

- X-Device-Model – device model string from the Android device
- X-Device-SDK – Android SDK version

These headers are later logged and surfaced in the dashboard via MongoDB.

1d. Receives and parses a JSON reply

The web service returns a **JSON** object with the following schema (as implemented in QuoteServlet):

```
{
 "coin": "bitcoin",
 "vs": "usd",
```

"price": 57893.12,

"change24h_pct": -1.23,

"volatility": "moderate",

"asOf": "2025-11-28T03:10:00Z"

}

```
String volatility = classifyVolatility(change);

Map<String, Object> out = new LinkedHashMap<>();
out.put("coin", coin);
out.put("vs", vs);
out.put("price", price);
out.put("change24h_pct", change);
out.put("volatility", volatility);
out.put("asOf", Instant.now().toString());
```

On the client, the app parses this response using a JSON library (JSONObject / similar) and extracts:

- price
- change24h_pct
- volatility
- asOf

On the client, the app parses this response using a JSON library (JSONObject / similar) and extracts:

- price
- change24h_pct
- volatility
- asOf

1e. Displays new information to the user

After parsing the JSON, the app updates the UI with:

- The current price (e.g., “Price: 57893.12 USD”)

- The 24h percentage change (e.g., “24h Change: −1.23%”)
- The volatility classification (calm, moderate, or volatile)
- The timestamp from asOf

If there is an error (e.g., invalid coin, network issue), the app shows an error message instead of leaving stale data on screen.

1f. Repeatable usage

The app is fully repeatable:

- The Activity stays open.
- The user can change the coin or currency and tap the button again.
  Each tap triggers a new request without restarting the app or Activity.

This satisfies the repeatability requirement.

2. Implement a web service

My web service is implemented entirely with **Jakarta Servlets**, not JAX-RS, and packaged as ROOT.war running on Tomcat in a Docker container inside GitHub Codespaces.

The main class is ds.task2.QuoteServlet.

2a. Simple API

The service exposes a simple, servlet-based API:

- GET /api/health
- GET /api/quote?coin=<coin>&vs=<vs>

/api/health is a lightweight health-check and does **not** log to MongoDB.
/api/quote is the main endpoint used by the Android app and is logged.

```
String url = "https://api.coingecko.com/api/v3/simple/price?ids=" +
        URLEncoder.encode(coin, StandardCharsets.UTF_8) +
        "&vs_currencies=" + URLEncoder.encode(vs, StandardCharsets.UTF_8) +
        "&include_24hr_change=true";
```

2b. Receives HTTP request from Android

QuoteServlet.doGet():

- Reads the full request path (/api/health vs /api/quote).

For /api/quote, it reads query parameters:

String coin = opt(req.getParameter("coin")).toLowerCase(Locale.ROOT);
String vs = opt(req.getParameter("vs")).toLowerCase(Locale.ROOT);

- Validates that both coin and vs are non-blank.

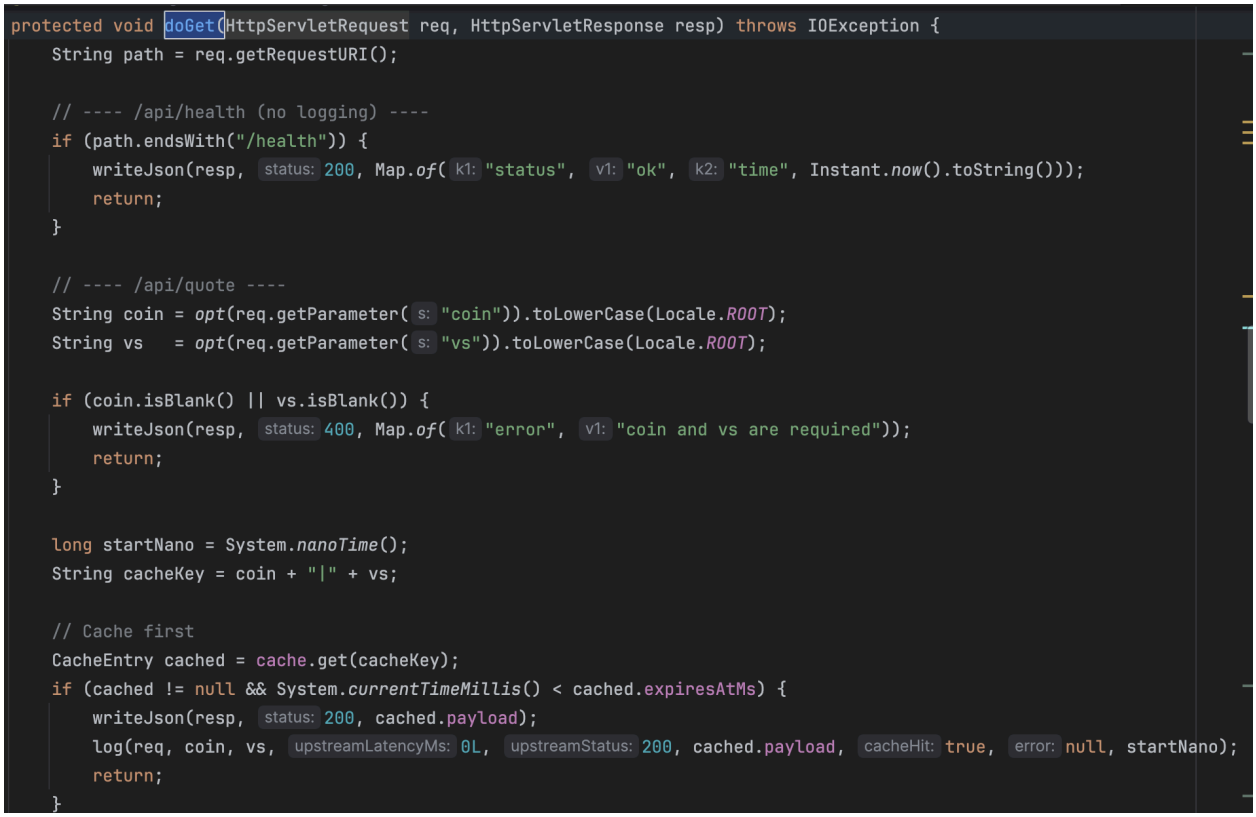
If either is missing, it returns HTTP 400 with JSON:

{ "error": "coin and vs are required" }

  -

- Captures device metadata from headers:
    - X-Device-Model
    - X-Device-SDK

These device headers are used only for logging and dashboard analytics.



```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    String path = req.getRequestURI();

    // ---- /api/health (no logging) ----
    if (path.endsWith("/health")) {
        writeJson(resp, status: 200, Map.of( k1: "status",  v1: "ok",  k2: "time", Instant.now().toString()));
        return;
    }

    // ---- /api/quote ----
    String coin = opt(req.getParameter( s: "coin")).toLowerCase(Locale.ROOT);
    String vs   = opt(req.getParameter( s: "vs")).toLowerCase(Locale.ROOT);

    if (coin.isBlank() || vs.isBlank()) {
        writeJson(resp, status: 400, Map.of( k1: "error",  v1: "coin and vs are required"));
        return;
    }

    long startNano = System.nanoTime();
    String cacheKey = coin + "|" + vs;

    // Cache first
    CacheEntry cached = cache.get(cacheKey);
    if (cached != null && System.currentTimeMillis() < cached.expiresAtMs) {
        writeJson(resp, status: 200, cached.payload);
        log(req, coin, vs, upstreamLatencyMs: 0L, upstreamStatus: 200, cached.payload, cacheHit: true, error: null, startNano);
        return;
    }
```

2c. Business logic and 3rd-party API usage

The business logic of /api/quote:

1. **In-memory caching**
   The servlet keeps a simple in-memory cache of responses keyed by coinlvs with a TTL (default 15 seconds, configurable via CACHE_TTL_MS init-param).
    - If the requested combination is in the cache and still fresh, the service returns the cached JSON directly and logs cacheHit = true.

Third-party API call (CoinGecko)
On a cache miss, it constructs a URL to the **CoinGecko** “simple price” API:

https://api.coingecko.com/api/v3/simple/price

?ids=<coin>
&vs_currencies=<vs>
&include_24hr_change=true

This is done using Java's HttpClient:

```
HttpResponse<String> response = httpClient.send(
  HttpRequest.newBuilder(URI.create(url)).GET().build(),
  HttpResponse.BodyHandlers.ofString()
);
```

2. **JSON parsing and volatility classification**
   If CoinGecko returns HTTP 200:
   - It parses the JSON using Jackson's ObjectMapper.
   - It extracts:
     - price = root.get(coin).get(vs).asDouble()
     - change24h_pct from <vs>_24h_change
   - It calls classifyVolatility(change):
     - |change| < 2.0 → "calm"
     - 2.0 ≤ |change| < 5.0 → "moderate"
     - |change| ≥ 5.0 → "volatile"

Error conditions (upstream failure)

If the upstream call fails (non-200 status, invalid JSON structure, or exceptions), the servlet returns HTTP 502 with a JSON body:

{ "error": "<description>" }

3. while also logging the error and upstream status into MongoDB.

2d. Returns JSON with only needed fields

On success, the servlet returns HTTP 200 and a **minimal JSON** object:

```
{
  "coin": "bitcoin",
  "vs": "usd",
  "price": 57893.12,
  "change24h_pct": -1.23,
  "volatility": "moderate",
  "asOf": "2025-11-28T03:10:00Z"
}
```

This includes exactly the fields the Android app needs:

- The coin and vs identifiers
- The price
- The 24h percent change
- A human-friendly volatility classification
- A timestamp

It does **not** return the entire raw CoinGecko response, so the phone does not have to do extra filtering or processing.

4. Logged Information (MongoDB)

4. Logged Information (MongoDB)

Each /api/quote request generates a detailed log entry in the cryptoapp.logs MongoDB collection. I log **more than 6 fields**, covering request, upstream, and response data.

The log() method in QuoteServlet creates a Document with:

- **Top-level fields**
  - ts – ISO timestamp when the server processed the request.
  - clientIp – req.getRemoteAddr().
  - serverLatencyMs – total server time (from request start to finish).
  - status – final status code returned to the client (200 or 502).
  - error – error string if something failed, otherwise null.
  - cacheHit – true if served from cache, false if upstream call was made.
  - appVersion – "1.0.0".

- **Device information**
  - device.model – from X-Device-Model header.
  - device.sdk – from X-Device-SDK header.

- **Request details**
  - req.coin – the coin requested.
  - req.vs – the vs currency.

- **Upstream details**
  - upstream.provider – "coingecko".
  - upstream.endpoint – "/simple/price".
  - upstream.latencyMs – round-trip time for the CoinGecko call.
  - upstream.status – HTTP status code from CoinGecko.

- **Result summary**

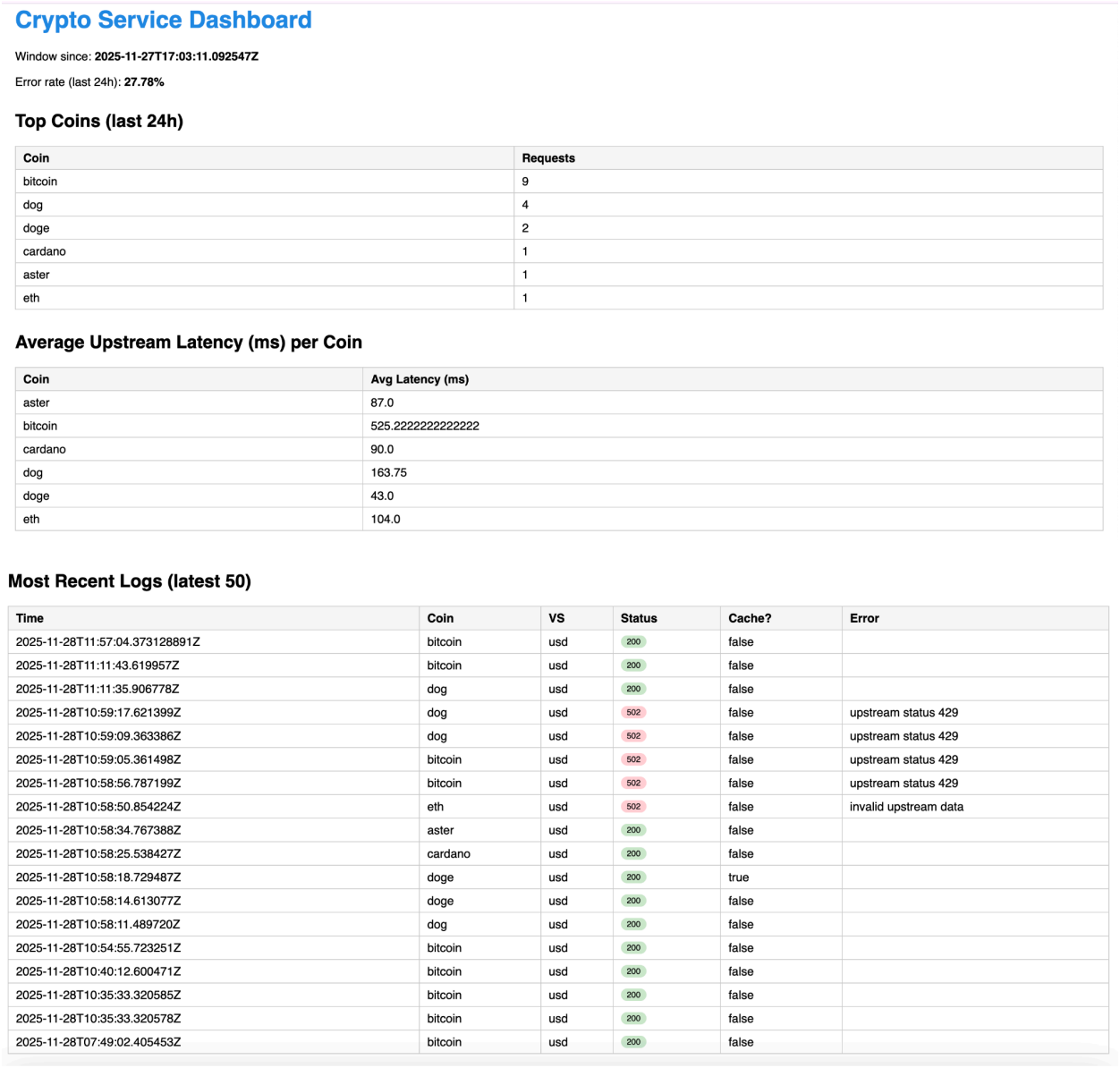
  - result – the JSON payload that was returned to the client (as a nested document) when successful, or null on errors.

This clearly satisfies the requirement to log at least 6 pieces of information, spanning:

- The mobile request
- The 3rd-party API call and response

- The reply back to the phone

The dashboard endpoints themselves (/dashboard) are *not* logged, as required.



Crypto Service Dashboard

Window since: **2025-11-27T17:03:11.092547Z**

Error rate (last 24h): **27.78%**

Top Coins (last 24h)

Coin	Requests
bitcoin	9
dog	4
doge	2
cardano	1
aster	1
eth	1

Average Upstream Latency (ms) per Coin

Coin	Avg Latency (ms)
aster	87.0
bitcoin	525.2222222222222
cardano	90.0
dog	163.75
doge	43.0
eth	104.0

Most Recent Logs (latest 50)

Time	Coin	VS	Status	Cache?	Error
2025-11-28T11:57:04.373128891Z	bitcoin	usd	200	false	
2025-11-28T11:11:43.619957Z	bitcoin	usd	200	false	
2025-11-28T11:11:35.906778Z	dog	usd	200	false	
2025-11-28T10:59:17.621399Z	dog	usd	502	false	upstream status 429
2025-11-28T10:59:09.363386Z	dog	usd	502	false	upstream status 429
2025-11-28T10:59:05.361498Z	bitcoin	usd	502	false	upstream status 429
2025-11-28T10:58:56.787199Z	bitcoin	usd	502	false	upstream status 429
2025-11-28T10:58:50.854224Z	eth	usd	502	false	invalid upstream data
2025-11-28T10:58:34.767388Z	aster	usd	200	false	
2025-11-28T10:58:25.538427Z	cardano	usd	200	false	
2025-11-28T10:58:18.729487Z	doge	usd	200	true	
2025-11-28T10:58:14.613077Z	doge	usd	200	false	
2025-11-28T10:58:11.489720Z	dog	usd	200	false	
2025-11-28T10:54:55.723251Z	bitcoin	usd	200	false	
2025-11-28T10:40:12.600471Z	bitcoin	usd	200	false	
2025-11-28T10:35:33.320585Z	bitcoin	usd	200	false	
2025-11-28T10:35:33.320578Z	bitcoin	usd	200	false	
2025-11-28T07:49:02.405453Z	bitcoin	usd	200	false	

5. Storing Logs in MongoDB (Cloud)

5. Storing Logs in MongoDB (Cloud)

The web service uses **MongoDB Atlas** as a cloud NoSQL database:

- Connection string is read from an environment variable ATLAS_URI (or another env name specified via servlet ATLAS_URI_ENV init-param).
- Database name: cryptoapp
- Collection for logs: logs

In QuoteServlet.init():

```
String envVarName = Optional.ofNullable(config.getInitParameter("ATLAS_URI_ENV"))
        .orElse("ATLAS_URI");

String uri = System.getenv(envVarName);
MongoClient client = MongoClients.create(uri);
MongoDatabase db = client.getDatabase("cryptoapp");
logs = db.getCollection("logs");
```

For Task 1 testing, I also implemented a small console program (MongoAtlasDemo) that:

- Reads ATLAS_URI or a fallback URI.
- Inserts a document { text: <user input>, ts: <timestamp> } into cryptoapp.messages.
- Reads back and prints stored documents sorted by timestamp.

This confirmed that my Atlas cluster, network access, credentials, and driver configuration were working correctly before integrating Mongo into the web service.

```java
public void init(ServletConfig config) throws ServletException {
    super.init(config);

    // Read cache TTL from init-param if present
    String ttl = config.getInitParameter( s: "CACHE_TTL_MS");
    if (ttl != null && !ttl.isBlank()) {
        try {
            cacheTtlMs = Long.parseLong(ttl.trim());
        } catch (NumberFormatException ignored) {}
    }

    // Mongo URI: get env name from init-param, default ATLAS_URI
    String envVarName = Optional.ofNullable(config.getInitParameter( s: "ATLAS_URI_ENV"))
            .orElse( other: "ATLAS_URI");

    String uri = System.getenv(envVarName);
    if (uri == null || uri.isBlank()) {
        throw new ServletException("Missing MongoDB Atlas URI in env var: " + envVarName);
    }

    try {
        MongoClient client = MongoClients.create(uri);
        MongoDatabase db = client.getDatabase( s: "cryptoapp");
        logs = db.getCollection( s: "logs");
    } catch (Exception e) {
        throw new ServletException("Failed to initialize Mongo connection: " + e.getMessage(), e);
    }
}
```

6. Web-Based Operations Dashboard

6. Web-Based Operations Dashboard

The dashboard portion is implemented using:

- ds.task2.DashboardServlet (controller)
- WEB-INF/jsp/dashboard.jsp (view with HTML tables)

The dashboard is meant for desktop/laptop browsers.

6a. Unique dashboard URL

The dashboard is available at:

GET /dashboard

For example (in Codespaces):

https://improved-meme-5gpj6g5v4gvrhv9jp-8080.app.github.dev/dashboard

This URL is public once the port is set to "Public" in the GitHub Codespaces Port view.

6b. Operations analytics (at least 3)

Inside DashboardServlet.doGet(), I compute several analytics over approximately the last 24 hours:

1. **Top 10 coins by request count (last 24h)**
   Aggregation pipeline on cryptoapp.logs:
   - Match: ts >= now - 24h
   - Group by: req.coin
   - Sum: count
   - Sort: count descending
   - Limit: 10
2. Displayed as an HTML table with columns: **Coin** and **Requests**.
3. **Average upstream latency per coin (last 24h)**
   Aggregation pipeline:
   - Match: ts >= now - 24h and upstream.latencyMs exists
   - Group by: req.coin
   - Compute: average upstream.latencyMs

- Sort by coin

4. Displayed as a table: **Coin** and **Avg Latency (ms)**.

5. **Error rate in the last 24 hours**
   Calculated using simple counts:

   - total = count of logs with ts >= since
   - errors = count of logs where ts >= since and status != 200
   - errorRate = errors * 100.0 / total

6. Formatted as a percentage string and shown near the top of the page (e.g., “Error rate (last 24h): 2.00%”).

All of these are **operations-focused analytics** that help monitor:

- Popular coins
- Upstream performance

- Overall reliability

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

    // IMPORTANT: we do NOT log dashboard hits (per requirements).
    Instant since = Instant.now().minus( amountToSubtract: 24, ChronoUnit.HOURS);

    // 1) Top 10 coins by request count
    List<Document> topCoins = logs.aggregate(asList(
            Aggregates.match(Filters.gte( fieldName: "ts", since.toString())),
            Aggregates.group( id: "$req.coin", Accumulators.sum( fieldName: "count",  expression: 1)),
            Aggregates.sort(Sorts.descending( ...fieldNames: "count")),
            Aggregates.limit(10)
    )).into(new ArrayList<>());

    // 2) Avg upstream latency per coin
    List<Document> avgLatency = logs.aggregate(asList(
            Aggregates.match(Filters.and(
                    Filters.gte( fieldName: "ts", since.toString()),
                    Filters.exists( fieldName: "upstream.latencyMs",  exists: true)
            )),
            Aggregates.group( id: "$req.coin",
                    Accumulators.avg( fieldName: "avgLatencyMs",  expression: "$upstream.latencyMs")),
            Aggregates.sort(Sorts.ascending( ...fieldNames: "_id"))
    )).into(new ArrayList<>());

    // 3) Error rate last 24h
    long total = logs.countDocuments(Filters.gte( fieldName: "ts", since.toString()));
    long errors = logs.countDocuments(Filters.and(
            Filters.gte( fieldName: "ts", since.toString()),
```

6c. Displaying formatted full logs

DashboardServlet also loads the **50 most recent logs** from the last 24 hours:

```
List<Document> recent = logs.find(Filters.gte("ts", since.toString()))
        .sort(Sorts.descending("ts"))
        .limit(50)
        .into(new ArrayList<>());
```

dashboard.jsp then renders these as an HTML table:

Columns:

- **Time** – ts
- **Coin** – from req.coin
- **VS** – from req.vs
- **Status** – the final status field, with a colored badge:
    - Green badge-ok for 200
    - Red badge-error for non-200
- **Cache?** – cacheHit value (true / false)
- **Error** – error message if any

The logs are displayed in a **formatted table**, not as raw JSON or XML, satisfying the requirement.

7. Deployment to GitHub Codespaces

7a. GitHub Classroom base repository

I accepted the provided GitHub Classroom assignment, which contained:

- .devcontainer/devcontainer.json
- Dockerfile
- A sample ROOT.war
- build-and-run.sh
- README.md

I replaced the sample ROOT.war with my own web application built from my IntelliJ/Maven project.

7b. Codespace creation and Tomcat startup

I created a Codespace on the repository (Code → Codespaces → "Create codespace on master").

The devcontainer configuration builds a Tomcat-based image and copies ROOT.war into /usr/local/tomcat/webapps/.

If Tomcat is not running automatically, I can run:

./build-and-run.sh

at the repository root to:

- Build the WAR
- Build the Docker image
- Start Tomcat in the container on port 8080

7c. Port forwarding and URL

In the Codespaces "Ports" panel:

- Port 8080 is forwarded.

Its URL is something like:

https://improved-meme-5gpj6g5v4gvrhv9jp-8080.app.github.dev/

- I changed its visibility from “Private” to **“Public”**.

This URL is used:

- In a browser to test /, /api/health, /api/quote, and /dashboard.
- As the base URL for HTTP requests from the Android app.

7d. Environment variable for MongoDB

In .devcontainer/devcontainer.json, I set:

```
"containerEnv": {
  "ATLAS_URI":
"mongodb+srv://luhanhuang_db_user:nX7EJuq1bufZiMPC@cluster0.004kzja.mongodb.net/?retryWrites
=true&w=majority&appName=Cluster0";

}
```

This ensures System.getenv("ATLAS_URI") (or the configured ATLAS_URI_ENV) is available to both:

- QuoteServlet (for logging)
- DashboardServlet (for analytics queries)

This allows the containerized Tomcat instance to connect securely to my MongoDB Atlas cluster.

7e. Final verification

I verified end-to-end that:

- GET /api/health returns { "status": "ok", ... }.
- GET /api/quote?coin=bitcoin&vs=usd returns the JSON described above.
- Each quote request inserts a log document into cryptoapp.logs.
- GET /dashboard shows:
    - The time window and error rate
    - The top coins table
    - The average latency table

    - A nicely formatted recent logs table

- The Android app can call /api/quote using the public Codespaces URL and display results to the user.

Crypto Service Dashboard

Window since: **2025-11-27T17:03:11.092547Z**

Error rate (last 24h): **27.78%**

Top Coins (last 24h)

Coin	Requests
bitcoin	9
dog	4
doge	2
cardano	1
aster	1
eth	1

Average Upstream Latency (ms) per Coin

Coin	Avg Latency (ms)
aster	87.0
bitcoin	525.2222222222222
cardano	90.0
dog	163.75
doge	43.0
eth	104.0

Most Recent Logs (latest 50)

Time	Coin	VS	Status	Cache?	Error
2025-11-28T11:57:04.373128891Z	bitcoin	usd	200	false	
2025-11-28T11:11:43.619957Z	bitcoin	usd	200	false	
2025-11-28T11:11:35.906778Z	dog	usd	200	false	
2025-11-28T10:59:17.621399Z	dog	usd	502	false	upstream status 429
2025-11-28T10:59:09.363386Z	dog	usd	502	false	upstream status 429
2025-11-28T10:59:05.361498Z	bitcoin	usd	502	false	upstream status 429
2025-11-28T10:58:56.787199Z	bitcoin	usd	502	false	upstream status 429
2025-11-28T10:58:50.854224Z	eth	usd	502	false	invalid upstream data
2025-11-28T10:58:34.767388Z	aster	usd	200	false	
2025-11-28T10:58:25.538427Z	cardano	usd	200	false	
2025-11-28T10:58:18.729487Z	doge	usd	200	true	
2025-11-28T10:58:14.613077Z	doge	usd	200	false	
2025-11-28T10:58:11.489720Z	dog	usd	200	false	
2025-11-28T10:54:55.723251Z	bitcoin	usd	200	false	
2025-11-28T10:40:12.600471Z	bitcoin	usd	200	false	
2025-11-28T10:35:33.320585Z	bitcoin	usd	200	false	
2025-11-28T10:35:33.320578Z	bitcoin	usd	200	false	
2025-11-28T07:49:02.405453Z	bitcoin	usd	200	false	