Project 4 Task 2 – Crypto Monitor & Prediction Game

Siyuan Liu (sliu5)

Description: The Android client lets users track live Binance spot prices and play a prediction mini-game backed by the same price feed. A Java servlet deployed via GitHub Codespaces proxies Binance, logs every request to MongoDB Atlas, and exposes an operations dashboard that summarizes usage analytics.

Requirement 1 – Native Android Application

Project: `android-app`

a. Multiple views
`activity_main.xml` combines `TextView`, `TextInputEditText`, `MaterialButton`, `ProgressBar`, `MPAndroidChart`'s `LineChart`, and `Snackbar` elements inside a `NestedScrollView`, satisfying the "three different views" rule.

b. Requires user input
Users type the trading symbol and server URL. Additional controls on `activity_home.xml` let the user jump between the monitor and the WebView game.

c. Makes an HTTP request
`PriceRepository` builds a Retrofit client (`PriceApi#fetchPrice`) that issues a `GET /api/price?symbol=...&clientId=...` request to the servlet. The WebView game reuses the same repository when streaming quotes.

d. Parses an XML/JSON reply
The app consumes the JSON payload via Moshi into `PriceResponse { symbol, price, fetchedAt }`. The ViewModel then converts the timestamp into `Instant` objects for charting.

e. Displays new information
`MainActivity` renders the latest price, timestamp, status, and any error. The price history also animates inside the line chart, and the WebView shows the live price inside the React game UI.

f. Repeatable
`PriceViewModel` polls continuously in `startRealtimeStream()` and exposes a manual **Fetch Price** button. Users can submit arbitrary symbols and repeat without relaunching the activity.

Screenshots:

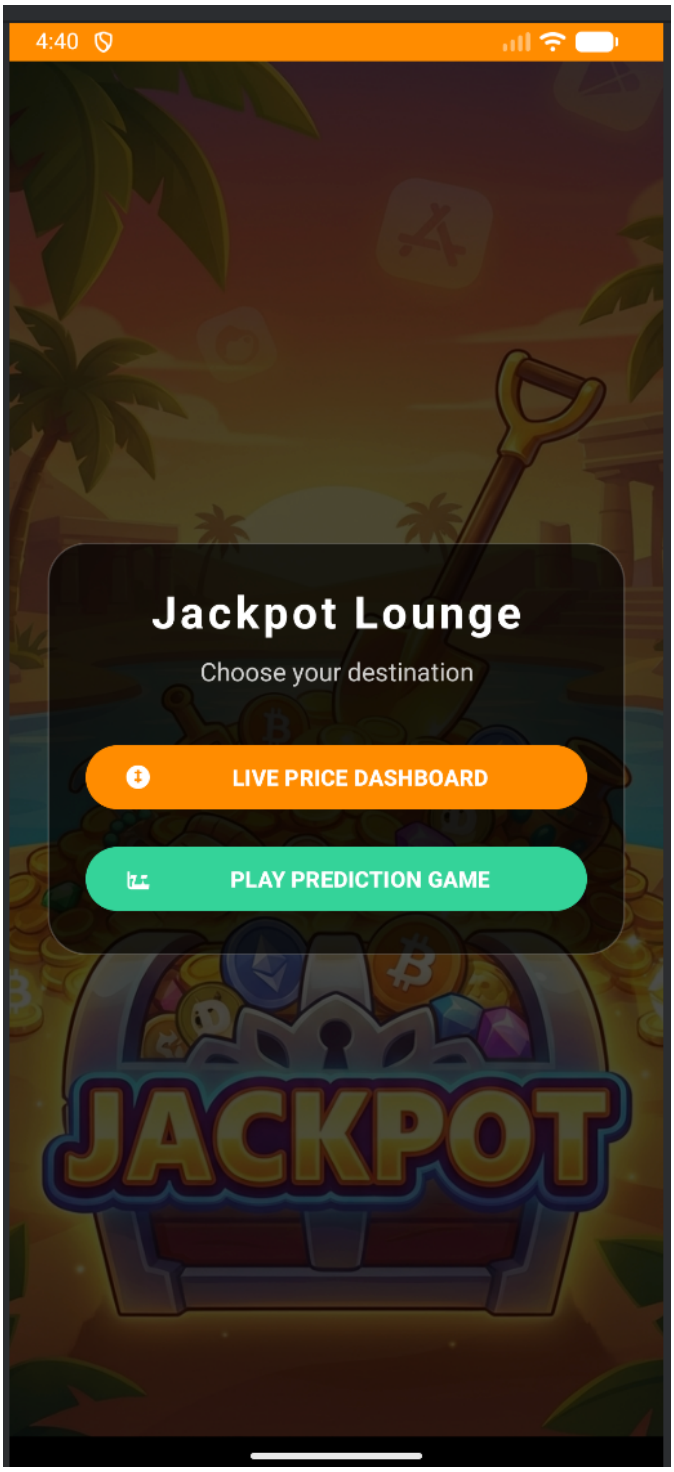


Figure 1: Home navigation screen

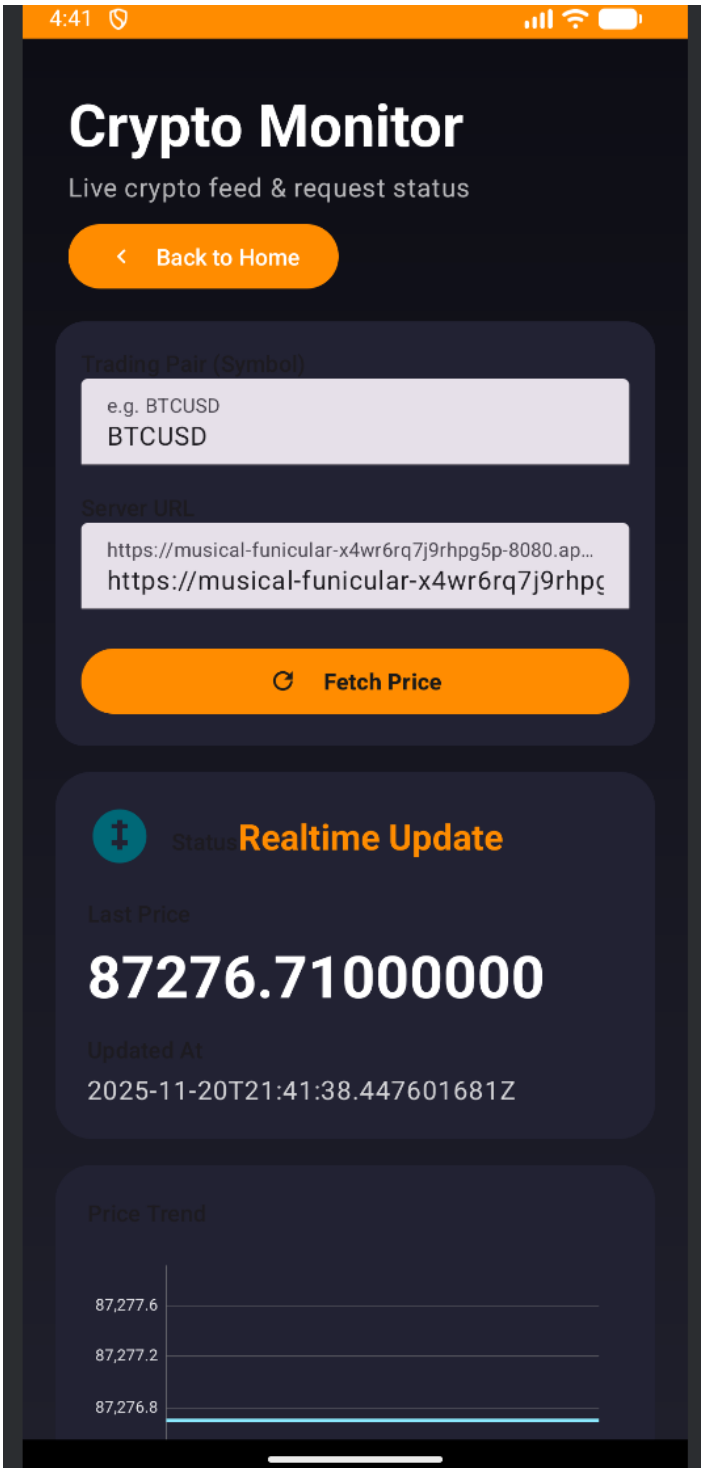


Figure 2: Monitoring workflow with live chart

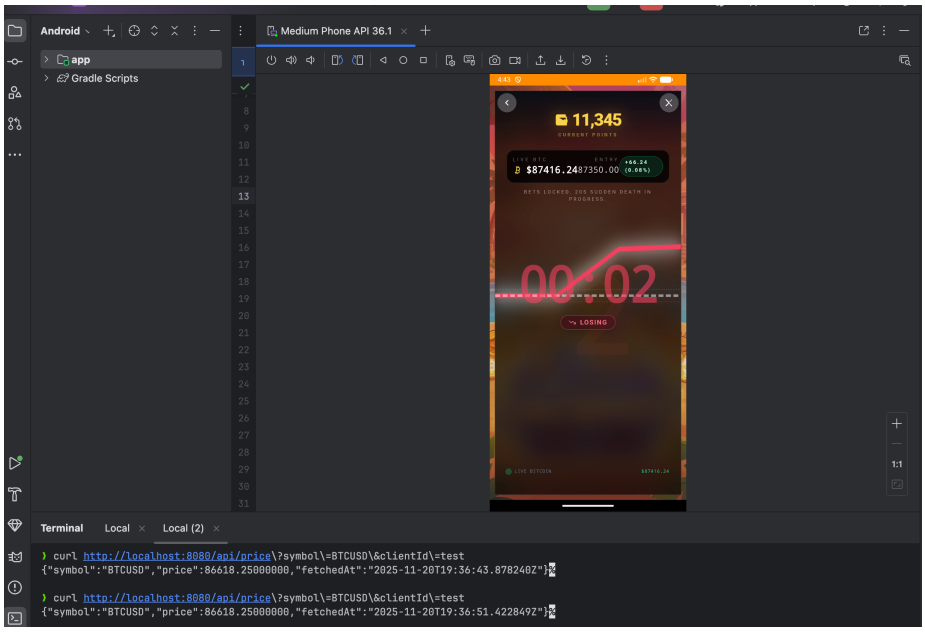


Figure 3: Prediction game fed by live prices

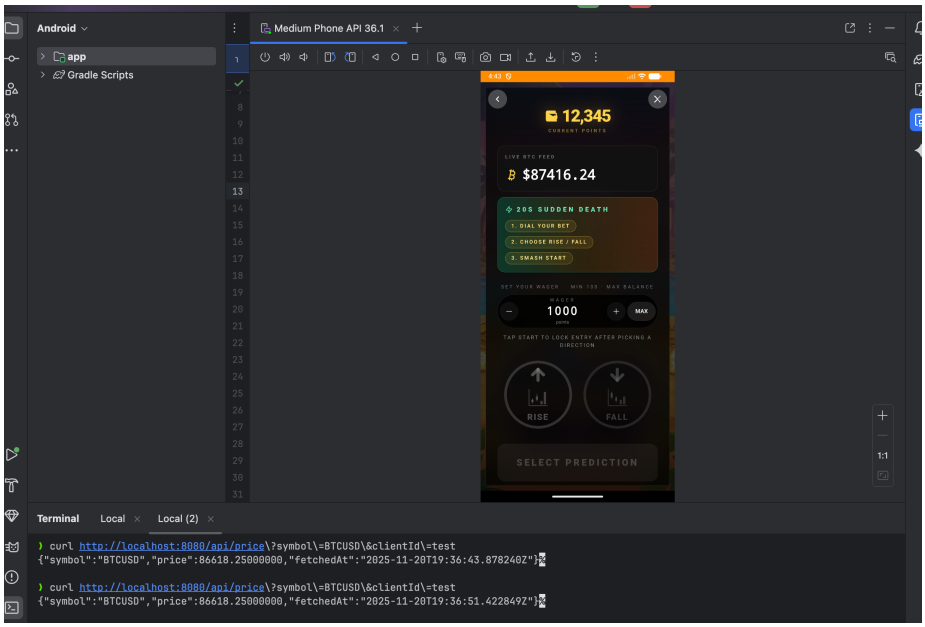


Figure 4: Jackpot win state surfaced inside the WebView game

Requirement 2 – Web Service

Project: `server`

a. Simple servlet API
`PriceServlet` is mapped to `/api/price`. `DashboardServlet` is mapped to `/dashboard` for human viewers. No frameworks beyond Servlets/JSP.

b. Receives requests from Android
The mobile client calls `/api/price` with `symbol` and `clientId` query params. The servlet reads both, normalizes the symbol, and stamps metadata (requestId, timestamps, IP).

c. Business logic & third-party API
`BinanceClient` performs the outbound HTTP call to Binance's Symbol Price Ticker endpoint and measures latency. The servlet stores the price, status, latencies, and client metadata in MongoDB before replying.

d. JSON reply
`respondWithQuote` writes JSON with just the fields the app needs (`symbol`, `price`, `fetchedAt`). Errors are returned as JSON `{ "error": "..." }` with appropriate HTTP status codes.

Figure 5: Standalone price fetch verification

Requirement 3 – Handle Error Conditions

- **Invalid mobile input:** Empty or null symbols default to `BTCUSD`; URL fields are validated via `TextInputLayout`. Retrofit failures surface via Snackbars with actionable text.
- **Server-side validation:** Servlet sanitizes/uppercases symbols, generates UUID request IDs, and wraps Binance failures in `BinanceClientException` to ensure deterministic error responses.
- **Network failures:** ViewModel catches network exceptions, stops the loading spinner, and shows the error string. WebView game throttles retries and warns the user every five failed polls.
- **Third-party API issues:** Non-200 responses and malformed Binance payloads raise `BinanceClientException` with the HTTP status and partial body for logging. The servlet returns `502` (Binance error) versus `500` (internal error).

Requirement 4 – Log Useful Information

`RequestLog` captures more than six attributes per request and explains server state: 1. `requestId`, `requestReceivedAt`, `responseSentAt` – correlate frontend and backend timings. 2. `clientId` and `clientIp` – aggregate analytics by device or network. 3. Requested `symbol` – drives the "Top Symbols" card. 4. `price` – useful for auditing prediction-game rounds. 5. Binance `statusCode`, `latencyMs`, and the queried endpoint – highlight upstream health. 6. Overall `totalLatencyMs`, `success` flag, and `errorMessage` – measure SLA and failure modes. 7. `binanceEndpoint` – documents whether Binance Global or Binance US was used. Only `/api/price` traffic is logged; dashboard page views are intentionally excluded per the spec.

Requirement 5 – Store Logs in MongoDB Atlas

- **Driver:** `mongodb-driver-sync` (v4.11.1) inside `MongoLogRepository`.
- **Atlas connection string (three shards):** `mongodb://gushi10546_db_user:suqianye10546@ac-dw50pnt-shard-00-00.igaq6gr.mongodb.net:2`
- **Database & collection:** `project4` / `requestLogs` (configurable via `MONGODB_DATABASE` and `MONGODB_COLLECTION`). `AppContextListener` creates the Mongo client at startup, shares it via `ServletContext`, and closes it cleanly when Tomcat shuts down.

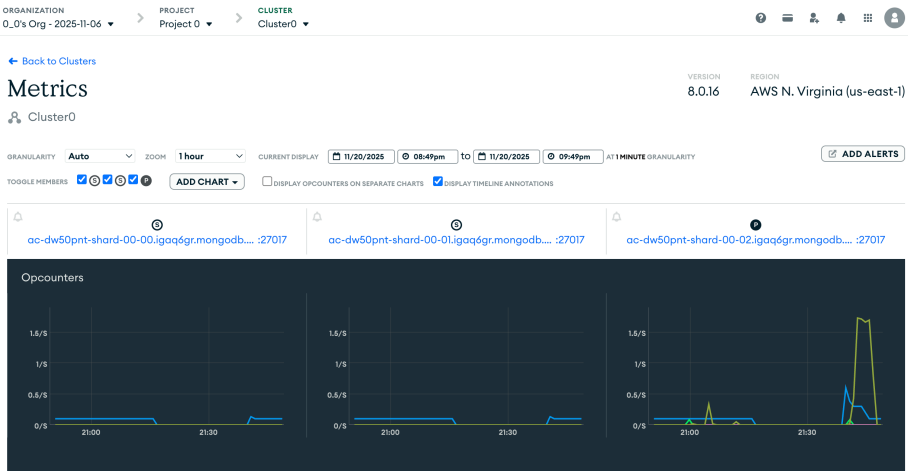


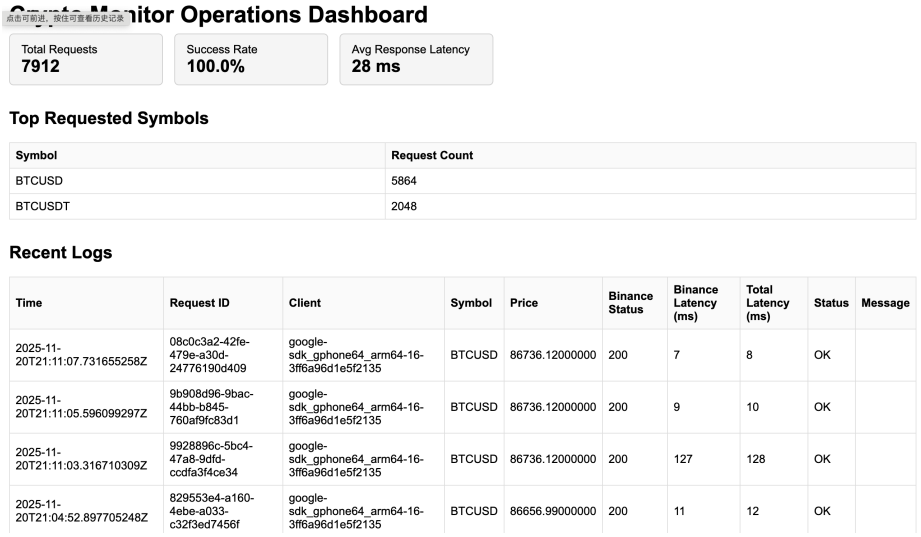
Figure 6: MongoDB Atlas collection populated with logs

Requirement 6 – Dashboard (Analytics + Logs)

URL: `/dashboard` (served from `WEB-INF/jsp/dashboard.jsp`).

Analytics shown: 1. **Total Requests** – via `MongoLogRepository#totalCount()`. 2. **Success Rate** – `successCount / totalCount` rendered as a percent. 3. **Average Response Latency** – aggregation pipeline calculating avg `totalLatencyMs`. 4. **Top Requested Symbols** – aggregated counts limited to the top five.

Formatted logs:
The JSP renders a full-width HTML table with timestamp, request ID, client identifier, symbol, price, Binance status, upstream latency, total latency, status (OK/Failed), and error text. Each row corresponds to a MongoDB document so TAs can audit individual requests without reading JSON.



itor Operations Dashboard

点击可前进，按住可查看历史记录

Total Requests	Success Rate	Avg Response Latency
7912	100.0%	28 ms

Top Requested Symbols

Symbol	Request Count
BTCUSD	5864
BTCUSDT	2048

Recent Logs

Time	Request ID	Client	Symbol	Price	Binance Status	Binance Latency (ms)	Total Latency (ms)	Status	Message
2025-11-20T21:11:07.731655258Z	08c0c3a2-42fe-479e-a30d-24776190d409	google-sdk_gphone64_arm64-16-3ff6a96d1e5f2135	BTCUSD	86736.12000000	200	7	8	OK	
2025-11-20T21:11:05.596099297Z	9b908d96-9bac-44bb-b845-760af9fc83d1	google-sdk_gphone64_arm64-16-3ff6a96d1e5f2135	BTCUSD	86736.12000000	200	9	10	OK	
2025-11-20T21:11:03.316710309Z	9928896c-5bc4-47a8-9dfd-ccdfa3f4ce34	google-sdk_gphone64_arm64-16-3ff6a96d1e5f2135	BTCUSD	86736.12000000	200	127	128	OK	
2025-11-20T21:04:52.897705248Z	829553e4-a160-4ebe-a033-c32f3ed7456f	google-sdk_gphone64_arm64-16-3ff6a96d1e5f2135	BTCUSD	86656.99000000	200	11	12	OK	

Figure 7: Dashboard analytics + formatted logs

Requirement 7 – Deployment to GitHub Codespaces

1. The repository includes `.devcontainer.json` + `Dockerfile` at the root. Codespaces builds Tomcat, copies `ROOT.war`, and launches Catalina automatically.
2. After Codespace launch, port **8080** is exposed and set to **Public** so the Android app can reach it without GitHub auth.
3. Current deployment URL: `https://musical-funicular-x4wr6rq7j9rhpg5p-8080.app.github.dev/`. This matches the default value baked into `BuildConfig.DEFAULT_BASE_URL`, so installing a fresh APK immediately targets the cloud instance.
4. `mvn -f server/pom.xml clean package` produces `ROOT.war`, which is copied to the repo before pushing to GitHub Classroom.
5. Verified from an Incognito browser plus an emulator hitting the same URL.

Figure 8: Codespaces port made public

Appendix – Screenshot Index

- `docs/snapshot/home.png` – Landing page with navigation buttons.
- `docs/snapshot/monitor.png` – Monitoring screen showing live price + chart (Req. 1/2).
- `docs/snapshot/game.png` – Prediction mini-game inside WebView.
- `docs/snapshot/price_fetch.png` – Standalone servlet response for `/api/price`.
- `docs/snapshot/dashboard.png` – Dashboard analytics and formatted logs (Req. 6).
- `docs/snapshot/codespace_port.png` – Codespaces Ports tab with 8080 set to Public (Req. 7).
- `docs/snapshot/jackpot.png` – Bonus view of the jackpot alert used in the WebView game.
- `docs/snapshot/mongodb.png` – Atlas collection showing persisted request logs (Req. 5).

These files now live in `docs/snapshot/` and are ready for inclusion when exporting to PDF.