ChatGPT

Urban Points Lebanon – Copilot Completion Prompt (Full-Stack 100 %)

You are acting as a **senior full-stack engineer and systems architect** tasked with finishing the Urban Points Lebanon project to production-ready status. This repository already contains a substantial codebase; your mission is to complete everything that is partial, implement what is missing and leave **no gaps**. Use the following detailed instructions as a roadmap. **Base all work solely on existing code**—do not invent features that aren't present in the repo. If something is referenced but not implemented, state that clearly and implement it if appropriate.

Guidance for Copilot Chat

1. **Use the entire codebase for context:** Include `#codebase` in your prompt to let Copilot search the repository and identify relevant files, functions and patterns.
2. **Explicitly reference files and folders:** When asking Copilot to update or create code, provide file paths via `#path/to/file` so it knows exactly where changes belong. Use the context picker or drag-and-drop to add multiple files or directories.
3. **Follow existing patterns:** Reuse established coding conventions, naming schemes and architectures found in the repository. For example, Cloud Functions follow the pattern in `src/backend/firebase-functions/src/*`, Flutter UIs follow widget structures under `source/apps/mobile-*/lib/screens`, and Next.js admin pages live in `source/apps/web-admin/pages/admin`.
4. **Break work into discrete tasks:** Each missing or partial feature should be tackled individually. Provide clear goals (e.g., "implement subscription payment UI in mobile apps") and reference relevant files.
5. **Avoid assumptions:** If the codebase lacks a feature or reference, mark it as **NOT FOUND** and do not fabricate an implementation unless the roadmap explicitly calls for it.
6. **Iterate and validate:** After implementing a feature, run tests and linting (where available) to ensure correctness. Use `flutter test`, `npm run test`, or the applicable commands in the repository.

Roadmap – close every gap

The following tasks are derived from the comprehensive reality map. Complete each subsystem so the platform reaches 100 % functionality. For each task, cite relevant files with `#` and provide necessary implementations.

1. Payments & subscriptions

- **Enable Stripe integration:** Configure Secret Manager variables for `STRIPE_ENABLED`, `STRIPE_SECRET_KEY` and `STRIPE_WEBHOOK_SECRET` in Firebase functions. Update `stripe.ts` to read secrets instead of hard-coded values, and remove the feature flag guard.
- **Implement payment UIs:** In the customer and merchant Flutter apps, complete `billing_screen.dart` to allow users to subscribe, purchase points and manage payment methods. Use the `flutter_stripe` package to present the Stripe Payment Sheet. Reference

existing services (e.g. `stripe_client.dart`) and call the `initiatePaymentCallable` from the backend. After payment, update Firestore `subscription_status` and show receipts.
- **Finish auto-renewal logic:** In `subscriptionAutomation.ts`, replace the TODO that returns `paymentSuccess = true` with real Stripe API calls. Charge the saved payment method, update subscription expiration dates and handle failures (e.g., notify users, set `subscription_status=past_due`).
- **Implement payment webhooks:** Uncomment and complete webhook handlers in `paymentWebhooks.ts` to process events from Stripe (payment succeeded/failed, subscription updated/deleted). Ensure idempotency and update Firestore accordingly. Register the webhook endpoint with Stripe.

2. Schedulers & automation

- **Enable Cloud Scheduler:** Uncomment the disabled scheduled function exports in `index.ts` and deploy them. Create scheduler jobs (in `phase3Scheduler.ts` and `subscriptionAutomation.ts`) for:
- `enforceMerchantCompliance` – run daily and hide merchants with <5 active offers until they comply.
- `sendExpiryReminders` – send reminders for expiring subscriptions and points.
- `processSubscriptionRenewals` – perform auto-renewal (see above).
- `sendBatchNotification` – schedule marketing campaigns.
- **Implement compliance UI:** Add a screen to the merchant app showing current offer count, compliance status and warnings if below the threshold. Use Firestore fields (e.g., `is_visible_in_catalog`) to inform merchants.

3. Authentication & SMS

- **Wire phone/OTP login:** Integrate a Lebanese SMS gateway into `sms.ts` and replace the placeholder `sendSMS()` implementation. Add phone/OTP login flows in both mobile apps (customer and merchant). Include screens for entering a phone number, receiving an OTP and verifying it via the `verifyOTP` callable.

4. Internationalisation & accessibility

- **Implement Arabic localisation:** Introduce a localisation package (e.g., `intl` for Flutter) and add Arabic translations for all user-facing strings in the mobile apps and web admin. Create language selector components. Update backend responses to include Arabic descriptions where needed.

5. Analytics & reporting

- **Merchant analytics:** Expand the merchant dashboard to show time-series metrics (redemptions per day, top offers, customer demographics if available). Create new callables if necessary to aggregate metrics from Firestore.
- **Admin analytics:** Build a new page in `source/apps/web-admin/pages/admin/analytics.tsx` that displays global metrics (daily redemptions, points awarded, subscription revenue, active merchants/customers). Call the existing `calculateDailyStats` function and format results. Include CSV export.

6. Compliance, GDPR & security

- **Data export & deletion UI:** Implement front-end screens in both mobile apps where users can request a copy of their data or permanent deletion. Wire these to the existing but disabled `privacy.ts` functions and ensure authentication checks. Admins should have a separate interface for handling export/deletion requests.
- **Audit logging:** Introduce a new collection (e.g., `audit_logs`) and middleware to record admin actions, user redemptions, subscription changes and payment events. Build a basic viewer in the admin dashboard.
- **Rate limiting:** Deploy Cloud Functions rate limiting rules (currently present but not enabled). Create UI feedback when rate limits are hit.

7. DevOps & infrastructure

- **Create CI/CD pipeline:** Add a `.github/workflows/deploy.yml` that runs tests, builds functions, and deploys to staging/production projects with manual approval. Use Firebase CLI for functions and Firestore rules, and Flutter build commands for mobile apps.
- **Monitoring & alerting:** Integrate Google Cloud Logging and Monitoring in Cloud Functions (use structured logging, correlation IDs). Create dashboards for error rates, latency and job executions. Configure alerting policies (e.g., high error rate, scheduler failures) to Slack or email.
- **Backup automation:** Write a scheduled job or script to export Firestore to Cloud Storage regularly and verify backup integrity.
- **Secret management:** Store all API keys and secrets in Secret Manager. Refactor functions to read from secrets instead of code.

8. Store readiness & release

- **Android & iOS signing:** Configure signing in `android/app/build.gradle` and create the necessary keystore. For iOS, add an `ios/` folder with an Xcode project, configure bundle identifiers, signing certificates and provisioning profiles.
- **Legal pages:** Host privacy policy and terms of service; link them in the apps and admin portal.
- **Crash reporting:** Integrate Sentry or Crashlytics and ensure DSN/keys are loaded from secrets.
- **Referral system:** Implement the `referred_by` field logic. Add UI where customers can enter a referral code and grant bonus points to both referrer and referee.

9. Documentation & testing

- **Write integration tests:** For each critical flow (QR redemption, subscription purchase, auto-renewal, GDPR requests), create integration tests using Firebase emulators and Flutter integration testing. Increase backend test coverage, especially for new payment and scheduler logic.
- **Update documentation:** Extend `docs/` with a deployment guide, API reference for new callables, monitoring setup instructions and incident response runbook. Ensure the README reflects how to run the project locally and in staging.

Deliverables

For each task above: 1. Update or create the relevant code files and tests in the repository. Use proper branching and commit messages if working in a version control context. 2. Ensure all linting and unit tests pass (`npm run lint && npm run test` for backend; `flutter analyze && flutter test` for mobile). Fix any failing tests. 3. Provide a final summary describing what was implemented and

confirming that no gaps remain. If a feature is marked **NOT FOUND** because it isn't referenced anywhere, note that explicitly.

This roadmap, when executed faithfully, will bring the Urban Points Lebanon project to full feature parity and production readiness.
