Litera.ai

Replit Completion Guide

Tactical steps and AI prompts to harden the demo into a production-ready pilot environment

February 2026 | Startup Garage

1. Environment & Secrets Configuration

Before any code changes, lock down the environment. Every step below happens in the Replit Secrets panel (lock icon in the left sidebar, or Tools > Secrets).

1.1 Required Secrets

Open Replit > Tools > Secrets. Add each of the following key-value pairs. If a key already exists, verify its value matches what is listed below.

`SESSION_SECRET` [CRITICAL]

Generate: run node -e "console.log(require('crypto').randomBytes(64).toString('hex'))" in the Replit shell, paste the output as the value

`DATABASE_URL` [Verify]

Should already exist if you provisioned PostgreSQL via Replit. Verify it starts with postgresql://

`OPENAI_API_KEY` [Required]

Your OpenAI API key starting with sk-. Get from platform.openai.com > API Keys

`RESEND_API_KEY` [Required]

Your Resend API key for transactional email. Get from resend.com/api-keys

`RESEND_FROM_EMAIL` [Required]

The verified sender email for Resend, e.g. care@yourdomain.com or onboarding@resend.dev for testing

`NODE_ENV` [Set on deploy]

Set to production for deployed app. Set to development when actively coding.

`REPLIT_DEV_DOMAIN` [Auto-set]

Replit auto-sets this. Verify it exists. Used for generating patient portal links in emails.

Replit AI Prompt — Verify Secrets

Check that my Replit environment has the following secrets configured: SESSION_SECRET, DATABASE_URL, OPENAI_API_KEY, RESEND_API_KEY, RESEND_FROM_EMAIL. In server/index.ts, remove the hardcoded fallback for SESSION_SECRET so the app fails fast if the secret is missing. The line to change is: secret: process.env.SESSION_SECRET || "litera-ai-secret-

key-change-in-production" — remove the || fallback so it becomes: secret: process.env.SESSION_SECRET (and add a startup check that throws if it is undefined).

2. Database Persistence & Seed Script Safety

2.1 Problem

The current seed.ts script deletes ALL data every time it runs (lines 18-22 call db.delete on every table). If seed.ts is in the startup flow or accidentally run in production, all patient data is destroyed.

2.2 Steps

1. Open server/seed.ts. Add an environment variable guard at the top of the seed() function, immediately after the console.log:

```
if (process.env.ALLOW_SEED !== 'true') { console.error('SEED BLOCKED: Set ALLOW_SEED=true to run. This destroys all data.'); process.exit(1); }
```

2. Verify that your package.json scripts do NOT include seed in the start or build commands. The seed should only run via an explicit npm run seed or tsx server/seed.ts command.
3. Run Drizzle migrations on every deploy to apply schema changes without touching data:

```
npx drizzle-kit push
```

4. After the pilot starts and real patient data exists, add a database backup step. In the Replit shell, you can dump the database with:

```
pg_dump $DATABASE_URL > backup_$(date +%Y%m%d).sql
```

Replit AI Prompt — Seed Safety

In server/seed.ts, add an environment variable guard so the seed script only runs when ALLOW_SEED=true is set. At the top of the seed() function, before any db.delete calls, add: if (process.env.ALLOW_SEED !== 'true') { console.error('SEED BLOCKED. Set ALLOW_SEED=true. This destroys all data.'); process.exit(1); }. Also check package.json and .replit to make sure seed.ts is not called automatically on start or build.

3. Authentication Hardening

3.1 Session Security

The session configuration in server/index.ts needs the following changes for production:

1. Remove the hardcoded session secret fallback (covered in Section 1).
2. Set cookie.secure dynamically based on the environment. Replit serves over HTTPS in production, so secure: true is correct for deploys. But during local dev, you may need secure: false.
3. Add sameSite: 'lax' to the cookie config to prevent CSRF attacks.
4. Add a session cleanup job or set a reasonable maxAge. Currently set to 24 hours, which is fine for a pilot.

Replit AI Prompt — Session Hardening

In server/index.ts, update the session configuration: 1) Remove the fallback string for SESSION_SECRET and throw an error if process.env.SESSION_SECRET is undefined. 2) Add sameSite: 'lax' to the cookie configuration. 3) Make sure cookie.secure is set to true when NODE_ENV is 'production' and false otherwise. 4) Add httpOnly: true (already present, verify). The resulting cookie config should look like: cookie: { secure: process.env.NODE_ENV === 'production', httpOnly: true, sameSite: 'lax', maxAge: 24 * 60 * 60 * 1000 }

3.2 Frontend Session Expiration Handling

When a session expires, the API returns 401. The frontend should catch this and redirect to the login page instead of showing a broken state.

Replit AI Prompt — Session Expiry Redirect

In client/src/lib/queryClient.ts (or wherever apiRequest is defined), add a global response interceptor: if any API response returns status 401, redirect the browser to /login using window.location.href = '/login'. This should apply to all API calls made through the query client. Also, in the React Query defaultOptions, add an onError handler that checks for 401 status and redirects. This ensures that if a clinician or admin's session expires mid-use, they see the login page rather than a broken dashboard.

3.3 Password Management

The seed script creates demo users with password123. For the pilot, you need to either:

- Option A: Create a password change endpoint so clinicians and admins can set their own passwords after first login.
- Option B: Pre-set strong passwords in the seed script and distribute them securely to the care team.

Option B is fine for a small pilot. Option A is better if you want the care team to manage their own credentials.

Replit AI Prompt — Password Change (Option A)

Add a POST /api/auth/change-password endpoint in server/routes.ts that requires authentication (use the requireAuth middleware). It should accept { currentPassword, newPassword } in the request body. Validate that currentPassword matches the stored hash using bcrypt.compare, then hash the newPassword with bcrypt.hash (salt rounds 10) and update the user record. Return 200 on success, 401 if current password is wrong. Also add a simple password change form accessible from the clinician and admin dashboards (a modal or settings page).

4. Clinician ↔ Admin Dashboard Sync

4.1 Current State

Both dashboards read from the same PostgreSQL database, so data is always consistent. The gap is that neither dashboard refreshes automatically when the other makes a change. A clinician sends a care plan, but the admin only sees it after a manual page refresh.

4.2 Solution: Polling (Pilot-Appropriate)

For a pilot with a small team, polling every 30 seconds is the right approach. WebSockets add complexity without meaningful benefit at this scale.

Replit AI Prompt — Auto-Refresh Dashboards

In both the clinician dashboard (client/src/pages/clinician-dashboard.tsx) and admin dashboard (client/src/pages/admin-dashboard.tsx), add automatic polling to the main data query. For the clinician dashboard, the useQuery for /api/care-plans should include refetchInterval: 30000 (30 seconds). For the admin dashboard, add the same to whatever queries fetch care plans and alerts. This ensures that when a clinician sends a care plan, the admin sees it within 30 seconds without a manual refresh, and vice versa. Also add refetchOnWindowFocus: true so the data refreshes immediately when a user switches back to the tab.

5. Email Delivery Setup

5.1 Resend Configuration

The app uses Resend for transactional email (sending care plan links and check-in reminders to patients). The following steps are required to get email actually delivering:

1. Create a Resend account at resend.com if you do not have one.
2. Navigate to resend.com/api-keys and create an API key. Copy it to the RESEND_API_KEY Replit secret.
3. For testing: Use the free onboarding@resend.dev sender. Emails can only be sent to the account owner's email address.
4. For pilot: Add and verify your own domain at resend.com/domains. This requires adding DNS records (SPF, DKIM, DMARC) to your domain provider. Resend provides the exact records.
5. Set RESEND_FROM_EMAIL in Replit secrets to the verified sender address.
6. Verify that server/services/resend.ts uses process.env.RESEND_FROM_EMAIL for the from field.

Replit AI Prompt — Verify Email Configuration

In server/services/resend.ts (or wherever sendCarePlanEmail and sendCheckInEmail are defined), verify that: 1) The Resend client is initialized with process.env.RESEND_API_KEY. 2) The 'from' field uses process.env.RESEND_FROM_EMAIL. 3) The patient portal URL uses process.env.REPLIT_DEV_DOMAIN to generate the correct base URL. 4) Add error logging that includes the Resend error response body, not just the error message. 5) If RESEND_API_KEY is not set, log a warning on startup rather than crashing.

6. Patient Portal Hardening

6.1 Patient Login Flow Verification

The patient login flow already exists in the codebase. The sequence is:

1. Patient receives email with a link: https://{domain}/p/{accessToken}
2. patient-portal.tsx renders at that route and asks for year of birth
3. Frontend calls POST /api/patient/{token}/verify with { yearOfBirth }
4. Server validates YOB against the patient record, rate-limits to 3 attempts
5. On success, the care plan is displayed in the patient's preferred language

Verify this flow works end-to-end by testing with one of the seeded patients.

Replit AI Prompt — Patient Portal Audit

Review client/src/pages/patient-portal.tsx and verify the following: 1) The page shows a year-of-birth input and a submit button before displaying any care plan content. 2) On failed verification, show the number of remaining attempts and the error message from the API. 3) On lockout (429 response), show a clear message that the patient should try again in 15 minutes or contact their care team. 4) On successful verification, display the translated care plan content. 5) The check-in traffic light buttons (green/yellow/red) call POST /api/patient/{token}/check-in and show a confirmation. 6) The page is responsive and looks good on mobile screen widths (375px and 414px). If any of these are missing or broken, fix them.

6.2 Clinician Preview vs. Patient Portal Separation

Clinicians should be able to preview the patient portal without mutating data. The demo token system handles this.

Replit AI Prompt — Demo Token Preview Mode

Verify that the clinician 'View as Patient' button in clinician-dashboard.tsx works as follows: 1) It calls POST /api/care-plans/{id}/demo-token to get a time-limited demo token. 2) It opens the patient portal URL with a ?demo=1&demoToken={token} query parameter. 3) In patient-portal.tsx, when demo=1 is in the URL, the page skips the year-of-birth verification and instead validates the demo token via POST /api/patient/{token}/validate-demo. 4) In demo mode, check-in responses should NOT be saved to the database. Add a visual banner at the top of the page in demo mode that says 'Clinician Preview Mode — Actions are not saved.' If any of this is not implemented, implement it.

7. Check-In Reminder System

7.1 Current Architecture

The check-in system schedules a reminder 24 hours after a care plan is sent. The endpoint POST /api/internal/send-pending-check-ins finds all pending check-ins and sends reminder emails. However, this endpoint needs to be called by something — there is no automatic scheduler.

7.2 Options for the Pilot

- Option A (Simplest): Use Replit's built-in cron feature or UptimeRobot (free) to ping the endpoint every hour.
- Option B: Add a setInterval in server/index.ts that calls the check-in logic every hour. This is less reliable (resets if server restarts) but requires zero external setup.

Replit AI Prompt — Check-In Scheduler (Option B)

In server/index.ts, after the server starts listening, add a setInterval that calls the check-in sending logic every hour. Create a function called sendPendingCheckIns() that contains the same logic as the POST /api/internal/send-pending-check-ins route handler (extract the logic from routes.ts into a shared function in a new file server/services/scheduler.ts). Then in index.ts, call: setInterval(sendPendingCheckIns, 60 * 60 * 1000); and also call it once on startup with a 30-second delay: setTimeout(sendPendingCheckIns, 30000). Log when the scheduler runs and how many check-ins were sent.

8. Replit Deployment Checklist

When you are ready to deploy, follow this checklist in order:

1. **Set all Replit Secrets**

   SESSION_SECRET, DATABASE_URL, OPENAI_API_KEY, RESEND_API_KEY, RESEND_FROM_EMAIL are all set. NODE_ENV is set to production.

2. **Run database migrations**

   In the Replit shell: npx drizzle-kit push. Confirm no errors.

3. **Seed initial users (one time only)**

   In the shell: ALLOW_SEED=true npx tsx server/seed.ts. This creates the clinician and admin accounts. You should only need to do this once.

4. **Test login**

   Open the deployed URL. Log in as nurse / password123 and as admin / password123. Both should work.

5. **Test email delivery**

   Upload a mock PDF, process it, and send it to your own email. Verify the email arrives with the correct portal link.

6. **Test patient portal**

   Click the link from the email. Enter the year of birth. Verify the care plan displays correctly.

7. **Deploy**

   In Replit, click the Deploy button (rocket icon). Select 'Reserved VM' for always-on hosting. Choose the smallest instance (pilot scale).

8. **Verify deployed URL**

   The deployed app gets a .replit.app domain. Test the full flow on this URL.

9. **Set up monitoring**

   Use UptimeRobot (free) to ping your deployed URL every 5 minutes and alert you if it goes down.