Documentation

Technical and product manual for the eAkto citizen journey platform.

VERSION	EDITION	FORMAT
0.1.0	eGovPH Hackathon 2026	Searchable PDF

Prototype guidance is not official agency policy. Do not use real citizen information in a local demo.

Contents

Use the chapter bookmarks in your PDF reader or select a page number below.

SECTION 01

Get started

This section is part of the consolidated eAkto technical and product documentation.

Documentation home

Start here

What the repository contains

Area	Technology	Responsibility
`frontend/`	Flutter, Riverpod, Dio, GoRouter	Citizen-facing UI, secure local session reference, routing, and presentation state
`backend/`	FastAPI, Pydantic, SQLAlchemy	Authentication, intent assessment, workflow resolution, persistence, ownership, and integrations
`backend/app/seeds/`	Validated workflow definitions	Versioned journey questions, documents, dependencies, and action metadata
`backend/app/integrations/`	Bounded HTTP adapters	eGovAI, eGovPH SSO, eVerify, Face Liveness, and Supabase
`docs/`	Markdown	Canonical project, API, setup, security, and operating documentation
`docs_theme/`	Jinja, CSS, JavaScript	Custom eAkto MkDocs theme

> Prototype scope
>
> eAkto was built for the eGovPH Hackathon 2026. Workflow requirements are prototype data, not official agency policy. Never upload real identity documents or use real citizen information in a local demo.

Current supported workflow families

The mounted backend currently recognizes two workflow codes:

1. `POST_MARRIAGE_RECORD_UPDATE` - update selected government records after marriage.
1. `GOVERNMENT_RECORD_INFORMATION_UPDATE` - correct or update information in a specific government record without requiring a life event.

AI can interpret many conversational inputs, but a journey is created only when the backend selects a supported workflow. Ambiguous or conversational messages remain a chat response.

Documentation principles

1. Implementation is the source of truth. Endpoint lists are based on routers mounted by `backend/app/main.py`.
1. Secrets stay private. Examples use placeholders and never include provider credentials, exchange codes, access tokens, private keys, or real identity data.
1. Restricted provider material is not reproduced. eGovPH-provided partner documentation is summarized only where needed to explain eAkto's integration boundary.
1. Citizen-facing language stays clear. Technical terms are defined in the glossary.

What is eAkto?

eAkto is a citizen-journey orchestration prototype created for the eGovPH Hackathon 2026. A citizen describes what changed or what government record needs attention. eAkto interprets that message, proposes an appropriate journey when enough context exists, and presents one valid next action at a time.

The problem

Government processes are commonly organized by agency. Citizens think in life events and goals:

1 “I got married.”

1 “My National ID information is wrong.”

1 “Which documents do I need?”

eAkto translates that citizen language into a structured process without claiming that AI has verified official records.

Product experience

1 1. Sign inAuthenticate through eGovPH SSO or eVerify with Face Liveness.

2 2. Describe the situationSend a message in English, Filipino, or Taglish.

3 3. Review the interpretationConfirm the detected goal, change, records, and possible agencies.

4 4. Create the journeyThe backend resolves a versioned workflow into applicable milestones.

5 5. Complete one action at a timeAnswers and progress persist in the database.

6 6. Reuse documents safelyHigh-assurance users can upload supported documents to a private vault.

AI does not control the journey

eGovAI is used for situation assessment and optional assistance such as translation or document extraction. The backend retains control:

1 Pydantic validates the AI response.

2 Compatibility rules decide whether a workflow is supported.

3 The deterministic resolver evaluates stored declarative rules.

4 The Personal Engine persists applicable milestones and progress.

5 Provider claims remain unverified until an authoritative integration confirms them.

This separation prevents a free-form model response from directly changing journey state.

Intended users

1 Citizens navigating government updates.

1 Hackathon judges and demo operators.

1 Flutter developers extending the citizen experience.

1 Backend developers adding validated workflow families.

1. Security reviewers evaluating identity and document boundaries.

What eAkto is not

1. It is not an official government service or source of policy.
1. It does not directly change an agency record in the current prototype.
1. It does not treat an AI statement as verified evidence.
1. It does not expose provider secrets to Flutter.
1. It is not ready for production without the controls listed in Scope and limitations.

Installation

Install the backend and frontend separately. On Windows, Git Bash is the most convenient way to use the included `start.sh` script.

Prerequisites

Tool	Requirement	Check
Git	Current supported release	`git --version`
Python	3.11 or newer recommended	`python --version`
Flutter	A release compatible with Dart `^3.12.2`	`flutter doctor`
Chrome	Required for the default web demo	`flutter devices`
curl	Required by `start.sh` health checks	`curl --version`

For mobile development, also install Android Studio or Xcode as required by Flutter.

Clone the repository

```bash
git clone https://github.com/Sisig-Mayo/eActo.git
cd eActo
```

Use the team branch assigned for your work. This documentation describes the code line that combines the Flutter frontend with the sandbox backend.

Install the FastAPI backend

From Git Bash:

```bash
cd backend
python -m venv .venv
.venv/Scripts/python.exe -m pip install -r requirements.txt
cp .env.example .env
cd ..
```

From PowerShell:

```powershell
cd backend
python -m venv .venv
.\.venv\Scripts\Activate.ps1
python -m pip install -r requirements.txt
Copy-Item .env.example .env
cd ..
```

`backend/.env` is ignored by Git. Keep it that way.

Install Flutter dependencies

```bash
cd frontend
flutter pub get
cd ..
```

The Lexend font and eAkto image assets are bundled in the repository; no font download is required at runtime.

Install the documentation site

Use an isolated environment so MkDocs does not alter backend dependencies:

```bash
python -m venv .venv-docs
.venv-docs/Scripts/python.exe -m pip install -r docs/requirements.txt
```

Preview the docs on a port that does not conflict with FastAPI (`8000`) or Flutter Web (`8001`):

```bash
.venv-docs/Scripts/python.exe -m mkdocs serve --dev-addr 127.0.0.1:8002
```

Open `http://127.0.0.1:8002`.

Confirm the installation

```bash
backend/.venv/Scripts/python.exe -m pytest -q backend/tests/test_health.py
cd frontend
flutter analyze
cd ..
.venv-docs/Scripts/python.exe -m mkdocs build --strict
```

> **Start with local-only behavior**
>
> The app can start without real provider credentials, but provider-backed sign-in and AI calls require authorized sandbox configuration. Never invent credential values to make a screen appear configured.

Configuration

Backend configuration is loaded by `pydantic-settings` from environment variables and `backend/.env`. Flutter accepts non-secret compile-time values through `--dart-define`.

Safe configuration workflow

```bash
cp backend/.env.example backend/.env
```

Then edit only `backend/.env`. Do not add provider secrets to:

1. Flutter source files
1. `--dart-define`
1. Markdown documentation
1. screenshots, logs, issues, or pull requests
1. shell history shared with other users

Required for a local UI preview

The backend defaults to SQLite and can start with blank provider settings:

```text
APP_ENV=development
DATABASE_URL=sqlite+pysqlite:///./eakto.db
CORS_ORIGINS=http://localhost:8001,http://127.0.0.1:8001
```

When Flutter runs on another port, update `CORS_ORIGINS` to the exact origin. Wildcard CORS is rejected.

Provider groups

Group	Variables	Secret boundary
eGovAI	`EGOV_AI_BASE_URL`, `EGOV_AI_ACCESS_CODE`, generation and threshold settings	Access code is backend-only
eGovPH SSO	`EGOVPH_SSO_BASE_URL`, `EGOVPH_SSO_PARTNER_CODE`, `EGOVPH_SSO_PARTNER_SECRET`	Partner secret is backend-only
eVerify	`EVERIFY_BASE_URL`, client ID, client secret, public key, identity-reference secret	Client and reference secrets are backend-only; public SDK key may be returned safely
Face Liveness	Base URL, API key, callback URL, confidence threshold	API key is backend-only
Supabase	Project URL, service key, database URL, private bucket and table	Service key and database URL are backend-only

Use the Configuration reference for every supported variable and default.

Flutter API URL

The default URL depends on the target:

1. Web and desktop: `http://localhost:8000/api/v1`
1. Android emulator: `http://10.0.2.2:8000/api/v1`

Override it when needed:

```bash
flutter run -d chrome \
  --web-port 8001 \
  --dart-define=API_BASE_URL=http://localhost:8000/api/v1
```

`API_BASE_URL` is not a secret.

Restricted integration specifications

Some provider contracts used during the hackathon are available only to authorized participants. This repository's docs intentionally do not reproduce private credentials, dashboard values, full partner specifications, or sensitive sample payloads. Team members should retrieve those details from the authorized eGovPH channels and map them to the environment variables above.

> Never commit `.env` or PEM files
>
> `backend/.env` and `backend/everify-public.pem` are ignored. Before pushing, run `git status --short --untracked-files=all` and verify that no credential artifact appears.

Run locally

The recommended development setup uses FastAPI on port `8000`, Flutter Web on `8001`, and MkDocs on `8002`.

Run both app processes from Git Bash

```bash
./start.sh
```

The script:

1. validates both ports;
2. finds `backend/.venv` or `backend/.venv311`;
3. creates `backend/.env` from the example when missing;
4. reuses an already healthy backend on the configured port;
5. starts FastAPI with reload;
6. waits for `/api/v1/health`;
7. installs Flutter packages when needed; and
8. starts Flutter Web with the correct API URL.

Press `Ctrl+C` once to stop the processes started by the script.

Change ports

```bash
BACKEND_PORT=9000 FRONTEND_PORT=9001 ./start.sh
```

Use another Flutter device

```bash
FLUTTER_DEVICE=windows ./start.sh
```

Run the services separately

Backend:

```bash
cd backend
.venv/Scripts/python.exe -m uvicorn app.main:app \
  --host 127.0.0.1 \
  --port 8000 \
  --reload
```

Frontend:

```bash
cd frontend
flutter run -d chrome \
  --web-port 8001 \
  --dart-define=API_BASE_URL=http://localhost:8000/api/v1
```

Documentation:

```bash
.venv-docs/Scripts/python.exe -m mkdocs serve \
  --dev-addr 127.0.0.1:8002
```

Useful local URLs

URL	Purpose
http://localhost:8001	Flutter Web
http://127.0.0.1:8000/api/v1/health	Backend readiness
http://127.0.0.1:8000/docs	Generated OpenAPI UI
http://127.0.0.1:8000/openapi.json	Machine-readable API schema
http://127.0.0.1:8002	Project documentation

Port already in use

On Windows:

```powershell
Get-NetTCPConnection -LocalPort 8000,8001,8002 -ErrorAction SilentlyContinue |
  Select-Object LocalPort, State, OwningProcess
```

Stop only the process you recognize, or choose different ports. Do not terminate a process solely because it appears in this list.

SECTION 02

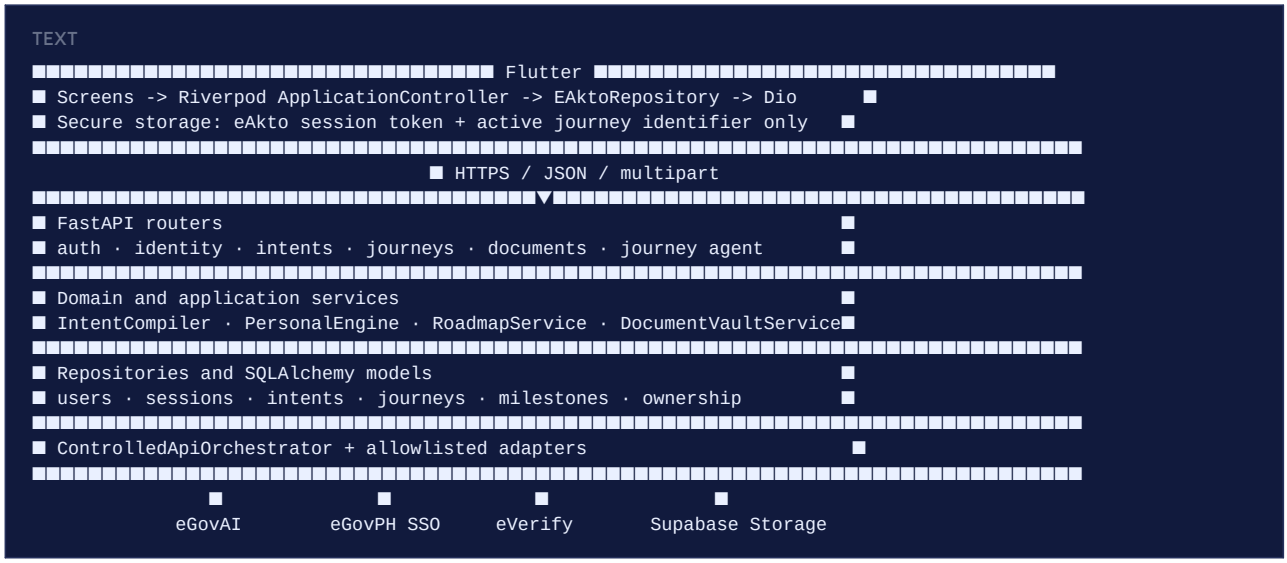
Architecture

This section is part of the consolidated eAkto technical and product documentation.

System overview

eAkto uses a Flutter client, a FastAPI application, relational journey state, and controlled adapters for external government and storage services. The current implementation is a modular monolith, not a distributed microservice system.

High-level architecture



Main responsibility boundaries

Layer	Owns	Does not own
Flutter presentation	Layout, interaction, local UI state, navigation	Provider secrets, workflow selection, official verification
Flutter repository	Typed HTTP call locations and multipart construction	Business rules
FastAPI routers	HTTP contract, dependencies, response models	Direct SQL details
Services/domain	Intent routing, assurance, journey resolution, evidence rules	UI rendering
Repositories	Queries and persistence	Provider HTTP calls
Integrations	Provider-specific transport and response validation	Citizen journey policy
Workflow seeds	Versioned declarative steps and content	Free-form executable code

Request path

For a typical journey:

```text
Citizen message
  -> POST /api/v1/intents/compile
  -> validated situation assessment
  -> compatible workflow code
  -> POST /api/v1/journeys/from-intent/{intent_id}
  -> rule/dependency resolution
  -> persisted journey and milestones
  -> GET /api/v1/journeys/{journey_id}/plan
  -> Flutter renders backend-provided questions and documents
```

Trust boundaries

1. Device boundary - Flutter stores only the internal session token and active journey ID in secure storage.
2. Application boundary - FastAPI hashes session tokens before persistence and applies resource ownership checks.
3. High-assurance boundary - journey creation with document evidence and every vault route require a current high-assurance result.
4. Provider boundary - credentials are loaded from backend settings and never returned to Flutter.
5. AI boundary - generated content is validated and cannot supply arbitrary provider URLs, methods, headers, or capabilities.

> Terminology
>
> Older source names still use `lifegraph` and `caseId`. The active backend domain calls the persisted object a personal journey and exposes it under `/journeys`.

Flutter frontend

The frontend is a Flutter Material 3 application organized by feature. It uses Riverpod for dependency injection and state, Dio for HTTP, GoRouter for routes, and `flutter_secure_storage` for the internal eAkto session reference.

Startup

```dart
void main() {
  WidgetsFlutterBinding.ensureInitialized();
  runApp(const ProviderScope(child: EAktoApp()));
}
```

`EAktoApp` configures the Lexend-based light theme and `MaterialApp.router`.

Layers

```text
presentation widgets
  v watch / invoke
Riverpod providers + ApplicationController
  v call
EAktoRepository
  v call
ApiClient (Dio)
  v
FastAPI /api/v1
```

Presentation

Feature screens live under `frontend/lib/features/<feature>/presentation/`. Shared eAkto cards, buttons, navigation, page shells, and error states live in `frontend/lib/core/widgets/lifegraph_widgets.dart`.

The Home shell uses:

1. an `IndexedStack` to preserve Home, Updates, and Profile state;
1. a `DraggableScrollableSheet` for the journey conversation;
1. a separate custom navigation container and Journey control; and
1. backend-backed “My journeys” cards.

State

`ApplicationController` is a `StateNotifier<AppState>`. It coordinates authentication, intent assessment, active journey state, milestone completion, evidence refresh, and sign-out.

```dart
final applicationControllerProvider =
    StateNotifierProvider<ApplicationController, AppState>((ref) {
  return ApplicationController(
    ref.watch(lifeGraphRepositoryProvider),
    ref.watch(sessionStorageProvider),
  );
});
```

`FutureProvider`s load independent collections such as journeys and government documents.

Repository and HTTP

Widgets never call Dio directly. `EAktoRepository` declares endpoint operations and `ApiClient`:

1. selects the configured base URL;
1. adds `Authorization: Bearer <session>` when available;
1. maps backend error envelopes into `AppError`;
1. uses a 10-second connection timeout and 75-second receive timeout; and
1. supports JSON and multipart uploads.

Navigation

GoRouter paths are declared centrally in `frontend/lib/app/router.dart`. Authenticated pages use `_AuthenticatedRoute`, which shows the government authentication choices when no profile is loaded.

Important routes:

Route	Purpose
`/auth`	Choose eVerify or eGovPH
`/egovph/sso`	Handle an SSO exchange-code callback
`/home`	Home, Updates, Profile, and journey chat shell
`/documents`	Government document vault
`/journeys`	Full journey collection
`/intent/processing`	AI processing and understanding flow
`/journey/plan`	Dynamic questions and documents
`/journey/document`	Upload one required document

Local secure storage

`SessionStorage` uses two keys:

1. `lifegraph_session_token`
1. `lifegraph_active_case_id`

No provider access token, partner secret, citizen demographic payload, QR value, or document bytes should be stored there.

Legacy demo paths

`EAktoRepository` still contains methods for older `/cases`, payment, notification, translation, and developer-scenario endpoints. These are used by legacy demo screens and tests but are not mounted by the current FastAPI application. New production-facing code should use the mounted `/auth`, `/intents`, `/journeys`, `/identity`, and `/documents` APIs.

FastAPI backend

The backend is a FastAPI modular monolith. `backend/app/main.py` constructs the database, provider clients, controlled orchestrators, workflow seeds, application services, middleware, and mounted routers.

Startup lifecycle

`create_app()` performs these steps:

1. Load validated settings.
2. Construct provider clients.
3. Build the SQLAlchemy engine and session factory.
4. Create compatible local tables and migrations.
5. Seed active workflow templates.
6. Construct the eGovAI capability registry and orchestrator during lifespan.
7. Mount API routers under `/api/v1`.
8. Register consistent error handlers.
9. Close provider clients and dispose the database engine during shutdown.

Backend packages

Package	Responsibility
`api/`	FastAPI routers and dependencies
`schemas/`	Pydantic request, response, rule, and provider models
`models/`	SQLAlchemy persistence models
`repositories/`	Queries, ownership, and transaction boundaries
`services/`	Authentication, intent compilation, identity, vault, and access services
`domain/`	Deterministic journey, roadmap, dependency, and rule engines
`integrations/`	External provider clients, adapters, registry, and orchestration
`agents/`	Bounded journey-assistance tools and run metadata
`seeds/`	Versioned workflow definitions

Dependency injection

FastAPI dependencies open and close a SQLAlchemy session per request. Authentication dependencies resolve the bearer token into an `AuthenticatedCitizen`.

```python
def get_db(request: Request):
    session = request.app.state.session_factory()
    try:
        yield session
    finally:
        session.close()
```

`require_high_assurance` builds on authentication and requires a current `DOCUMENT_VAULT_ACCESS` assurance record.

Controlled external capabilities

The `ControlledApiOrchestrator` accepts a capability code, not a caller-supplied URL. For each call it validates:

1. whether the capability exists and is enabled;
1. required permissions;
1. internal-only restrictions;
1. consent references;
1. current journey/action context;
1. idempotency keys;
1. Pydantic argument shape;
1. a per-capability in-memory rate limit; and
1. a timeout.

This prevents an AI response or API caller from turning the backend into an arbitrary HTTP proxy.

Persistence behavior

SQLite is the local default. SQLAlchemy 2 models and repositories also support a configured relational database URL. Workflow templates are versioned and seeded idempotently; existing personal journeys retain the workflow version used at creation.

Error handling

Expected failures raise `EaktoError` and return a safe error envelope. Unexpected exceptions are converted to a generic `INTERNAL_SERVER_ERROR`; exception details are not returned to the client.

See Error format.

Data model

The relational model separates identity, application sessions, intent assessment, journey state, and resource ownership.

Core relationships

```
TEXT
User
 ■■■ ApplicationSession
 ■    ■■■ IdentityAssurance
 ■■■ ResourceOwner ■■> CitizenIntent
 ■                 ■■> PersonalJourney
 ■■■ private Supabase document metadata (owner UUID)

CitizenIntent
 ■■■ PersonalJourney
      ■■■ WorkflowTemplate (versioned reference)
      ■■■ JourneyMilestone [many]
```

Main entities

`User`

Stores:

1. internal UUID;
1. pseudonymous or provider-derived citizen reference;
1. minimized provider profile JSON; and
1. creation/update timestamps.

The profile is minimized before storage and response.

`ApplicationSession`

Stores a SHA-256 hash of the eAkto session token, never the raw token. Sessions have a status, assurance level, expiry, and user reference.

`CitizenIntent`

Stores the original and normalized citizen message, validated parsed context, selected workflow, support decision, confidence, and timestamps.

`WorkflowTemplate`

Stores a validated JSON workflow definition under a unique `(workflow_code, version)` pair. Only active templates are selected for new journeys.

`PersonalJourney`

Stores the selected workflow/version, title, status, current milestone, progress percentage, and resolved context. The resolved context contains normalized intent context, redacted evidence state, high-assurance state, and submitted milestone answers.

`JourneyMilestone`

Stores a resolved snapshot of one candidate milestone:

1. applicability and position;
1. current status;
1. inclusion or skip reason;
1. dependencies;
1. source rule; and
1. action definition with questions and documents.

Because the resolved action is persisted, the client does not need to hardcode a form per agency.

`ResourceOwner`

Connects a user UUID to an owned `INTENT` or `JOURNEY` without copying identity fields into domain tables. API routes check this relation before returning or changing a resource.

`IdentityAssurance`

Records assurance purpose, status, level, verified field names, consent time, expiry, and temporary provider references. Services clear sensitive provider references after completion where appropriate.

Document storage

Document objects and metadata live in Supabase, not in the local SQLAlchemy journey tables. The backend returns a redacted evidence summary to the journey engine:

1. available document codes;
1. verified and active document codes; and
1. allowlisted record-state attributes.

Storage paths, object hashes, provider secrets, and signed URLs are not added to journey context.

Data retention note

The repository implements technical minimization, but it does not define a complete production retention schedule. A deployment must add an approved policy for session, intent, assurance, journey, audit, and document retention.

Journey engine

eAkto turns a validated intent into a versioned, deterministic journey. AI proposes structured context; it does not generate executable workflow code.

Workflow selection

`workflow_mapper.py` applies backend compatibility rules:

1. marriage-related record updates map to `POST_MARRIAGE_RECORD_UPDATE`;
1. specific non-marriage record corrections map to `GOVERNMENT_RECORD_INFORMATION_UPDATE`;
1. generic conversation, unsupported goals, portal failures, and completed-status claims do not automatically create a journey.

The selected workflow must be present as an active template.

Declarative rule evaluation

Rules use validated operators such as:

1. `equals` / `not_equals`
1. `contains` / `not_contains`
1. `in` / `not_in`
1. `exists` / `not_exists`
1. `is_empty` / `is_not_empty`
1. nested `all`, `any`, and `none`

The evaluator resolves dot-separated fields in a read-only context. It never executes code stored in the database.

```python
WorkflowRule(
    field="intent.mentioned_records",
    operator=RuleOperator.CONTAINS,
    value="NATIONAL_IDENTITY",
)
```

Dependency ordering

Applicable milestones are topologically ordered. A milestone is blocked when:

1. it names a dependency not present in the workflow definition;
1. its dependency is already blocked; or
1. a cycle prevents a valid order.

Within valid dependency order, priority and stable milestone codes provide deterministic ordering.

Journey creation

The Personal Engine:

1. verifies that the intent exists and is supported;
2. loads the active workflow template;
3. builds context from intent, redacted document evidence, and assurance;
4. resolves every milestone;
5. persists applicable and skipped milestone snapshots;
6. marks the first applicable milestone `CURRENT`; and
7. links the journey to its citizen owner.

Creating a journey from the same intent is idempotent: an existing journey is returned.

Progress and answers

Only the current applicable milestone can be completed. Submitted answers are stored under:

```JSON
{
  "milestone_answers": {
    "CONFIRM_REQUESTED_INFORMATION": {
      "target_record": "National ID",
      "requested_change": "Address"
    }
  }
}
```

Progress is calculated from persisted milestone state:

```TEXT
completed applicable milestones / all applicable milestones × 100
```

Flutter animates from the previous visual value to the new server percentage; the backend remains the source of truth.

Reevaluation

Evidence or context updates can re-run resolution. Previously completed milestones remain completed. Newly applicable steps are inserted into the resolved order, and the first incomplete step becomes current.

This allows a newly uploaded, verified document to remove a redundant preparation step without losing citizen progress.

SECTION 03

Core flows

This section is part of the consolidated eAkto technical and product documentation.

Authentication and identity

eAkto supports two citizen sign-in choices: eGovPH SSO and eVerify. Both end by issuing an internal eAkto application session. Provider secrets and provider access tokens remain in FastAPI.

eGovPH SSO flow

```text
Flutter requests /auth/egovph/start
  -> backend reports whether partner credentials are configured
  -> citizen obtains/scans an eGovPH one-time exchange code
  -> Flutter sends only that code to /auth/egovph/callback
  -> backend exchanges it using partner credentials
  -> backend obtains the authorized citizen profile
  -> eAkto creates or reuses a user and returns an eAkto session
```

The provider contract used by this repository does not require an `EGOVPH_SSO_AUTHORIZE_URL`. The configured values are the base URL, partner code, and partner secret.

The exchange code is short-lived and single-use. Never log it or store it in Flutter secure storage.

eVerify sign-in flow

eVerify sign-in uses explicit biometric consent and Face Liveness.

```text
POST /auth/everify/start
  -> one-time eAkto challenge + public Web SDK key
  -> official Face Liveness Web SDK
  -> one-time face_liveness_session_id
  -> demographics OR signed National ID QR
  -> POST /auth/everify/complete or /auth/everify/qr/complete
  -> provider verification
  -> HIGH-assurance eAkto session
```

The standard demographic flow submits the name and birth date entered by the citizen. The QR flow relays a signed QR value for verification. The raw QR value and biometric media are not persisted in the eAkto domain database.

> Two different QR concepts
>
> An eGovPH SSO exchange code signs the citizen into a partner application. A National ID QR is an eVerify identity input. Sending one type to the other flow results in an invalid or unauthorized response.

eAkto application session

On successful sign-in, the backend returns a high-entropy session token once. The database stores only its SHA-256 hash.

Clients authenticate with:

```
HTTP
Authorization: Bearer EAKTO_SESSION_TOKEN
```

`X-Session-Token` is accepted for compatibility, but Bearer authentication is preferred.

Sessions default to an eight-hour lifetime. Sign-out marks the server session inactive and clears Flutter secure storage. Flutter clears local state even when the server session has already expired.

High assurance

Some operations require more than an ordinary authenticated session:

1. creating a journey with private document evidence;
1. refreshing journey evidence;
1. listing, uploading, or opening vault documents.

`require_high_assurance` requires a current assurance record for `DOCUMENT_VAULT_ACCESS`. The default assurance lifetime is 15 minutes and is configurable.

Profile data

The profile screen is read-only for SSO-derived information. The backend returns a minimized profile containing fields required by eAkto's citizen experience, such as display name and optional contact/photo information. Profile editing is not exposed because authoritative updates belong to the identity provider.

Production controls still required

1. Rate-limit public challenge creation and completion routes.
1. Bind callbacks to approved HTTPS origins.
1. Add replay monitoring for challenge and exchange-code failures.
1. Define biometric consent records and retention with legal review.
1. Review the exact authorized provider contract before deployment.

AI intent assessment

The intent compiler converts an English, Filipino, or Taglish message into a validated situation assessment. It distinguishes a journey-worthy government request from conversation or ambiguity.

Processing modes

Authorized eGovAI configured

When `EGOV_AI_BASE_URL`, `EGOV_AI_ACCESS_CODE`, and generation are configured, the compiler can use the authorized eGovAI adapter for requests that are uncertain, complex, or explicitly require AI assessment. Clear, high-confidence rules may resolve without a provider call.

Provider unavailable locally

Named deterministic heuristics handle supported high-confidence requests and provide a safe fallback when provider credits are unavailable. The heuristic path produces the same Pydantic assessment shape. If neither path can establish a supported request confidently, eAkto asks for clarification instead of inventing a journey.

Assessment shape

The current response separates:

1. `explicit_context` - facts stated by the citizen;
1. `inferred_context` - possible affected services and blockers;
1. `uncertain_claims` - statements expressed with uncertainty;
1. `claims_requiring_verification` - claims that need an authoritative source;
1. `workflow_candidates` - bounded workflow suggestions;
1. `assistant_reply` - normal conversation when no journey should be proposed; and
1. backend-selected workflow and routing diagnostics.

Journey proposal decision

A review card appears only when the compiler determines that:

1. the message concerns a government goal or record;
2. enough specific context exists;
3. a compatible workflow is registered; and
4. the backend confidence and compatibility checks pass.

Messages such as “hello,” “how are you,” or “what should I do?” receive a concise normal reply. The conversation composer remains available so the citizen can clarify.

Direct requests such as changing an address or correcting National ID information do not require an artificial marriage or life-event classification.

Authority rules

The prompt and backend enforce these principles:

1. Uncertain wording remains uncertain.
1. AI is not an acceptable verification source.
1. A provider response cannot mark an agency record complete without authoritative evidence.
1. AI cannot create new workflow codes.
1. The backend, not the model, selects the final workflow.
1. Provider output is parsed once, with at most one bounded schema-repair generation.

Controlled capabilities

eGovAI is accessed through capability codes:

Capability	Purpose	Key guard
`EGOVAI_CREDITS_READ`	Confirm available sandbox credits	Brief in-memory cache
`EGOVAI_ASSISTANT_GENERATE`	Assess citizen situation	Permission-gated
`EGOVAI_TRANSLATE`	Translate current guidance	Current journey action required
`EGOVAI_SPEECH_GENERATE`	Produce written speech-style guidance	Optional and non-blocking
`EGOVAI_DOCUMENT_EXTRACT`	Normalize allowed document fields	Consent + idempotency + confirmation

Callers cannot supply a raw provider URL, HTTP method, access code, bearer token, or arbitrary header.

Testing AI behavior

Ordinary tests use fakes and never spend provider credits:

```bash
cd backend
.venv/Scripts/python.exe -m pytest -q \
  tests/test_intents.py \
  tests/test_marriage_precision.py \
  tests/test_situation_assessment.py
```

The opt-in sandbox tests are described in Testing.

Journey lifecycle

A journey is a persisted instance of a versioned workflow. It belongs to one citizen and tracks applicable milestones, answers, progress, and the current valid action.

Create a journey

Prerequisites:

1. authenticated citizen;
1. current high assurance;
1. an owned, supported intent;
1. active workflow template.

```bash
curl -X POST \
  -H "Authorization: Bearer $EAKTO_SESSION" \
  http://localhost:8000/api/v1/journeys/from-intent/INTENT_ID
```

Creation reads only redacted vault evidence. Calling the endpoint twice for the same intent returns the existing journey.

List and resume journeys

`GET /journeys` returns every owned journey ordered by recent activity. Home displays:

1. one active journey overview when status is `ACTIVE` or `BLOCKED`;
1. all saved journeys in “My journeys,” including completed items; and
1. the no-active-journey illustration when no active journey exists.

Flutter persists the active journey ID as a convenience. On restore, it falls back to the server collection if the local reference is missing.

Render a dynamic plan

`GET /journeys/{id}/plan` returns applicable milestones with backend-defined:

1. title and description;
1. position and status;
1. questions, input type, and options;
1. required documents;
1. vault document code;
1. why each document matters; and
1. how to add it.

Flutter renders this structure rather than selecting a hardcoded marriage form.

Complete a milestone

```bash
curl -X POST \
  -H "Authorization: Bearer $EAKTO_SESSION" \
  -H "Content-Type: application/json" \
  -d '{"answers":{"requested_change":"Address"}}' \
  http://localhost:8000/api/v1/journeys/JOURNEY_ID/milestones/MILESTONE_CODE/complete
```

The backend rejects attempts to:

1. complete a step that is not applicable;
1. skip ahead;
1. complete a milestone that is not current; or
1. mutate another citizen's journey.

Completing an already completed current milestone is idempotent.

Refresh evidence

After a document upload, Flutter calls:

```http
POST /api/v1/journeys/{journey_id}/evidence-refresh
```

The Personal Engine reevaluates the workflow with new redacted evidence and preserves completed milestones.

Complete a journey

When no applicable incomplete milestone remains:

1. journey status becomes `COMPLETED`;
1. `current_milestone_code` becomes `null`;
1. progress becomes `100`;
1. completion time is stored; and
1. Flutter clears the active journey reference.

The completed journey remains visible in journey history.

Document vault

The Vault gives a high-assurance citizen quick access to supported government-document copies. Objects are private and owner-scoped.

Supported prototype document types

Code	Label	Agency code
`NATIONAL_ID`	National ID	`PHILSYS`
`BIRTH_CERTIFICATE`	Birth Certificate	`PSA`
`MARRIAGE_CERTIFICATE`	Marriage Certificate	`PSA`
`DRIVER_LICENSE`	Driver's License	`LTO`
`PHILHEALTH_ID`	PhilHealth ID	`PHILHEALTH`
`PASSPORT`	Passport	`DFA`

Allowed content types are PDF, JPEG, and PNG. The configured default maximum is 8 MiB.

Upload path

```text
Flutter file picker
  -> local size and format check
  -> multipart POST /api/v1/documents
  -> backend validates document code
  -> reads at most configured limit + 1 byte
  -> validates file signature and MIME type
  -> uploads to randomized owner-scoped private object path
  -> stores safe metadata
  -> refreshes journey evidence
```

The citizen-facing required-document row opens a detail page containing backend-provided "Why it matters" and "How to add it" guidance.

List and open documents

`GET /documents` returns owner-safe summaries. The client requests a short-lived signed URL through:

```http
POST /api/v1/documents/{document_id}/signed-url
```

Storage paths, service keys, and object hashes are never returned as ordinary list metadata.

Journey evidence

A vault file does not automatically prove an official fact. Evidence distinguishes:

1. available document;
1. active document;

1 verification status; and

1 allowlisted record attributes relevant to a requested change.

AI-extracted fields are marked `AI_EXTRACTED_UNVERIFIED` and require citizen confirmation before use.

Operator seeding

Authorized demo operators can seed a synthetic document after identifying the backend user UUID:

```powershell
cd backend
.\.venv\Scripts\python.exe -m scripts.seed_government_document `
  --owner-user-id USER_UUID `
  --file C:\synthetic\driver-license.pdf `
  --document-code DRIVER_LICENSE `
  --agency-code LTO
```

Never use a real government document in the hackathon demo.

Supabase setup

Apply the idempotent schema migration:

```bash
cd backend
.venv/Scripts/python.exe -m scripts.apply_supabase_migrations
```

The preferred database variable is `SUPABASE_DATABASE_URL`. The historical misspelling `SUPABSE_URI` remains accepted for compatibility.

SECTION 04

API reference

This section is part of the consolidated eAkto technical and product documentation.

API conventions

FastAPI mounts the current API under `/api/v1`. The local base URL is:

```text
TEXT
http://localhost:8000/api/v1
```

Interactive OpenAPI documentation is available at `/docs`.

Authentication

Use the internal eAkto session:

```http
HTTP
Authorization: Bearer EAKTO_SESSION_TOKEN
```

Do not pass eGovPH, eVerify, eGovAI, or Supabase credentials to eAkto endpoints.

Content types

Content	Type
Ordinary requests	`application/json`
Vault and extraction uploads	`multipart/form-data`
Ordinary responses	`application/json`

Access levels

Level	Meaning
Public	No existing eAkto session is required
Authenticated	Valid, unexpired eAkto session
High assurance	Authenticated plus current `DOCUMENT_VAULT_ACCESS` assurance
Development only	Endpoint behavior is restricted by `APP_ENV`
Prototype internal	Not safe for public exposure until authentication/ownership guards are added

Mounted endpoint groups

1. Auth and identity
1. Intents and journeys
1. Documents and agent
1. Error format

Cache behavior

Journey collections, document catalog responses, and vault lists set:

```
HTTP
Cache-Control: private, no-store
```

Clients should not persist high-assurance document responses in shared caches.

Identifier format

Internal users, sessions, challenges, intents, journeys, milestones, assurances, and documents use opaque identifiers. Treat them as strings; do not derive meaning or authorization from their format.

Source of truth

This reference includes only routes mounted in `backend/app/main.py`. Older Flutter repository methods referencing `/cases`, `/notifications`, `/dev/scenarios`, and similar demo endpoints are not part of the current mounted API.

Auth and identity API

Authentication endpoints

Method	Path	Access	Purpose
GET	/auth/egovph/start	Public	Return partner-code metadata and configuration status
POST	/auth/egovph/callback	Public	Exchange a one-time SSO code and create an eAkto session
POST	/auth/everify/start	Public	Create a consented, expiring sign-in challenge
GET	/auth/everify/{challenge_id}	Public	Read challenge status and public SDK configuration
POST	/auth/everify/complete	Public	Complete demographics plus Face Liveness sign-in
POST	/auth/everify/qr/complete	Public	Complete signed QR plus Face Liveness sign-in
GET	/auth/session	Authenticated	Return the current minimized profile and session
POST	/auth/sign-out	Authenticated	Invalidate the current eAkto session

Start eGovPH

```bash
curl http://localhost:8000/api/v1/auth/egovph/start
```

Safe response shape:

```json
{
  "partner_code": "CONFIGURED_PARTNER_CODE",
  "configured": true
}
```

The partner secret is never included.

Complete eGovPH

```bash
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"exchange_code":"ONE_TIME_EXCHANGE_CODE"}' \
  http://localhost:8000/api/v1/auth/egovph/callback
```

Never place a real exchange code in committed examples or logs.

Start eVerify

```bash
curl -X POST \
  -H "Content-Type: application/json" \
  -d '{"consent":true}' \
  http://localhost:8000/api/v1/auth/everify/start
```

The response contains a challenge ID, expiry/status information, and the public Web SDK key when configured. It does not expose the eVerify client secret.

Identity assurance endpoints

Method	Path	Access	Purpose
`POST`	`/identity/liveness/session`	Authenticated	Start a purpose-bound liveness session
`GET`	`/identity/liveness/{assurance_id}`	Authenticated	Refresh liveness state
`POST`	`/identity/everify`	Authenticated	Verify an assurance after successful liveness
`GET`	`/identity/assurance`	Authenticated	Return current assurance status

Example liveness request:

```json
{
  "consent": true,
  "purpose": "DOCUMENT_VAULT_ACCESS"
}
```

Session response

An authentication completion returns:

1. a raw eAkto `session_token` once;
1. expiry and assurance information; and
1. a minimized citizen `profile`.

Applications must store the token in platform-secure storage and must not print it.

Intents and journeys API

Intent endpoint

Method	Path	Access	Purpose
POST	/intents/compile	Authenticated	Assess and persist one citizen message

Request:

```JSON
{
  "message": "I need to correct the address on my National ID."
}
```

Important response fields:

```JSON
{
  "intent_id": "INTENT_ID",
  "original_message": "I need to correct the address on my National ID.",
  "supported": true,
  "selected_workflow_code": "GOVERNMENT_RECORD_INFORMATION_UPDATE",
  "response_mode": "JOURNEY_PROPOSAL",
  "assistant_reply": null,
  "explicit_context": {
    "goal": "CORRECT_PERSONAL_INFORMATION",
    "requested_changes": ["ADDRESS"]
  }
}
```

The complete response also contains inference, uncertainty, verification, candidates, and routing diagnostics.

Journey endpoints

Method	Path	Access	Purpose
GET	/journeys	Authenticated	List every owned journey
POST	/journeys/from-intent/{intent_id}	High assurance	Create or return the journey for an intent
GET	/journeys/{journey_id}/roadmap	Authenticated	Return resolved roadmap state
GET	/journeys/{journey_id}/plan	Authenticated	Return dynamic milestones, questions, and documents
POST	/journeys/{journey_id}/milestones/{milestone_code}/complete	Authenticated	Persist answers and advance
GET	/journeys/{journey_id}/current-action	Authenticated	Return one current valid action
POST	/journeys/{journey_id}/reevaluate	Authenticated	Update allowed intent context and resolve again
POST	/journeys/{journey_id}/evidence-refresh	High assurance	Resolve again with current vault evidence
GET	/journeys/{journey_id}/debug-resolution	Authenticated	Return rule-resolution diagnostics

Plan response

```JSON
{
  "journey_id": "JOURNEY_ID",
  "title": "Update Government Record Information",
  "status": "ACTIVE",
  "progress_percent": 25,
  "current_milestone_code": "CONFIRM_REQUESTED_INFORMATION",
  "milestones": [
    {
      "code": "CONFIRM_REQUESTED_INFORMATION",
      "title": "Tell us what needs to change",
      "position": 2,
      "status": "CURRENT",
      "questions": [],
      "documents": []
    }
  ]
}
```

Complete milestone

Request:

```JSON
{
  "answers": {
    "record": "National ID",
    "requested_change": "Address"
  }
}
```

The server rejects out-of-order actions with `409 INVALID_JOURNEY_ACTION`.

Ownership

Compilation claims the intent for the current citizen. Journey creation verifies intent ownership and claims the journey. Every subsequent journey endpoint checks that link.

Documents and agent API

Government document endpoints

All vault endpoints require high assurance.

Method	Path	Purpose
GET	/documents/catalog	Supported document types, MIME types, and upload limit
POST	/documents	Upload one owner-scoped private copy
GET	/documents	List safe summaries for the current citizen
POST	/documents/{document_id}/signed-url	Create short-lived owner-scoped access

Upload

```bash
curl -X POST \
  -H "Authorization: Bearer $EAKTO_SESSION" \
  -F "document_code=MARRIAGE_CERTIFICATE" \
  -F "file=@synthetic-marriage-certificate.pdf;type=application/pdf" \
  http://localhost:8000/api/v1/documents
```

Use only synthetic files in development.

Journey agent endpoints

Method	Path	Purpose
POST	/journeys/{journey_id}/agent/continue	Run one bounded current-action assistance operation
GET	/journeys/{journey_id}/agent/status	Read the latest safe agent state
GET	/journeys/{journey_id}/agent/debug	Development/test-only safe diagnostics
POST	/journeys/{journey_id}/guidance/translate	Translate current guidance
POST	/journeys/{journey_id}/guidance/speech	Generate written speech-style guidance
POST	/journeys/{journey_id}/documents/consent	Record journey-scoped extraction consent
POST	/journeys/{journey_id}/documents/extract	Extract and normalize an allowed file

Translation request

```json
{
  "target_lang": "fil"
}
```

The backend builds the source guidance from the current action. The caller cannot provide a provider prompt.

Document extraction

The extraction request is multipart:

1. `file` - PDF, JPEG, or PNG;
1. `consent_id` - active consent belonging to the journey.

Normalized evidence remains unverified until citizen confirmation.

> Current authorization gap
>
> The journey-agent router does not currently attach the same authentication and resource-ownership dependencies used by the main journey router. Treat these routes as local prototype endpoints only. Do not expose them publicly until authentication, ownership, high-assurance checks where appropriate, and abuse controls are added and tested.

Error format

Expected application failures return a stable JSON envelope:

```JSON
{
  "error": {
    "code": "IDENTITY_ASSURANCE_REQUIRED",
    "title": "Identity verification required",
    "message": "A current Face Liveness and eVerify result is required to access
      government documents.",
    "suggested_action": "Complete identity verification and try again.",
    "retryable": false
  }
}
```

Fields

Field	Meaning
`code`	Stable machine-readable identifier
`title`	Short citizen-friendly heading
`message`	Safe explanation without stack details
`suggested_action`	What the citizen or developer should do next
`retryable`	Whether repeating the same action may succeed

Common status classes

Status	Example
`400`	Capability or request not registered
`401`	Missing, invalid, expired, or signed-out session
`403`	High assurance, consent, or capability permission required
`404`	Owned resource or development-only endpoint unavailable
`409`	Invalid journey order, unavailable workflow, or missing idempotency
`413`	Upload exceeds configured size
`422`	Invalid request or unsupported document type
`429`	Capability rate limit reached
`500`	Unexpected server failure, redacted
`503`	Provider or configuration unavailable
`504`	Provider timeout

Flutter mapping

`ApiClient` converts a backend envelope into `AppError`. Network failures without an envelope use a local, retryable “backend unavailable” message.

UI code should display `title`, `message`, and `suggestedAction`; it should not branch on human-readable strings. Use `code` when program behavior differs.

SECTION 05

Development

This section is part of the consolidated eAkto technical and product documentation.

Project structure

eAkto is a single repository with a Flutter client, a FastAPI server, tests, and this documentation site. The backend is a modular monolith: features are separated by Python packages while sharing one process and database.

```text
eActo/
■■■ frontend/                # Flutter application
■   ■■■ lib/
■   ■   ■■■ core/             # Routing, theme, networking, shared services
■   ■   ■■■ features/         # Feature-first UI and state
■   ■■■ assets/               # Fonts, icons, and illustrations
■   ■■■ test/                 # Widget and unit tests
■■■ backend/
■   ■■■ app/
■   ■   ■■■ api/              # Routers mounted under /api/v1
■   ■   ■■■ core/             # Configuration, security, shared policy
■   ■   ■■■ models/           # SQLAlchemy persistence models
■   ■   ■■■ schemas/          # Pydantic API contracts
■   ■   ■■■ services/         # Application and provider integrations
■   ■   ■■■ workflows/        # Journey templates and execution logic
■   ■■■ tests/                # API, service, and workflow tests
■■■ docs/                     # Markdown documentation
■■■ docs_theme/               # Custom eAkto MkDocs theme
■■■ mkdocs.yml                # Documentation navigation and build config
■■■ .env.example              # Safe configuration names and examples
```

Frontend boundaries

Flutter code is organized around user-facing features. A feature should own its screens, providers, models, and widgets when those elements are not broadly reusable. Cross-feature concerns belong in `lib/core`.

The frontend communicates with the backend through repositories and API clients. Screens should not assemble HTTP requests or persist access tokens directly.

Backend boundaries

The backend follows a request-to-service flow:

```text
API router -> schema validation -> service/workflow -> repository/model -> response
  schema
```

1. Routers translate HTTP into application calls.
1. Schemas define validated input and output contracts.
1. Services coordinate business rules and external systems.
1. Workflow definitions describe questions, documents, milestones, and actions.
1. Models persist users, identity assurance, intents, journeys, and documents.

Keep provider-specific payloads inside their adapter. The rest of eAkto should depend on an eAkto-owned interface.

Where new code belongs

Change	Preferred location
New Flutter screen	`frontend/lib/features/<feature>/`
Reusable visual primitive	Existing shared widget or theme package
New API endpoint	Matching router in `backend/app/api/`
Business rule	Feature service, not the router
External provider request	Dedicated service adapter
New citizen journey	Workflow registry and versioned definition
Documentation	Matching section in `docs/`

Avoid adding code to legacy modules simply because a similar class exists there. Confirm that its router or feature is part of the currently mounted application first.

Testing

Run the smallest relevant test while developing, then run the complete local checks before opening a pull request. Tests must use fabricated identities and documents.

Backend tests

From the repository root:

PowerShell

```powershell
backend\.venv\Scripts\python.exe -m pytest backend\tests -q
```

Bash

```bash
backend/.venv/bin/python -m pytest backend/tests -q
```

If the backend environment does not exist, create and install it first as described in Installation.

Backend tests cover:

1. authentication and assurance boundaries;
1. API request and response contracts;
1. intent compilation and route selection;
1. journey creation, progress, and milestone transitions;
1. document metadata, consent, and access control;
1. provider adapter failures and timeouts.

Sandbox integration tests

Tests that call a provider sandbox are opt-in because they require authorized credentials, network access, and may consume credits:

```bash
RUN_EGOV_AI_SANDBOX_TESTS=true pytest -m sandbox
```

Never enable sandbox tests in an untrusted fork or paste their output into public issue trackers without reviewing it.

Flutter checks

From `frontend/`:

```bash
flutter pub get
flutter analyze
flutter test
```

Use widget tests for stateful navigation, chat transitions, form validation, and dynamic workflow rendering. Use golden tests sparingly for stable, high-value visual states and review intentional pixel changes.

Documentation checks

From the repository root:

PowerShell

```powershell
.venv-docs\Scripts\python.exe -m mkdocs build --strict
```

Bash

```bash
.venv-docs/bin/python -m mkdocs build --strict
```

Strict mode fails on navigation and reference warnings. Also review the generated site at phone, tablet, and desktop widths after changing the theme.

Test data rules

Use obviously fictional names, addresses, document numbers, images, and provider responses. Do not:

1. commit access tokens, exchange codes, session IDs, face images, or QR payloads;
1. use a real citizen account in an automated test;
1. record provider traffic containing personal data;
1. snapshot authorization headers or signed storage URLs.

Add a workflow

A workflow is a versioned, structured definition of a government journey. It should be driven by an approved service catalog entry - not generated directly from an unconstrained model response.

1. Define the catalog identity

Give the workflow a stable machine code, citizen-facing title, supported agencies, and eligibility boundary. Codes are persisted, so rename the display title instead of changing an existing code.

```python
WORKFLOW_CODE = "EXAMPLE_RECORD_UPDATE"
WORKFLOW_VERSION = 1
```

2. Create the workflow definition

Add a definition alongside the existing workflow seeds. Describe:

1. intake questions and validation;
1. required and conditional documents;
1. milestones and their ordering;
1. actions within each milestone;
1. completion rules and next-action text;
1. the agencies that may receive an eventual request.

Use stable IDs for every question, document requirement, milestone, and action. The Flutter client renders these structures dynamically.

3. Register and seed it

Register the definition in the workflow catalog and include it in the backend startup seed/test fixture. Seeding must be idempotent. If an existing definition changes meaning, create a new version and preserve active journeys on their original version.

4. Teach the intent layer

Add compact routing signals that help the intent compiler distinguish the workflow from nearby requests. The AI may normalize free text, but deterministic policy decides whether the confidence is high enough to suggest or create a journey.

Do not require a life event when the citizen has made a direct service request. “Update an address on my National ID” can be actionable even without a marriage, move, or loss event.

5. Verify the citizen experience

Test at least:

1. a clear request that should select the workflow;

2 an ambiguous request that should remain conversational;

3 a related but unsupported request;

4 missing and invalid answers;

5 every conditional document branch;

6 save, reload, and resume;

7 progress calculation and final completion.

6. Document the workflow

Update the supported scope and relevant API or operations pages. Record the source and approval status of each requirement internally. Restricted agency material must not be copied into public documentation.

> Requirements are policy, not decoration
>
> A seeded workflow is a prototype implementation until its requirements, forms, and agency routing have been validated by the responsible authority.

Contributing

Contributions should keep citizen flows clear, state recoverable, and integrations secure. Work from a focused branch and avoid mixing generated files, credentials, and unrelated refactors into the same change.

Development workflow

1. Sync the intended base branch.
2. Create a short-lived branch.
3. Make the smallest coherent change.
4. Add or update tests and documentation.
5. Run backend, Flutter, and documentation checks that apply.
6. Review the diff for secrets and generated artifacts.
7. Open a pull request that explains behavior, risk, and verification.

Suggested commit style:

```text
feat(journeys): persist milestone completion
fix(auth): reject expired identity assurance
docs(api): document journey error responses
```

Pull request checklist

1. [ ] Citizen-visible behavior is explained.
1. [ ] API contracts remain backward compatible or the migration is documented.
1. [ ] Authorization and ownership checks are tested.
1. [ ] Loading, empty, error, and retry states are handled.
1. [ ] No secret, token, personal data, private key, or restricted provider text is included.
1. [ ] `flutter analyze`, `flutter test`, and relevant backend tests pass.
1. [ ] `mkdocs build --strict` passes for documentation changes.
1. [ ] Screens remain usable with text scaling and narrow widths.

Coding principles

1. Prefer typed contracts over loose maps.
1. Keep HTTP and provider concerns out of UI widgets.
1. Keep business rules out of FastAPI route functions.
1. Make transitions idempotent where requests may be retried.
1. Preserve existing user state during schema or workflow changes.
1. Log stable internal identifiers, not sensitive payloads.

Documentation style

Write for an engineer who is new to eAkto. Lead with the purpose, give a safe runnable example, state preconditions, and explain failure modes. Use `YOUR_VALUE` or environment variable names - never a real credential.

SECTION 06

Operations

This section is part of the consolidated eAkto technical and product documentation.

Security and privacy

eAkto handles identity, contact, government-document, and journey data. Treat all citizen data as sensitive and minimize what is collected, retained, logged, and shared.

Trust boundaries

```
TEXT
Citizen browser/app
       ■ HTTPS
       ▼
eAkto API ■■■■■ authorized identity and AI providers
       ■
       ■■■■■ application database
       ■■■■■ private document storage
```

The Flutter application is an untrusted public client. It may contain public identifiers, but never partner secrets, server API credentials, database keys, or signing material.

Secret handling

1. Load backend secrets from a secrets manager or process environment.
1. Commit only `.env.example` with blank or unmistakably fake values.
1. Never place secrets in `--dart-define`, Flutter assets, web JavaScript, QR codes, logs, screenshots, or documentation.
1. Rotate a credential immediately if it enters Git history; deleting the file in a later commit is not sufficient.
1. Keep private keys outside the repository and restrict filesystem permissions.

See Configuration reference for variable names and sensitivity.

Authentication and authorization

An authenticated route must validate the eAkto session and then enforce resource ownership. High-assurance actions additionally require a current identity assurance record.

Apply these checks to:

1. journey creation and mutation;
1. document catalog and vault operations;
1. consent and extraction;
1. signed URL generation;
1. agent status, debug, and continuation endpoints;
1. any response containing citizen-specific state.

> Known blocker
>
> The current agent router does not consistently declare the authentication and ownership dependencies used by the other citizen APIs. Treat those endpoints as non-production until the guard is added and covered by tests.

Document protection

1. Use a private storage bucket.
1. Store object paths and metadata in the database; return short-lived signed URLs only after authorization.
1. Validate file size, declared type, detected type, and extension.
1. Scan uploads before downstream processing.
1. Do not expose provider extraction output to another user or journey.
1. Record explicit consent and purpose before extraction or sharing.

Logging and observability

Use structured logs with a request/correlation ID. Redact:

1. authorization and cookie headers;
1. provider tokens and exchange codes;
1. QR contents and liveness session data;
1. full names, addresses, birth dates, phone numbers, and emails;
1. document contents and signed URLs.

Record security events such as denied ownership checks, repeated authentication failures, and assurance expiry without storing the rejected secret.

Privacy by design

Collect only fields needed for the selected journey. Define retention and deletion rules before production, lock profile attributes that are authoritative from eGovPH, and provide a traceable consent history. Deployment owners remain responsible for validating compliance with applicable Philippine privacy rules and provider agreements.

Provider specifications supplied through authorized eGovPH channels are restricted integration material. This site intentionally documents eAkto-owned behavior rather than reproducing those specifications.

Deployment

The repository provides a hackathon prototype, not a turnkey production environment. A production deployment needs managed secrets, HTTPS, durable storage, database migrations, monitoring, rate limits, and security review.

Build the Flutter web client

```bash
cd frontend
flutter pub get
flutter build web --release \
  --dart-define=API_BASE_URL=https://api.example.gov.ph/api/v1
```

Publish `frontend/build/web` through a static host or CDN. Configure SPA route fallback to `index.html`, HTTPS, cache rules for hashed assets, and restrictive security headers. `API_BASE_URL` is public configuration; no secret belongs in the Flutter build.

Run the FastAPI service

Install locked dependencies, configure the environment, and run an ASGI server behind a TLS-terminating reverse proxy. For example:

```bash
uvicorn app.main:app --host 0.0.0.0 --port 8000
```

Use multiple workers only after confirming that background state, startup seeding, and provider clients are safe across processes.

Required infrastructure

1. HTTPS endpoint and DNS for the API;
1. exact CORS origins for the deployed web client;
1. managed relational database with backups;
1. private Supabase Storage bucket or equivalent object storage;
1. secret manager for provider and database credentials;
1. centralized redacted logs, metrics, alerting, and audit events;
1. provider callback URLs registered for the deployed environment.

Release order

1. Apply compatible database migrations.
2. Deploy the backend and verify `/api/v1/health`.
3. Run smoke tests against non-sensitive test data.
4. Deploy the Flutter client with the final API URL.

5 Verify authentication, identity assurance, journey save/resume, and document upload.

6 Monitor error rate and provider failures during rollout.

Prefer backward-compatible API changes so the previous frontend can continue working during deployment.

Documentation deployment

```bash
python -m mkdocs build --strict
```

Publish the generated `site/` directory as static content. Do not copy `.env` files, source maps containing secrets, test fixtures with personal data, or restricted provider documentation into the published artifact.

Production readiness gate

Before public use, resolve the agent-route authorization gap, define data retention, add rate limiting and upload malware scanning, validate workflow requirements with agencies, complete accessibility and privacy reviews, and exercise backup restoration.

Troubleshooting

Start with the visible symptom, then verify configuration and the boundary where the request fails. Do not print entire environment files or authorization headers while diagnosing.

Flutter web port is already in use

If Flutter reports `Only one usage of each socket address ... is normally permitted`, another process owns the requested port.

PowerShell

```powershell
Get-NetTCPConnection -LocalPort 8000 |
  Select-Object LocalAddress, LocalPort, State, OwningProcess
```

Bash

```bash
netstat -ano | grep ':8000'
```

Stop the known development process or choose another port:

```bash
flutter run -d chrome --web-port 8001
```

Browser cannot reach the API

Confirm:

1. the backend is listening on the expected port;
2. Flutter was started with an API URL that includes `/api/v1`;
3. the browser origin is in `CORS_ORIGINS`;
4. HTTPS pages are not calling an HTTP API;
5. a reverse proxy is forwarding the full path.

Test the public health endpoint before debugging authentication.

eGovPH or eVerify returns 401

A `401` usually means the token presented to that endpoint is absent, expired, for the wrong environment, or the wrong kind of token. Partner code and partner secret are credentials used server-side to obtain/exchange a token; they are not substitutes for a Bearer access token.

1. Confirm sandbox URLs and sandbox credentials are paired.
1. Start a fresh flow; exchange codes are typically short-lived and single-use.
1. Keep the device clock accurate.

1. Do not scan an eVerify identity QR into an eGovPH SSO exchange-code flow; they are separate contracts.
1. Check redacted server logs for the provider status and correlation ID.

   Do not paste the token into an issue, chat, or documentation page.

QR scanner is black

On Flutter web, camera access requires a secure context (`https://` or local development), browser permission, a selected video input, and an available camera.

1. Confirm the browser did not block camera permission.
1. Close other applications using the camera.
1. Test on a real phone if the desktop has no camera.
1. Verify the SDK/Flutter camera element has non-zero dimensions.
1. Show a permission-denied or no-camera message instead of an indefinite black view.

AI response is unrelated

Check the normalized request and catalog candidate scores before blaming the model. Ambiguous conversation should remain conversational; journey creation requires a service-oriented interpretation with sufficient confidence.

1. Confirm the workflow catalog is seeded.
1. Confirm the provider response passed schema validation.
1. Inspect redacted routing reasons and thresholds.
1. Retry with the AI provider disabled to compare deterministic routing.
1. Add a regression test for the exact fabricated phrase.

   Never send real identity or document content to a debugging prompt.

Journey does not persist

Verify that the creation endpoint returned a journey ID, the client stored/refetched it, and subsequent milestone calls use the same authenticated user. Check database writes and transaction rollback messages. A UI-only progress animation must follow - not replace - the persisted backend transition.

Documentation build fails

Run:

```bash
python -m mkdocs build --strict
```

The error normally identifies a missing navigation file, invalid Markdown reference, or template issue. Do not commit the generated `site/` directory.

SECTION 07

Project

This section is part of the consolidated eAkto technical and product documentation.

eGovPH Hackathon 2026

eAkto is an eGovPH Hackathon 2026 prototype for guiding citizens through government journeys one clear step at a time.

The problem

Government updates often span agencies, documents, eligibility rules, and unfamiliar terminology. A citizen may understand the life change - marriage, a lost ID, or an incorrect record - without knowing the official service name or the order of work.

eAkto turns that plain-language request into a reviewable journey:

1. understand the citizen's request;
2. ask only the missing questions;
3. suggest a supported, structured workflow;
4. show required documents and why they matter;
5. preserve progress across milestones;
6. keep reusable government documents in a protected vault.

Prototype capabilities

1. eGovPH SSO and eVerify-oriented authentication flows;
1. identity assurance for sensitive actions;
1. conversational intent assessment with deterministic safeguards;
1. versioned journey templates and progress tracking;
1. dynamic questions and document requirements;
1. private document metadata and upload flows;
1. Flutter web/mobile client backed by a FastAPI API.

Design principles

Citizen language first. The user describes what happened instead of selecting an agency form.

Review before action. AI-assisted understanding is shown for confirmation. The model does not silently complete an official transaction.

Structured execution. Once selected, journeys use versioned questions, evidence, milestones, and policy rules.

Identity-aware security. Sensitive operations require a current authenticated session and, where needed, stronger identity assurance.

Honest status. eAkto distinguishes preparation, document readiness, and local journey completion from an agency-confirmed submission.

Hackathon status

The current repository demonstrates the product and technical direction. Workflow definitions are prototype catalog entries and must be validated with the responsible agencies before production. Provider access, branding, privacy obligations, and deployment controls remain subject to the authorized hackathon and partner agreements.

Scope and limitations

This documentation describes the behavior implemented on the current `frontend`-derived code line. It does not claim that the prototype is an accredited government production service.

Supported workflow templates

The checked-in catalog currently focuses on:

1. post-marriage government-record updates;
1. direct government-record information updates.

The architecture supports additional versioned templates, but a newly imagined journey is not automatically authoritative. Questions, documents, agency responsibilities, and submission rules require validation.

What the prototype does

1. captures and classifies citizen requests;
1. keeps ambiguous conversation in chat;
1. prepares a reviewable intent assessment;
1. creates structured local journeys for supported requests;
1. persists milestone, answer, and document state;
1. manages private document metadata and upload paths;
1. integrates with configured identity and AI provider adapters.

What it does not guarantee

1. final acceptance by an agency;
1. legal completeness of every requirement;
1. automatic mutation of a government system of record;
1. production-grade identity accreditation;
1. permanent availability of hackathon sandbox providers;
1. universal coverage of Philippine government services;
1. safe operation without the production controls listed below.

Known technical limitations

Area	Current limitation
Agent API	Authentication and ownership dependencies are not consistently applied; this is a production blocker.
Workflow catalog	The seed set is intentionally small and requires agency review.
Database operations	Local development favors SQLite and startup compatibility; production needs managed migrations and rollback procedures.

Area	Current limitation
Documents	Upload protection still needs production malware scanning, retention, and deletion policies.
Provider sandboxes	Codes, tokens, and sessions may be short-lived, single-use, rate-limited, or unavailable.
Legacy code	Some older route/repository modules remain in the tree but are not mounted in the current API.
Operations	The repository does not include a complete production platform, monitoring stack, or incident runbook.

Restricted integration material

Detailed eGovPH and eVerify partner specifications are made available through authorized channels. They may contain non-public endpoints, credential requirements, or operational constraints. eAkto documentation therefore records only:

1. eAkto-owned endpoints and data boundaries;
1. public configuration variable names;
1. high-level flow and security responsibilities;
1. links or references appropriate for authorized developers.

It does not reproduce private specifications, examples containing live identifiers, or any credential value.

Configuration reference

The backend loads configuration from process environment variables and, for local development, `backend/.env`. Copy `backend/.env.example`; do not commit the resulting file.

Values shown as secret must exist only on the backend. The Flutter application must never receive them.

Application

Variable	Secret	Purpose
`APP_NAME`	No	Service display name.
`APP_VERSION`	No	API version reported by the service.
`APP_ENV`	No	Environment such as `development`, `test`, or `production`.
`API_PREFIX`	No	Mounted API prefix; default is `/api/v1`.
`CORS_ORIGINS`	No	Comma-separated allowed browser origins. Use exact production origins.
`DATABASE_URL`	Yes	SQLAlchemy database connection string. Treat embedded credentials as secret.
`WORKFLOW_CANDIDATE_MIN_CONFIDENCE`	No	Minimum score for retaining workflow candidates.
`HEURISTIC_AUTO_ROUTE_THRESHOLD`	No	Confidence above which deterministic routing may resolve directly.
`HEURISTIC_AI_FALLBACK_THRESHOLD`	No	Boundary for using AI as a fallback to heuristic assessment.
`AI_WORKFLOW_SELECTION_THRESHOLD`	No	Confidence required to select an AI-suggested workflow.

eGov AI

Variable	Secret	Purpose
`EGOV_AI_BASE_URL`	No	Authorized provider environment URL.
`EGOV_AI_ACCESS_CODE`	Yes	Server-side provider credential.
`EGOV_AI_CATEGORY`	No	Provider category used by the adapter.
`EGOV_AI_GENERATION_ENABLED`	No	Enables provider-backed generation when configured.
`EGOV_AI_REQUEST_TIMEOUT_SECONDS`	No	Request timeout.
`EGOV_AI_MINIMUM_CREDITS`	No	Minimum available credits before an eligible request.
`EGOV_AI_CREDIT_CACHE_SECONDS`	No	Duration for cached credit status.
`EGOV_AI_MAX_DOCUMENT_BYTES`	No	Maximum document size sent through the AI adapter.

eGovPH SSO

Variable	Secret	Purpose
`EGOVPH_SSO_BASE_URL`	No	Authorized SSO environment base URL.
`EGOVPH_SSO_PARTNER_CODE`	Sensitive	Partner identifier; keep it server-side.

Variable	Secret	Purpose
EGOVPH_SSO_PARTNER_SECRET	Yes	Partner credential used only by the backend.
EGOVPH_SSO_REQUEST_TIMEOUT_SECONDS	No	Provider request timeout.
APPLICATION_SESSION_TTL_HOURS	No	eAkto session lifetime.

There is no required EGOVPH_SSO_AUTHORIZE_URL setting in the current application. The backend builds its flow from the configured provider contract and eAkto callback.

Face liveness and eVerify

Variable	Secret	Purpose
FACE_LIVENESS_BASE_URL	No	Liveness integration base URL.
FACE_LIVENESS_API_KEY	Yes	Backend liveness credential.
FACE_LIVENESS_CALLBACK_URL	No	Allowed callback into the eAkto frontend.
FACE_LIVENESS_REQUEST_TIMEOUT_SECONDS	No	Liveness request timeout.
FACE_LIVENESS_CONFIDENCE_THRESHOLD	No	Minimum accepted confidence under current policy.
EVERIFY_BASE_URL	No	Authorized eVerify environment base URL.
EVERIFY_CLIENT_ID	Sensitive	Backend client identifier.
EVERIFY_CLIENT_SECRET	Yes	Backend client secret.
EVERIFY_PUBLIC_KEY	Public	Public key passed to an approved client-side verification flow.
EVERIFY_IDENTITY_REFERENCE_SECRET	Yes	Secret used to protect internal identity references.
EVERIFY_REQUEST_TIMEOUT_SECONDS	No	eVerify request timeout.
EVERIFY_SIGN_IN_CHALLENGE_TTL_MINUTES	No	Sign-in challenge lifetime.
IDENTITY_ASSURANCE_TTL_MINUTES	No	High-assurance identity lifetime.

Despite its name, use EVERIFY_PUBLIC_KEY only for the purpose and origin allowed by the provider. A public key is not a substitute for the server credentials.

Supabase documents

Variable	Secret	Purpose
SUPABASE_URL	No	Supabase project URL.
SUPABASE_SECRET_KEY	Yes	Server-side key for protected database/storage operations.
SUPABASE_DATABASE_URL	Yes	Direct database URL when used.
SUPABASE_DOCUMENT_BUCKET	No	Private document bucket name.
SUPABASE_DOCUMENT_TABLE	No	Citizen document metadata table.
SUPABASE_REQUEST_TIMEOUT_SECONDS	No	Supabase request timeout.
DOCUMENT_MAX_UPLOAD_BYTES	No	Maximum accepted upload size.

Never use a privileged Supabase key in Flutter. Enforce private bucket policy even if the backend already checks ownership.

Flutter

Build definition	Secret	Purpose
`API_BASE_URL`	No	Public eAkto API URL including `/api/v1`.

Pass it at run or build time:

```bash
flutter run -d chrome \
  --dart-define=API_BASE_URL=http://localhost:8000/api/v1
```

Safe validation

Check that required names exist without printing their values. In production, prefer platform health checks and secrets-manager metadata over dumping the process environment.

References

These references explain the public frameworks and standards used by eAkto. Provider-specific partner documents are intentionally not mirrored here.

eGov ecosystem

1. eGov AI developer portal - authorized AI service entry point used by the backend adapter.
1. eGovPH SSO partner documentation - restricted material supplied through authorized eGovPH channels.
1. NIDAS eVerify and Face Liveness integration documentation - restricted material supplied through authorized eGovPH channels.

   Access to a provider document does not make it safe to publish. Follow its distribution terms and use the current sandbox/production contract issued to the project.

Flutter client

1. Flutter application architecture guide
1. Flutter testing overview
1. go_router package
1. Riverpod documentation
1. Dio package
1. flutter_secure_storage package

FastAPI server

1. FastAPI dependencies
1. FastAPI security
1. FastAPI larger applications
1. Pydantic settings
1. SQLAlchemy 2.0 documentation
1. Supabase Storage access control

Documentation and security

1. MkDocs configuration
1. MkDocs custom themes
1. OWASP API Security Top 10
1. OWASP Mobile Application Security
1. National Privacy Commission - Data Privacy Act

Source-of-truth order

When two references disagree, use this order:

1. applicable law and binding government/provider agreement;
2. current authorized provider contract for the selected environment;
3. eAkto API schemas and automated tests;
4. this documentation;
5. examples and historical code.

Report documentation drift instead of silently coding to an outdated example.

Glossary

Active journey
A citizen journey that has been created and is not yet completed or archived.

Agent
Backend capabilities that guide a citizen, track follow-up state, or prepare document assistance. It is not an autonomous government decision-maker.

Assurance
A time-limited record that the authenticated user completed the required identity verification level.

Catalog
The approved set of workflow definitions eAkto can suggest and execute.

Citizen intent
A structured interpretation of a free-text request, including task, possible agencies, confidence, and missing information.

Document vault
The citizen-facing view of protected government document metadata and upload access.

eGov AI
An external eGovernment AI service used by an authorized backend adapter. Deterministic eAkto policy validates and constrains its output.

eGovPH SSO
The partner single sign-on flow used to establish an eAkto application session from an eGovPH identity.

eVerify
An identity verification service used to establish stronger identity assurance.

Evidence requirement
A document or fact required by a workflow action, sometimes conditional on an answer.

Exchange code
A short-lived, single-use authorization artifact that the backend exchanges under the authorized SSO contract. It is not a reusable access token.

High-assurance action
An operation, such as creating a sensitive journey or accessing documents, that requires recent identity assurance in addition to authentication.

Journey
A persisted instance of a workflow for one citizen, including answers, milestones, documents, and progress.

Milestone
An ordered stage in a journey. Milestones contain actions and completion rules.

Provider adapter
Backend code that translates between an external service contract and eAkto-owned types.

Signed URL
A short-lived object-storage URL issued after authorization. It must not be logged or treated as permanent.

Workflow definition
A versioned template defining questions, documents, milestones, actions, and rules for a supported government journey.