# Contents



# 1. Administrator Manual

## 1.1. Configuration

Deployment options which are specific to the environment are configured with environment variables. The primary app configuration which is agnostic to the environment is done through `app-config.yaml` in the root of the repository, and includes the questions asked and the system prompt templates for example.

### 1.1.1. Global

| Environment Variable | Default Value | Description |
|---|---|---|
| `MAX_INPUT_TOKENS` | `2000` | Instructs the inference provider to not allow more than `MAX_INPUT_TOKENS` number of input tokens. Useful to bound the cost of inference and malicious requests |
| `MAX_OUTPUT_TOKENS` | `800` | Instructs the inference provider to not output more than `MAX_OUTPUT_TOKENS` number of output tokens. Useful to bound the cost of inference |

### 1.1.2. `app-config.yaml`

The backend's text generation is controlled by the inference section of `app-config.yaml`. It contains two prompts, a user prompt and a system prompt. The system prompt should contain the

instructions for the model, the specific task it needs to perform. The user prompt should contain the user's input, formatted in a way to assist the model's understanding of it.

Both the system and user prompts are Go templates. The system prompt is templated with the current time (key `CurrentTime`), and the user prompt is templated with the user's responses to the survey questions (each keyed by the question's `name`).

```
inference:
  systemPrompt: |
    Rewrite the input into more formal language.
  userPrompt: |
    The tenant's main problems are {{.mainProblem}}.
```

The frontend UI elements are defined in app-config.yaml as well. This configuration file should act as an easy place for the administrator to update any text they want on the frontend including the title, the landing page of the website (and all elements within it), the button text, the questions, the terms of service, the tips, and any other text elements on the frontend. This allows for easy testing as well, as the tests read from this configuration file, so updates to text on the site do not break the CI pipeline.

#### 1.1.3. Inference

There are multiple backend inference providers which require different configuration.

##### 1.1.3.1. Ollama

Ollama is an open source, self hosted LLM inference runner.

| Environment Variable | Example | Description |
|---|---|---|
| `INFERENCE_PROVIDER` | `ollama` | Must be `ollama` |
| `OLLAMA_HOST` | `http://ollama:11434` | URL to a running Ollama instance reachable by the container |
| `OLLAMA_MODEL_ID` | `gemma3:4b` | Model ID. The list of model IDs are available at `ollama.com/search` |

##### 1.1.3.2. Amazon Bedrock

Amazon Bedrock is a managed LLM inference runner. It provides very large and fast models, and is billed per token. Also, the user of Bedrock as an inference provider does not require running the rest of the application on AWS, so long as `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are set to an IAM user which can assume the `AWS_ACCESS_KEY_ID` role.

| Environment Variable | Example | Description |
|---|---|---|
| `INFERENCE_PROVIDER` | `aws` | Must be `aws` |
| `AWS_REGION` | `us-east-2` | Region which the Bedrock model is available in. See `AWS_BEDROCK_MODEL_ID` for more information |
| `AWS_ACCESS_KEY_ID` | `AKIAXXXXXXXXXXXXXXXX` | Access key to a user which can assume `AWS_BEDROCK_ROLE_ARN` role. Not required if the container is running on AWS with |

| Environment Variable | Example | Description |
|---|---|---|
| | | proper IAM assume role principals |
| AWS_SECRET_ACCESS_KEY | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX | Secret access key to a user which can assume AWS_BEDROCK_ROLE_ARN role. Not required if the container is running on AWS with proper IAM assume role principals |
| AWS_BEDROCK_ROLE_ARN | arn:aws:iam::XXXXXXXXXXXX:role/YYY | AWS role ARN which has the inline policy JSON below |
| AWS_BEDROCK_MODEL_ID | meta.llama3-3-70b-instruct-v1:0 | Must be a Model ID from the list available at docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html. Note that not all models are available in all regions |

#### 1.1.3.2.1. IAM Inline Policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "BedrockAPIs",
            "Effect": "Allow",
            "Action": [
                "bedrock:InvokeModel"
            ],
            "Resource": "*"
        }
    ]
}
```

### 1.1.3.3. OpenAI

The OpenAI inference provider can attach to any OpenAI v1 compatible API. This includes OpenAI's API as well as others like the NRP cluster managed LLMs.

| Environment Variable | Example | Description |
|---|---|---|
| INFERENCE_PROVIDER | openai | Must be openai |
| OPENAI_API_KEY | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX | API Key to use for requests |
| OPENAI_MODEL_ID | gemma3 | The model to use for inference. NRP models listed at https://nrp.ai/documentation/userdocs/ai/llm-managed/#available-models. OpenAI models listed at https:// |

| Environment Variable | Example | Description |
|---|---|---|
| | | platform.openai.com/docs/models |
| OPENAI_BASE_URL | https://ellm.nrp-nautilus.io/v1 | Base URL for an OpenAI compatible API |

#### 1.1.3.4. Mock

When no other inference provider is specified, it defaults to the mock inference provider. Alternatively, it can be forced by setting `INFERENCE_PROVIDER=mock`. The mock inference provider is available in all container images. It simply echoes back the input as the output, and does not use an LLM.

## 1.2. Deployment

Continuous integration and continuous delivery build new containers weekly on Tuesdays at 4am UTC with the latest updates from all dependencies. A dashboard of the artifacts is available at `github.com/capstone-au2025/project/pkgs/container/project`.

Certain values of the `INFERENCE_PROVIDER` environment variable only work when using the correct container image. The available images are provided below:

| Container Image | Available Environment Variable Value |
|---|---|
| ghcr.io/capstone-au2025/project:aws | INFERENCE_PROVIDER=aws |
| ghcr.io/capstone-au2025/project:ollama | INFERENCE_PROVIDER=ollama |
| ghcr.io/capstone-au2025/project:openai | INFERENCE_PROVIDER=openai |

### 1.2.1. Recipes

#### 1.2.1.1. Local

A production ready docker compose file is available at `infra/docker-compose/docker-compose.yaml` in the repository which can be copied. It is also reproduced below for reference.

This configuration uses the `gemma3` model from Google at 4 billion parameters, which strikes a balance of usable outputs while still being runnable on a few CPUs. It also uses `watchtower` to automatically pull updates to the containers as they come out to ensure the latest security updates are always applied. The server is available on port 3001 which should be configured by a reverse proxy of choice to expose over HTTPS to the internet.

```yaml
services:
  backend:
    image: ghcr.io/capstone-au2025/project:ollama
    ports:
      - 3001:3001
    environment:
      - INFERENCE_PROVIDER=ollama
      - OLLAMA_HOST=http://ollama:11434
      - OLLAMA_MODEL_ID=gemma3:4b
    depends_on:
      - ollama
  ollama:
    image: ollama/ollama
    volumes:
      - ./ollama:/root/.ollama
```

```
watchtower:
  image: beatkind/watchtower
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
```

#### 1.2.1.2. NRP Kubernetes Cluster

Kubernetes manifests are available in the repository to run on the National Research Platform. To create the resources, simply run the following command:

```
kubectl -n landlord-letters apply -f infra/nrp/manifest.yaml
```

The website will then be available at letter-generator.nrp-nautilus.io.

# 2. Developer Manual

## 2.1. File Overview

- `https://github.com/capstone-au2025/project` — Project root directory
  - `backend/` — Directory containing all backend code
    - `typst-wrapper/` — Directory containing sandboxed Typst wrapper
      - `main.go` — Executable using Landlock to sandbox Typst
    - `analytics.go` — Implementation of analytics
    - `analytics_test.go` — Tests for analytics implementation
    - `aws.go` — Implementation of Amazon Bedrock inference provider
    - `aws_test.go` — Tests for Amazon Bedrock inference provider
    - `go.mod` — Golang module file
    - `go.sum` — Golang dependency checksum file
    - `inference.go` — Definition of inference provider interface, and mock implementation
    - `inference_test.go` — Tests for mock inference provider implementation
    - `letter-template.typst` — Typst template to render generated PDFs with
    - `letter.go` — Implementation of PDF letter rendering
    - `main.go` — Entrypoint and server route definitions
    - `main_test.go` — Tests of server routes
    - `ollama.go` — Implementation of Ollama local inference provider
    - `ollama_test.go` — Tests for Ollama local inference provider
    - `openai.go` — Implementation of OpenAI inference provider
  - `frontend/` — Directory containing all frontend code
    - `public/` — Directory containing the Favicon and other public resources
      - `favicon.svg` — Favicon displayed in browser tab
    - `src/` — Directory containing the source code for the frontend
      - `index.html` — HTML entry point that mounts the React application
      - `main.tsx` — React DOM initialization and app mounting
      - `App.tsx` — Main app component that initializes React Query and loads configuration
      - `index.css` — Global stylesheet definitions
      - `App.css` — Component-level styling
      - `vite-env.d.ts` — TypeScript type definitions for Vite
      - `assets/` — Directory containing image assets
        - `icons8-home.svg` — Home icon SVG asset
      - `components/` — Directory containing React components
        - `FormContainer.tsx` — Main routing hub and state management for the form workflow
        - `FormPage.tsx` — Generic form page component for rendering configurable questions

        - `IntroPage.tsx` — Landing page introducing features and getting started
        - `AddressPage.tsx` — Form page for collecting sender and recipient addresses
        - `EditPage.tsx` — Form page for allowing tenants to edit letter before generation
        - `LegalDisclaimer.tsx` — Component for displaying legal disclaimers
        - `TOSPage.tsx` — Terms of Service display and acceptance page
        - `SubmittedPage.tsx` — Success page with PDF download and certified mail options
        - `PageLayout.tsx` — Common layout wrapper for consistent page structure
        - `QuestionBox.tsx` — Individual form field component for user input
        - `ProgressIndicator.tsx` — Visual step indicator showing form completion progress
        - `icons.tsx` — SVG icon components used throughout the UI
      - `config/` — Directory containing code to load configuration file for frontend
        - `configLoader.ts` — Loads YAML configuration at build time and exports TypeScript interfaces for UI setup
    - `tests/` — Directory containing the unit tests for the frontend
      - `setup.ts` — Test environment configuration
      - `app.test.tsx` — Unit tests for the main App component
      - `FormContainer.test.tsx` — Tests for routing and state management
      - `FormPage.test.tsx` — Tests for the form page component
      - `IntroPage.test.tsx` — Tests for the landing page
      - `AddressPage.test.tsx` — Tests for address input functionality
      - `TOSPage.test.tsx` — Tests for Terms of Service page
      - `SubmittedPage.test.tsx` — Tests for the success page
      - `QuestionBox.test.tsx` — Tests for individual form field component
      - `ProgressIndicator.test.tsx` — Tests for progress indicator
      - `certifiedmail.test.ts` — Tests for certified mail utilities
    - `package.json` — Project dependencies and scripts for React, TypeScript, Vite, and testing
    - `vite.config.ts` — Build configuration with dev server proxy to backend API
    - `tsconfig.json` — TypeScript compiler configuration
    - `tsconfig.app.json` — TypeScript configuration for application code
    - `tsconfig.node.json` — TypeScript configuration for build tools
    - `eslint.config.js` — Linting rules for code quality
    - `.gitignore` — Git ignore patterns for node_modules and build artifacts
    - `app-config.yaml` — Symlink to shared YAML configuration for UI text and form questions
    - `README.md` — Project documentation for frontend
    - `reset-storage.html` — Utility page for clearing browser localStorage during development
  - `infra/` — Directory containing different recipes for infrastructure deployment
    - `docker-compose/` — Directory to deploy locally with Ollama
      - `docker-compose.yaml` — Production ready Docker Compose file to deploy locally
    - `nrp/` — Directory to deploy to the National Research Platform (NRP)
      - `manifest.yaml` — Kubernetes manifest containing all resources required to deploy
  - `Dockerfile` — Production Dockerfile used to build main container image in CI
  - `Dockerfile.backend-dev` — Development Dockerfile for backend
  - `Dockerfile.frontend-dev` — Development Dockerfile for frontend
  - `app-config.yaml` — Main app configuration
  - `docker-compose.yaml` — Development Docker Compose file
  - `flake.nix` — Optional Nix Flake for development dependency management

- `openapi.yaml` — OpenAPI schema to communicate between backend and frontend
- `user-manual.typ` — Source Typst code for this document
- `user-manual.pdf` — PDF rendered version of this documentation

## 2.2. Development Environment

In a terminal, run the following command to launch development servers for both the frontend and backend:

```
docker compose up --watch --build --remove-orphans
```

Then go to http://localhost:5173 for the frontend. The backend is also on the same port, but under `/api`, so http://localhost:5173/api/hello for example.

The default inference provider for development is a local Ollama instance on CPU, and is configured to work out of the box with the docker compose watch command.

## 2.3. Production Build

Container images are built automatically as described in Section 1.2. To build for production locally, follow these steps:

Build the `Dockerfile` at the root of the repository:

```
docker build -t project .
```

The server is running on port 3001 by default. For example, it can then be run with the following command:

```
docker run --rm -p 3001:3001 project
```

Then it will be available at http://localhost:3001.

The default inference provider for production is Amazon Bedrock 1.1.3.2, and requires configuration through its environment variables.

The available inference provider for production can be configured with a build argument to docker:

```
docker build --build-arg BUILD_TAGS=openai,aws,ollama -t project .
```

## 2.4. Build Tags

The available build tags are as follows. They should be set in your editor settings for `gopls` to provide accurate autocomplete.

- `aws`
- `ollama`
- `openai`

Having multiple backends helps improve developer experience so that external services do not need to be configured or payed for. However different backends requires different golang packages. The packages are also all auto updating. By using build tags, if one package has breaking changes, the other providers can still be built and the update process can continue.

## 2.5. Unit Tests

Run the following command from the root of the repository to test the backend:

```
docker build -t backend backend && docker run --env-file .env -it backend go test -tags aws,ollama -v
```

Note that the required environment variables must be set to run the tests for each inference provider.

The unit tests are required to pass in CI in order for the inference provider container image to be published.

Run the following in the frontend directory to test the frontend:

```
npm run test
```

And run the following in the frontend directory to test with a coverage report, which allows you to see where you need to add unit tests for new components or pages added later:

```
npm run coverage
```

The frontend uses Vitest as its testing framework paired with React Testing Library for component testing. This test suite covers all major components and utilities, including routing logic, form submission flows, and state management. Tests are configured with happy-dom as the dom environment and utilize a shared setup file for consistent test configuration. The project includes npm scripts for running tests and generating coverage reports shown above, which helps ensure code quality and prevent regressions as the application evolves in the future.

# 3. API Consumer Manual

### 3.1. `/healthz`

### 3.2. `/api/pdf`

### 3.3. `/api/text`

# 4. User (Tenant Using Tool) Manual

**The general user flow will be as follows:**

1. User lands on the landing page - they receive basic information about the website like the title, a quick summary of what the tool does, some information about how long it takes to use and that it is completely free, and some information about what they may need to answer the questions in the following section. Additionally, they are presented with an option to agree to the terms of service or click into the terms of service to read them and agree there. This tool only takes around 3 - 5 minutes to use.
2. Once the user has agreed to the terms of service and clicked get started, they are presented with the first page of questions. This page of questions asks them critical information like what problems are occurring with their house, where these problems are happening, and when each problem started. Users are also presented with a tip at the top telling them to be specific, and example text in the boxes that disappears when they start typing, but gives them an idea of what is expected out of each answer. ‘
3. Users will progress through 2 more pages of questions, with the last page of questions allowing them to continue and generate the text output that will be placed in their letter. They must also do a Captcha before progressing here, to protect the inference endpoint in the next portion of the tool from bots.
4. Upon landing on the text edit page, the user can proof read and edit the text that will become part of their letter, or go back and provide new answers to questions if they feel that this is easier.
5. Once the user feels that the generated and edited text fits their requirements, they can click generate letter. They will be redirected to a page asking them to answer basic address information

about themselves and their landlord, which goes into the final PDF version of the letter and is passed to the certified mail provider if they choose to use that option.

6. Finally, the user can decide to download the letter, or progress to the certified mail provider's website, where their information is already filled out. The user can create an account here by entering their email, and only has to enter payment information to send their letter. They can also access proofs of their letter, and will receive receipts through email, as with any other certified mail service.