

# Contents

1. Administrator Manual .....	1
1.1. Configuration .....	1
1.1.1. Global .....	1
1.1.2. app-config.yaml .....	1
1.1.3. Inference .....	2
1.1.3.1. Ollama .....	2
1.1.3.2. Amazon Bedrock .....	2
1.1.3.2.1. IAM Inline Policy .....	3
1.1.3.3. OpenAI .....	3
1.1.3.4. Mock .....	4
1.2. Deployment .....	4
1.2.1. Recipes .....	4
1.2.1.1. Local .....	4
1.2.1.2. NRP Kubernetes Cluster .....	5
2. Developer Manual .....	5
2.1. File Overview .....	5
2.2. Development Environment .....	7
2.3. Production Build .....	7
2.4. Build Tags .....	7
2.5. Unit Tests .....	7
3. API Consumer Manual .....	8
3.1. /healthz .....	8
3.2. /api/pdf .....	8
3.3. /api/text .....	8
4. User (Tenant Using Tool) Manual .....	8

## 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 <code>MAX_INPUT_TOKENS</code> 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 <code>MAX_OUTPUT_TOKENS</code> 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
<code>INFERENCE_PROVIDER</code>	<code>ollama</code>	Must be <code>ollama</code>
<code>OLLAMA_HOST</code>	<code>http://ollama:11434</code>	URL to a running Ollama instance reachable by the container
<code>OLLAMA_MODEL_ID</code>	<code>gemma3:4b</code>	Model ID. The list of model IDs are available at <a href="#">ollama.com/search</a>

#### 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
<code>INFERENCE_PROVIDER</code>	<code>aws</code>	Must be <code>aws</code>
<code>AWS_REGION</code>	<code>us-east-2</code>	Region which the Bedrock model is available in. See <code>AWS_BEDROCK_MODEL_ID</code> for more information
<code>AWS_ACCESS_KEY_ID</code>	<code>AKIAXXXXXXXXXXXXXX</code>	Access key to a user which can assume <code>AWS_BEDROCK_ROLE_ARN</code> role. Not required if the container is running on AWS with

Environment Variable	Example	Description
		proper IAM assume role principals
AWS_SECRET_ACCESS_KEY	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	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 <a href="https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html">docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html</a> . 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	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX	API Key to use for requests
OPENAI_MODEL_ID	gemma3	The model to use for inference. NRP models listed at <a href="https://nrp.ai/documentation/userdocs/ai/llm-managed/#available-models">https://nrp.ai/documentation/userdocs/ai/llm-managed/#available-models</a> . OpenAI models listed at <a href="https://">https://</a>

Environment Variable	Example	Description
		<a href="https://platform.openai.com/docs/models">platform.openai.com/docs/models</a>
OPENAI_BASE_URL	<a href="https://ellm.nrp-nautilus.io/v1">https://ellm.nrp-nautilus.io/v1</a>	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](https://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
<code>ghcr.io/capstone-au2025/project:aws</code>	<code>INFERENCE_PROVIDER=aws</code>
<code>ghcr.io/capstone-au2025/project:ollama</code>	<code>INFERENCE_PROVIDER=ollama</code>
<code>ghcr.io/capstone-au2025/project:openai</code>	<code>INFERENCE_PROVIDER=openai</code>

### 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.

```

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](http://letter-generator.nrp-nautilus.io).

## 2. Developer Manual

### 2.1. File Overview

• <a href="https://github.com/capstone-au2025/project">https://github.com/capstone-au2025/project</a>	Project root directory
▶ backend/	Directory containing all backend code
- typst-wrapper/	Directory containing sandboxed <a href="#">Typst</a> wrapper
• main.go	Executable using <a href="#">Landlock</a> to sandbox Typst
- analytics.go	Implementation of analytics
- analytics_test.go	Tests for analytics implementation
- aws.go	Implementation of <a href="#">Amazon Bedrock inference provider</a>
- aws_test.go	Tests for Amazon Bedrock inference provider
- go.mod	<a href="#">Golang module file</a>
- go.sum	<a href="#">Golang dependency checksum file</a>
- inference.go	Definition of <a href="#">inference provider</a> interface, and <a href="#">mock implementation</a>
- 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 <a href="#">Ollama local inference provider</a>
- ollama_test.go	Tests for Ollama local inference provider
- openai.go	Implementation of <a href="#">OpenAI inference provider</a>
▶ 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
▶ 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 <u>recipes</u> 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 <u>National Research Platform (NRP)</u>
• manifest.yaml	<u>Kubernetes manifest</u> containing all resources required to deploy
▶ Dockerfile	Production <u>Dockerfile</u> 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 <u>Docker Compose</u> file
▶ flake.nix	Optional <u>Nix Flake</u> for development dependency management
▶ openapi.yaml	<u>OpenAPI</u> 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**