Apodeixi 2: POC-1

Contents

[1 Purpose of POC 1 1](#_Toc95408423)

[2 Use Case 1](#_Toc95408424)

[3 Component Diagram 2](#_Toc95408425)

[4 Information Architecture 2](#_Toc95408426)

[5 Deployment Diagram 3](#_Toc95408427)

[6 DevOps 3](#_Toc95408428)

[6.1 Repos 3](#_Toc95408429)

[6.2 Packaging 3](#_Toc95408430)

[6.3 Pipeline 3](#_Toc95408431)

# Purpose of POC 1

To validate the proposed technical stack for decoupling Apodeixi 1.0’s monolith, specifically:

* Resource-oriented decomposition, with interactions mediated via REST APIs
* Use of Falcon as a framework to build REST microservices
* Use of waitress as a Windows WebServer running Falcon applications (1 per microservice)
* Use of pytest to support a TDD development approach to the above, as a way to build a modus operandi for using sprints to build Apodeixi 2.0 by incrementally adding features in a way that ensures having running software for each sprint.
* Use of mocks in developing tests
* Use of the coverage Python module to assess test coverage
* Assess the performance penalty of out-of-process hops
* Click CLI to invoke microservices
* New-generation parser that removes need for much of Apodeixi 1.0’s controller logic specific to a posting.
* Basic form generator.

POC-1 will run on Windows 10 or above.

# Use Case

Functionality of POC-1 is: ability to post and get forms for a particular postings API.

The postings API that will be used in POC-1 is inspired by Apodeixi 1.0’s big-rocks.journeys.a6i posting API.

These CLI commands comprise the POC-1 functionality:

$ apo post [EXCEL FILE]

$ apo get form big-rocks.journeys.a6i my-corp.production

These commands will be run in a certain folder structure. We denote $POC the root folder for POC-1. Then under that root the following folder must exist:

$POC/kb/excel-postings/journeys/FY 22/

These files must exist under that folder, adhering to Apodeixi 1.0’s schema for the big-rocks.journeys.a6i posting API.

Astrea/Official/Astrea.modernization.big-rocks.journeys.a6i.xlsx

Britt/Official/Britt.modernization.big-rocks.journeys.a6i.xlsx

For POC-1, CLI commands must run in one of these leaf folders, and only can post one of these files. The files are editable as per Apodeixi 1.0’s schema semantics (e.g., may add more rocks and may add additional columns for sub-rocks).

After posting, Excel files are removed from the directory. They can be brought back by the apo get form command. They will be identical to what was previously posted (i.e., for POC-1 we don’t implement the equivalent of Apodeixi 1.0’s functionality to increment version numbers or to generate UIDs).

# Component Diagram

POC-1 entails 4 components interacting over REST:

* **Client-side**: CLI running on Bash from which the CLI commands can be invoked with results displayed in the shell.
* **KnowledgeBase**: receives CLI requests and processes them, possibly delegating some or much of the processing to other components. Is the central coordinator for business logic.
* **PostingParser**: receives a filename for an Excel posting and returns one or more YAML manifests representing the parsed contents of the Excel file
* **FormGenerator**: generates and saves an Excel file for a posting API, with content coming from the most recent versions of the participating YAML manifests.
* **KnowledgeBaseStore**: provides functionality to store and retrieve YAML manifests.



# Information Architecture

We follow the conventions from [REST Resource Naming Guide (restfulapi.net)](https://restfulapi.net/resource-naming/)

The table below summarizes the top-level resource taxonomy and how it maps to the kinds of resources in the naming guide. Notice that each resource is implemented by exactly one service.

|  |  |  |
| --- | --- | --- |
| Resource | Resource Type from Naming Guide | Service exposing the resource |
| /form | Document (singular noun) | KnowledgeBase |
| /parse-form | Controller (verb) | PostingParser |
| /generate-form | Controller (verb) | FormGenerator |
| /manifests | Store (plural noun) | KnowledgeBaseStore |

# Deployment Diagram

POC-1 makes a number of deployment simplifications:

* **No networking**: Everything runs in 1 machine. Services communicate via loopback IP.
* **Singleton deployment**: For each service, there is a single instance
* **No service discovery**: REST dependencies between services use hard-coded URLs (basically, hostname:PORT\_NUMBER, where each service hardcodes the PORT\_NUMBER of other services).
* **No concurrency**: This is a single-user, one-request-at-a-time system. All requests are synchronous and services can’t handle more than 1 request at a time.



# DevOps

## Repos

In the spirit of microservices, each service must have its own dedicated repo.

## Packaging

In the spirit of microservice, each service is separately packaged and can be deployed/upgraded independently of other services.

## Pipeline

Each service has a separate packaging CI/CD pipeline. For POC-1 they will be manual pipelines: developer uses the shell to manually create and deploy the package for each individual service.