# Solid Microservices Prototype Specification

## Introduction

This document outlines the implementation plan for a modular microservices-based prototype system hosted locally on a Mac. The system includes a Solid-compliant Personal Data Store (PDS), services for issuing and verifying Verifiable Credentials (VCs), a user interface for WebID-based authentication and credential management, and integration with an existing MERN stack benefit application.

Define and implement a strategy to deliver these services incrementally in a modular fashion. Every service build should have no knowledge of internals of other services and should only communicet via well defined API. Create the parent folder and sub folders as modules and push to git. The parent should manage the overall delivery. Draft specifications for each service and put in a specifications folder in each modules repo. If we need tp copy across api spec to other modules as we build previous module put that in plan.

This needs to be a simple, modular implementation approach where we can build each module independantly of others. This is critical.

## Core Services

* **Solid PDS (Community Solid Server)**: Hosts user pods and WebIDs.
* **VC Consumer/verifier Service**: consumes VC from PDS and Verifies VCs using DID resolution and signature validation.
* **Test VC Creator Service**: Generates mock credentials (name, address, DOB, benefit awards) signed as if issued by government departments and publishes to PDS.
* **DID Registry Service**: Provides a lightweight service to register DIDs and associated DID Documents, retrieve DID Documents for verification, and optionally update or deactivate DIDs. This service ensures each microservice and external party can use decentralized identifiers in a verifiable way, supporting key lifecycle management and promoting interoperability across services.

## Consumer Services

* **React-based UI**: Allows users to register/login to PDS via WebID, request and manage VCs.
* **MERN Stack Benefit Application**: Allows user authenticate their WebID so the benefit app can Retrieve and verifies their VCs from the user's Solid Pod and integrates validated data into its existing evidence processing pipeline.

## Test VC Creator Service

This microservice generates mock Verifiable Credentials for testing allowing mern benefit app to consumepurposes. It uses Node.js and libraries such as did-jwt-vc or @digitalbazaar/vc-js to issue credentials. The credentials include attributes like name, address, date of birth, and benefit awards, and are signed using a DID (e.g., did:web or did:key).

The service exposes a REST API to issue credentials and store them in the user's Solid Pod using @inrupt/solid-client. The DID Registry Service should be used by this service to register and retrieve its DID Document.

## Suggested Container Images

* Solid PDS: solidproject/community-solid-server
* VC Issuer: Node.js base image with VC libraries (node:18-alpine)
* VC Verifier: Node.js base image with VC verification libraries
* Test VC Creator: Node.js base image with VC issuance libraries
* DID Registry Service: Python (FastAPI) or Node.js (Express) base image
* React UI: Node.js base image with React build
* MERN App: Existing MERN stack container

## suggested API Endpoints

**Test VC Creator API**:

* POST /issue-test-vc — Generate and sign a test VC
* GET /did — Retrieve the DID used for signing

**MERN App VC Module**:

* GET /vc/fetch-from-pod — Retrieve VC from user's Solid Pod
* POST /vc/verify — Verify VC signature and extract data

**DID Registry Service**:

* POST /dids — Register a new DID and its DID Document
* GET /dids/:did — Retrieve a DID Document
* GET /dids — List all registered DIDs (optional)

## Secrets and Key Management

For this prototype, secrets such as private keys and DIDs can be generated locally and stored in .env files or mounted volumes. Each service will manage its own keys independently. The DID Registry Service supports key rotation and revocation by enabling updates to DID Documents. Authentication between services will use Solid OIDC tokens.

## Deployment Strategy

The system must be as modular as possible, built as containerised microservices.

All services should define API using open spec that can be consumed by other dependent services building ontefraces. They should be high;y cohesive and loosely coupled.

Reuse existying container images wherever possible, if this cannot be done re-use existing code if possible.

Each microservice must be maintained in its own Git repository and added to a parent orchestration repository as a Git submodule. The parent repo will contain a docker-compose.yml file to build and run all services together. Services will communicate over Docker's internal network using service names.

## Step-by-Step Implementation Plan

1. Create the parent orchestration repository and initialize Docker Compose.
2. Create separate Git repositories for each microservice: Solid PDS, VC Issuer, VC Verifier, Test VC Creator, DID Registry, React UI, MERN App.
3. Add each microservice to the parent repo as a Git submodule.
4. Implement the Solid PDS using the Community Solid Server container.
5. Implement the Test VC Creator to generate mock credentials and sign them using DIDs using registry. Add them to users PDS.
6. Implement the DID Registry Service to manage DID registration, resolution, and document updates.
7. Implement the React UI for user registration, login, and VC management.
8. Extend the existing MERN stack app to retrieve and verify VCs from the user's Solid Pod.
9. Use Docker Compose in parent directory to wire all services together and expose necessary ports.
10. Use Cloudflared to expose the local environment securely if needed.
11. Ideally use test driven developmengt. Implement test framewor FIRST and then build services testing against it.

## Appendix: Implementation Details Suugestions Only.

### Sample Verifiable Credential

json

CopyEdit

{

"@context": ["https://www.w3.org/2018/credentials/v1"],

"type": ["VerifiableCredential", "GovernmentBenefitCredential"],

"issuer": "https://gov.example.com/issuer",

"issuanceDate": "2024-01-01T00:00:00Z",

"credentialSubject": {

"id": "did:web:user.example.com",

"name": "Jane Doe",

"dateOfBirth": "1990-01-01",

"benefit": "Universal Credit"

},

"proof": {

"type": "Ed25519Signature2020",

"created": "2024-01-01T00:00:00Z",

"proofPurpose": "assertionMethod",

"verificationMethod": "https://gov.example.com/issuer#key-1",

"jws": "eyJhbGciOiJFZERTQSJ9..."

}

}

### OpenAPI Spec Example (Test VC Creator)

yaml

CopyEdit

openapi: 3.0.0

info:

title: Test VC Creator API

version: 1.0.0

paths:

/issue-test-vc:

post:

summary: Issue a test Verifiable Credential

requestBody:

required: true

content:

application/json:

schema:

type: object

properties:

name:

type: string

dateOfBirth:

type: string

benefit:

type: string

responses:

'200':

description: VC issued successfully

content:

application/json:

schema:

type: object

properties:

vc:

type: object

### Recommended Folder Structure

pgsql

CopyEdit

service-name/

├── src/

│ ├── index.js

│ ├── routes/

│ └── controllers/

├── tests/

│ └── vc.test.js

├── openapi.yaml

├── Dockerfile

├── .env.example

└── README.md

### VC Creator UI Wireframe Description

The UI should include:

* WebID login button
* Form fields: Name, Date of Birth, Benefit Type
* 'Generate VC' button
* VC preview panel
* 'Store in Pod' button

### Solid Pod Setup Instructions

Use Community Solid Server locally:

1. Run docker run -p 3000:3000 solidproject/community-solid-server
2. Register a WebID at [http://localhost:3000](http://localhost:3000/)
3. Use the WebID to authenticate in the VC Creator UI

### Sample did:web DID Document

json

CopyEdit

{

"@context": "https://www.w3.org/ns/did/v1",

"id": "did:web:gov.example.com",

"verificationMethod": [{

"id": "did:web:gov.example.com#key-1",

"type": "Ed25519VerificationKey2020",

"controller": "did:web:gov.example.com",

"publicKeyBase58": "DmgN7..."

}],

"assertionMethod": ["did:web:gov.example.com#key-1"]

}

### Error Handling Guidance

* Return 400 for malformed VC requests.
* Return 401 for unauthorized access to Solid Pod.
* Return 500 for internal errors with stack trace in logs.

### User Roles and Personas

* **Citizen**: Uses WebID to login and manage VCs.
* **Developer**: Builds and tests microservices.
* **Caseworker**: Uses MERN app to verify VCs.

### .env.example Template

makefile

CopyEdit

PORT=3001

DID\_SEED=abc123

SOLID\_IDP=https://localhost:3000

WEBID=https://localhost:3000/profile/card#me

PRIVATE\_KEY=-----BEGIN PRIVATE KEY-----

...

-----END PRIVATE KEY-----

### Local Development and CI Instructions

To run locally:

1. Clone the parent repo and submodules.
2. Run docker-compose up.
3. Access services via localhost ports.

To run tests:

1. Navigate to each service folder.
2. Run npm install && npm test.

CI can be added using GitHub Actions with workflows for linting and testing.

### DID Registry Service Technical Details (Moved from standalone section)

#### Purpose

A lightweight service to:

* Register DIDs and associated DID Documents.
* Retrieve DID Documents for verification.
* Optionally update or deactivate DIDs (for more advanced use cases).

#### Core Features

| Function | Description |
| --- | --- |
| registerDID | Register a new DID and its DID Document |
| getDIDDocument | Retrieve a DID Document by DID |
| listDIDs | List all registered DIDs (optional) |
| updateDIDDocument | Update an existing DID Document (optional) |
| deactivateDID | Mark a DID as inactive (optional) |

#### API Design (RESTful)

* **POST /dids**: Register a new DID.
* **GET /dids/:did**: Retrieve a DID Document.
* **GET /dids**: List all registered DIDs (optional).

#### Suggested Tech Stack

| Component | Tech |
| --- | --- |
| Language | Python (FastAPI) or Node.js (Express) |
| Storage | SQLite or MongoDB |
| DID Tools | DIDKit or Veramo |
| Container | Docker |
| DID Method | did:key or did:web for simplicity |

#### Example Project Structure (FastAPI)

css

CopyEdit

did-registry/

├── app/

│ ├── main.py

│ ├── models.py

│ ├── routes.py

│ └── db.py

├── Dockerfile

├── requirements.txt

└── README.md

#### Example Dockerfile (FastAPI)

sql

CopyEdit

FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY ./app /app

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

#### Example requirements.txt

nginx

CopyEdit

fastapi

uvicorn

pydantic

sqlalchemy

#### Existing Containers You Can Use or Extend

1. SpruceID DIDKit Docker Image — includes CLI and HTTP server for DID and VC operations.
2. Veramo Agent Docker Template — full-featured agent with DID and VC support, can be extended to include registry endpoints.