ASDs

Full design

Product description

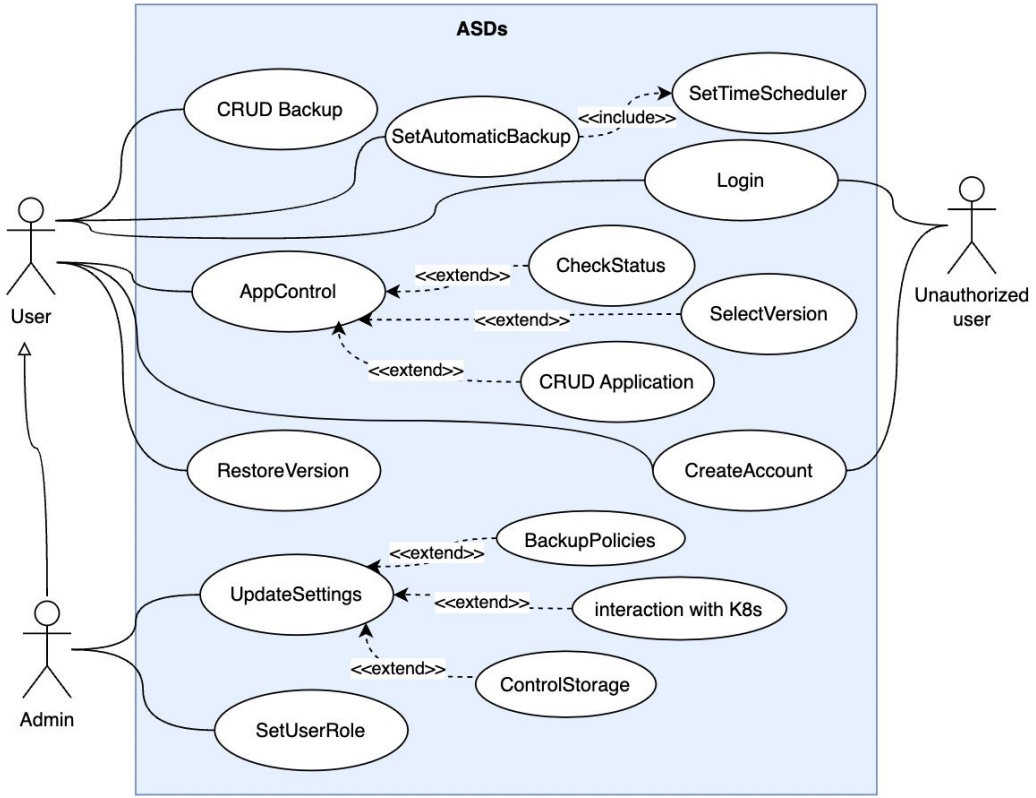
An application continuity service dynamically backs up and restores the state of applications running within the framework on K8s. The service uses the DSL of the framework to determine the relevant application state and its backup policies. Developers of an application are able to backup and restore a specific version from images and relevant state from the service. The application state is stored externally on the given S3-compatible object storage.

Team: Gorelyi Mikhail, Lukashin Daniil, Derezovskiy Ilya, Sigal Lev

Repo: https://github.com/gorelyi-code/advanced_software_designers

Report: a link to this slides within project repo/doc storage

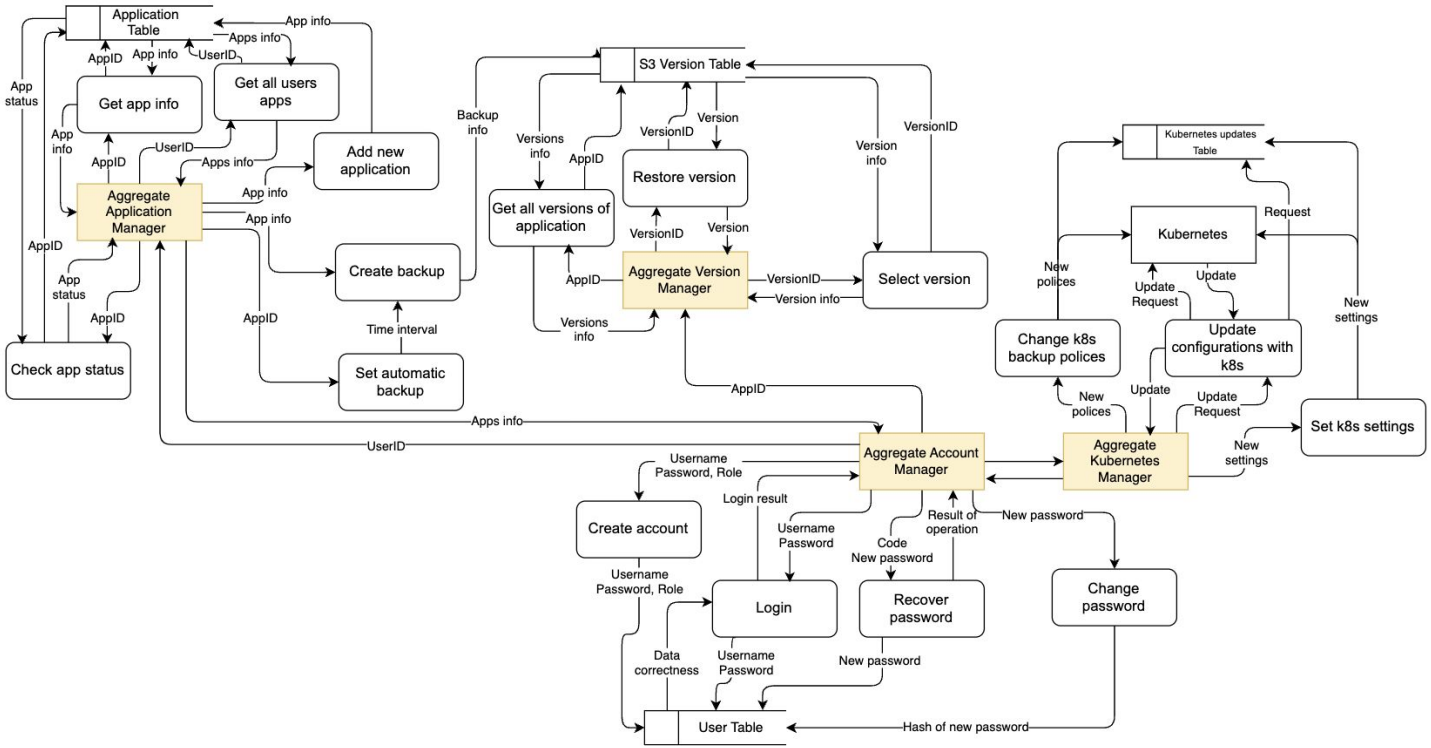
Use case diagram



CRUD: create, read, update, and delete

Textual description in github:
https://github.com/gorelyi-code/advanced_software_designers/blob/main/Use%20case%20diagram/main_scenarios.md

System architecture



System architecture

Micro service architecture

Principle: The system should be designed as a set of independent microservices that interact via an API.

Rationale: Allows for scalability, flexibility, simplified deployment and fault isolation.

Division of Responsibility (SRP)

Principle: Each component or microservice should perform only one responsibility (Single Responsibility Principle).

Rationale: Simplifies testing, support, and modification of components.

Scalability and fault tolerance

Principle: The system should be designed to support horizontal scaling and minimize the impact of failures of one component on the entire system.

Justification: Takes into account the requirements for system performance and availability.

Solution stack

Implementation

- Language Python
- API definition OpenAPI
- Connection server for API python gunicorn
- App framework python FastAPI
- Serialization/state format json

Asynchronous interactions (optional)

- Message queue rabbitmq
- Messaging client library celery

Testing tools pytest

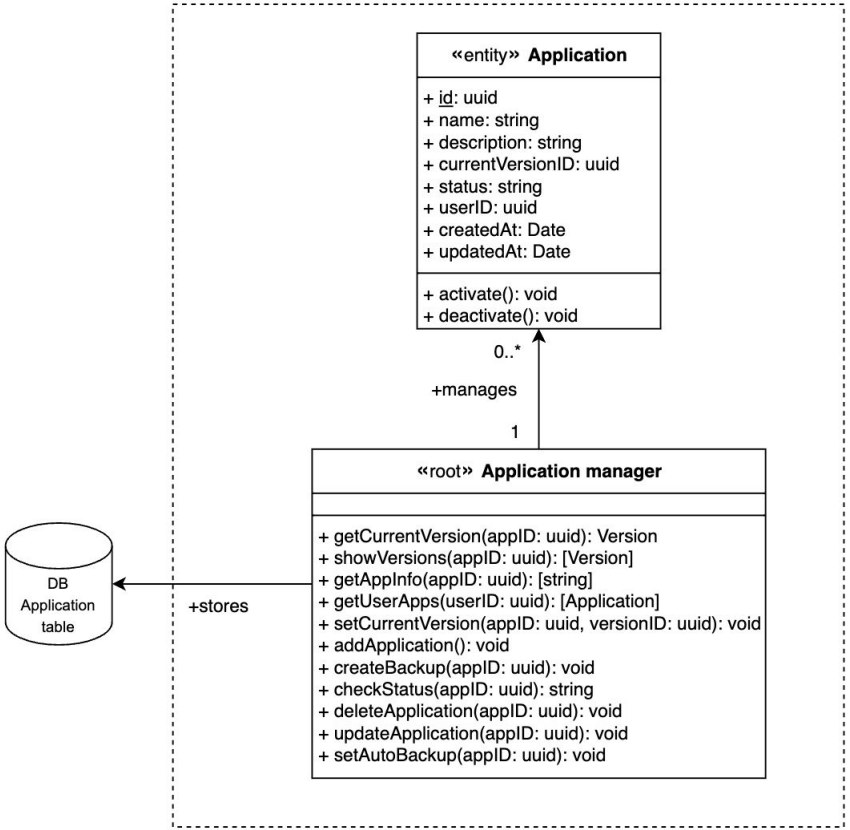
Operations

- App initializer systemd
- Code build makefile
- CI/CD pipeline gitlab
- Delivery method docker
- Logging & monitoring prometheus, loki, grafana

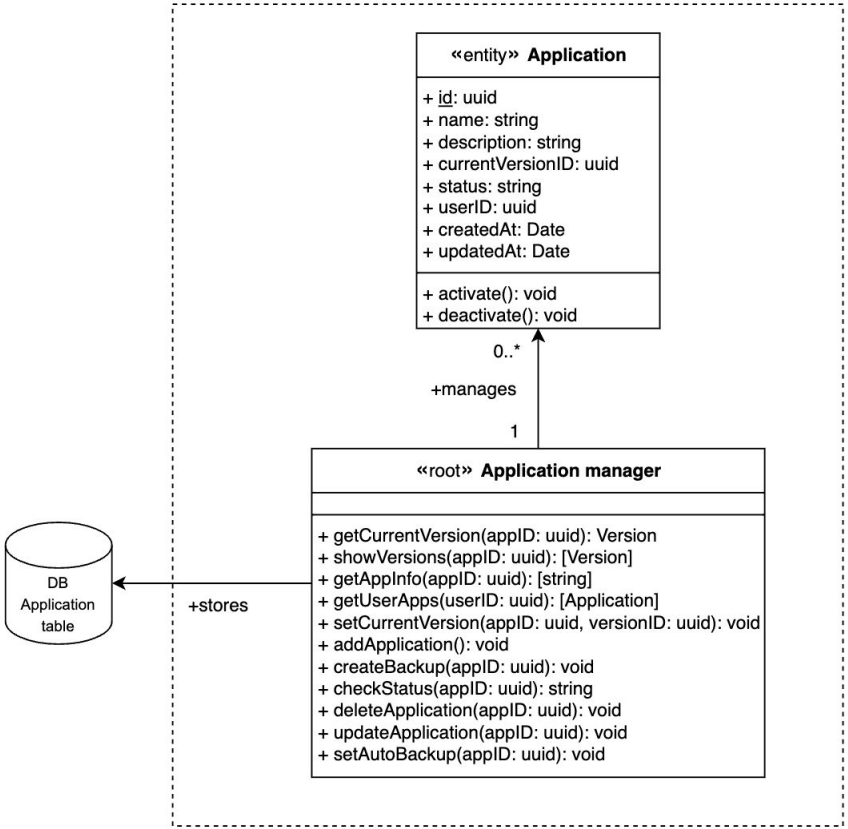
Design case Application Manager

Problem: Scaling difficulties: If a microservice is not able to scale effectively, this can lead to congestion, especially with a large number of requests.

Solution: CQRS (Command Query Responsibility Segregation) using the CQRS pattern will help to separate the logic of data recording



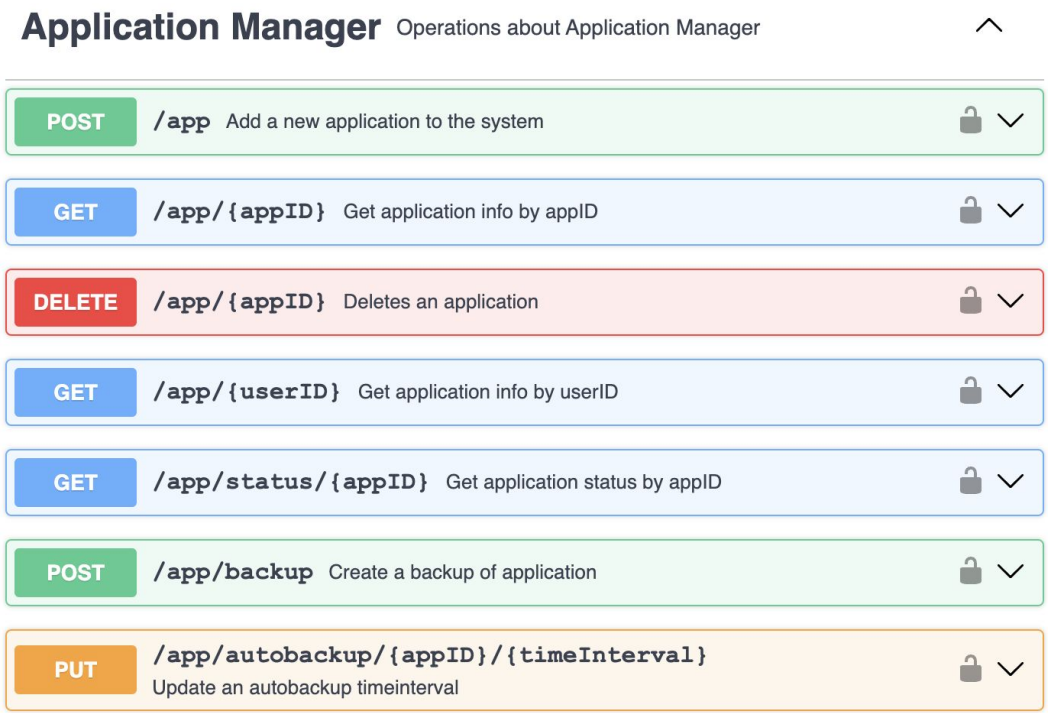
Logical data model for Application Manager



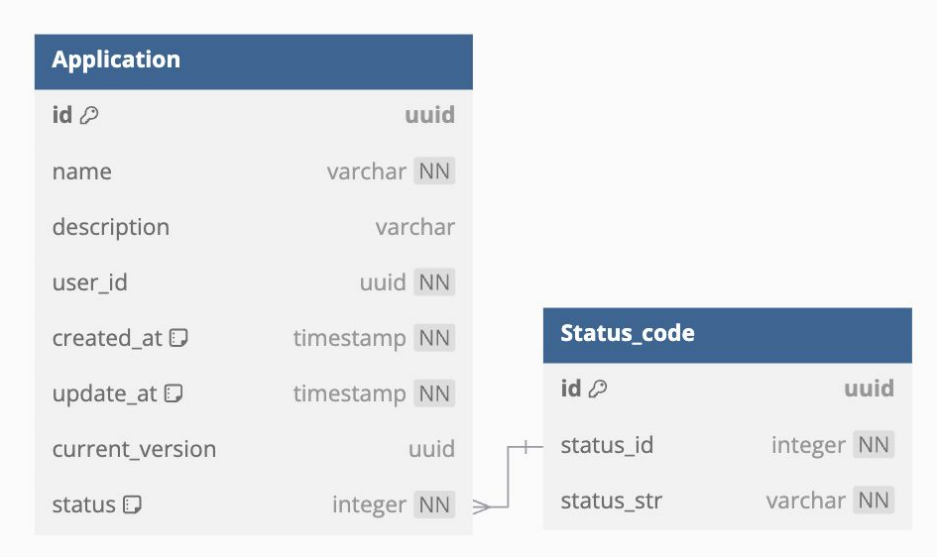
API usage for Application Manager

Use cases:

- Creating a new application
- Creating application backup
- Getting application status



Physical schema for Application Manager




```
CREATE TABLE "Application" (
  "id" uuid PRIMARY KEY,
  "name" varchar UNIQUE NOT NULL,
  "description" varchar,
  "user_id" uuid NOT NULL,
  "created_at" timestamp NOT NULL DEFAULT (now()),
  "update_at" timestamp NOT NULL DEFAULT (now()),
  "current_version" uuid,
  "status" integer NOT NULL DEFAULT 0
);
```

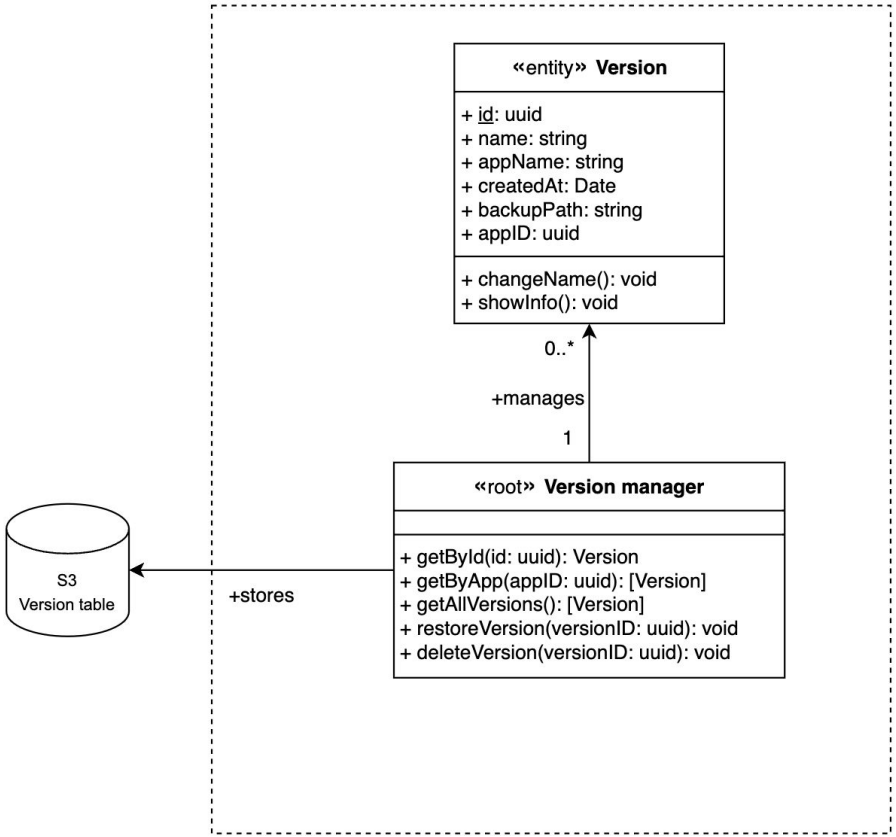
```
CREATE TABLE "Status_code" (
  "id" uuid PRIMARY KEY,
  "status_id" integer NOT NULL,
  "status_str" varchar NOT NULL
);
```

Design case Version Manager

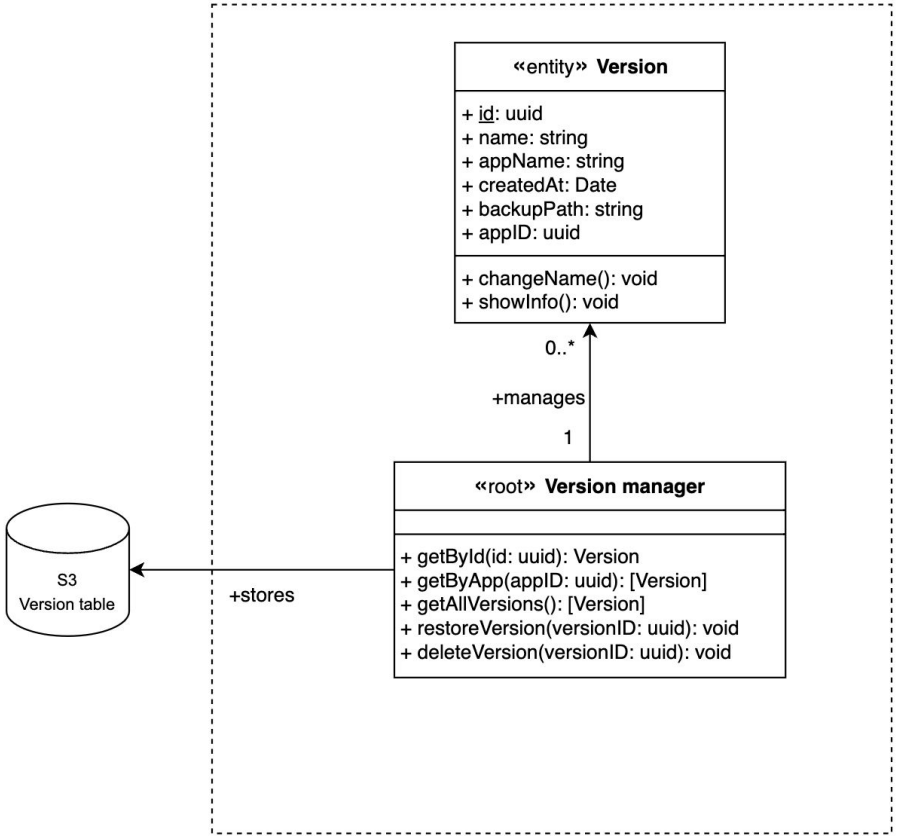
Problem: Storage of data in distributed systems: Integration with cloud storage (for example, S3) is required for reliable storage of backup copies of versions.

Solution: Applied pattern: Adapter

To interact with cloud storage, an adapter interface is used that abstracts the implementation details (for example, S3 API).



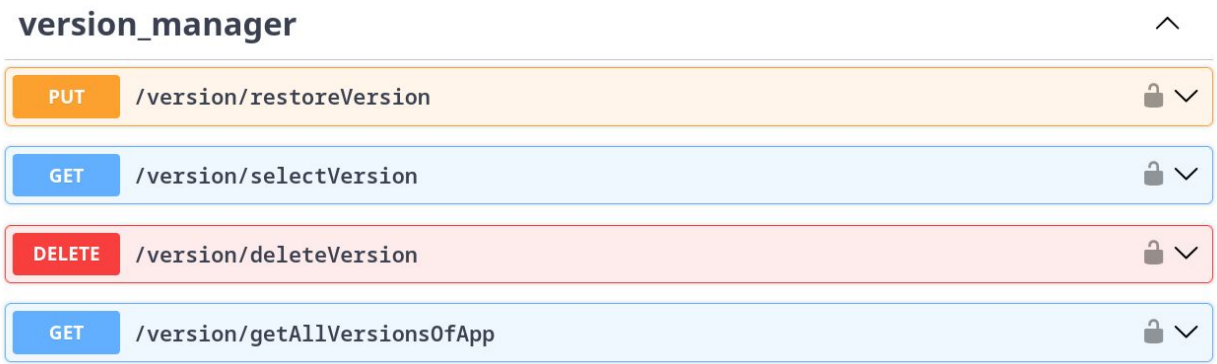
Logical data model for Version Manager



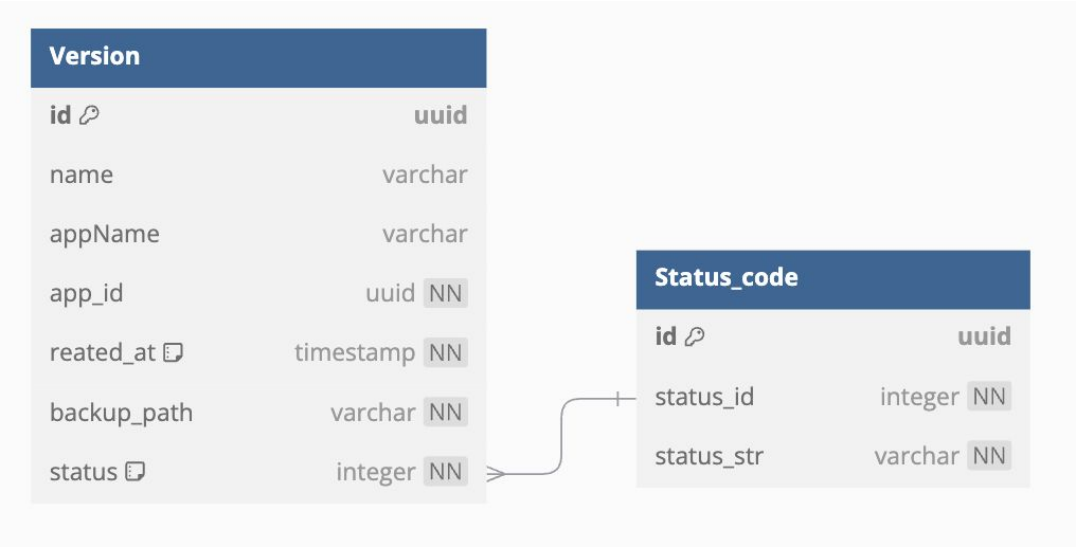
API usage for Version Manager

Use cases:

- Restoring to a version
- Deleting a version



Physical schema for Version Manager



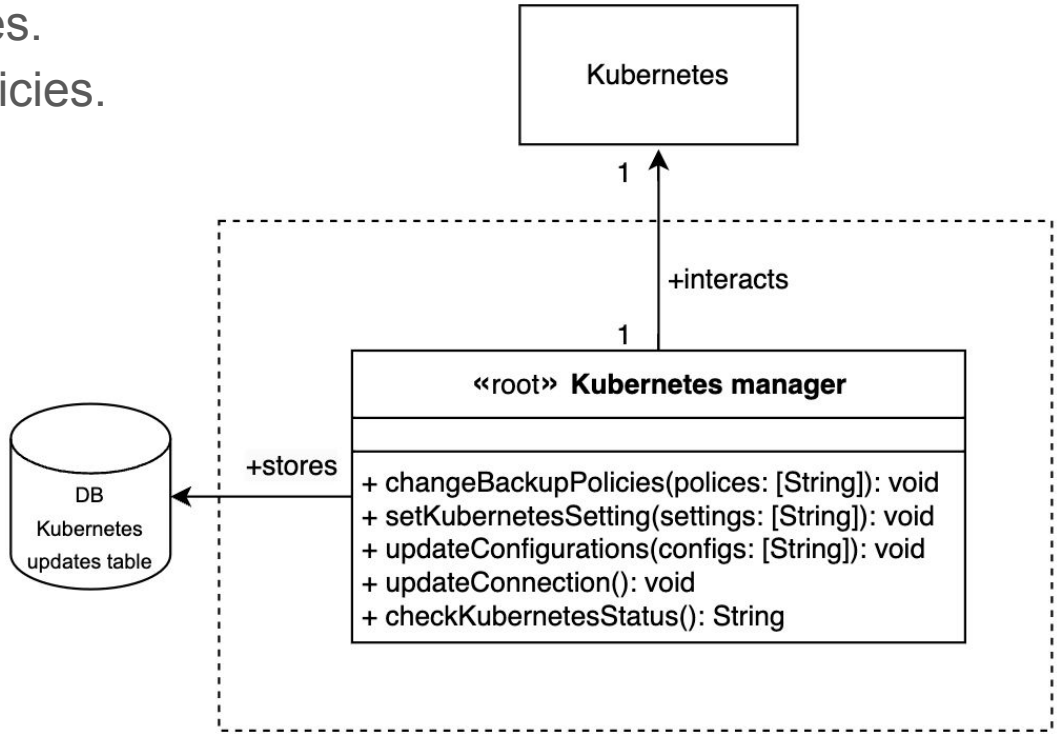

```
CREATE TABLE "Version" (
  "id" uuid PRIMARY KEY,
  "name" varchar,
  "appName" varchar,
  "app_id" uuid NOT NULL,
  "reated_at" timestamp NOT NULL DEFAULT (now()),
  "backup_path" varchar NOT NULL,
  "status" integer NOT NULL DEFAULT 0
);

CREATE TABLE "Status_code" (
  "id" uuid PRIMARY KEY,
  "status_id" integer NOT NULL,
  "status_str" varchar NOT NULL
);
```

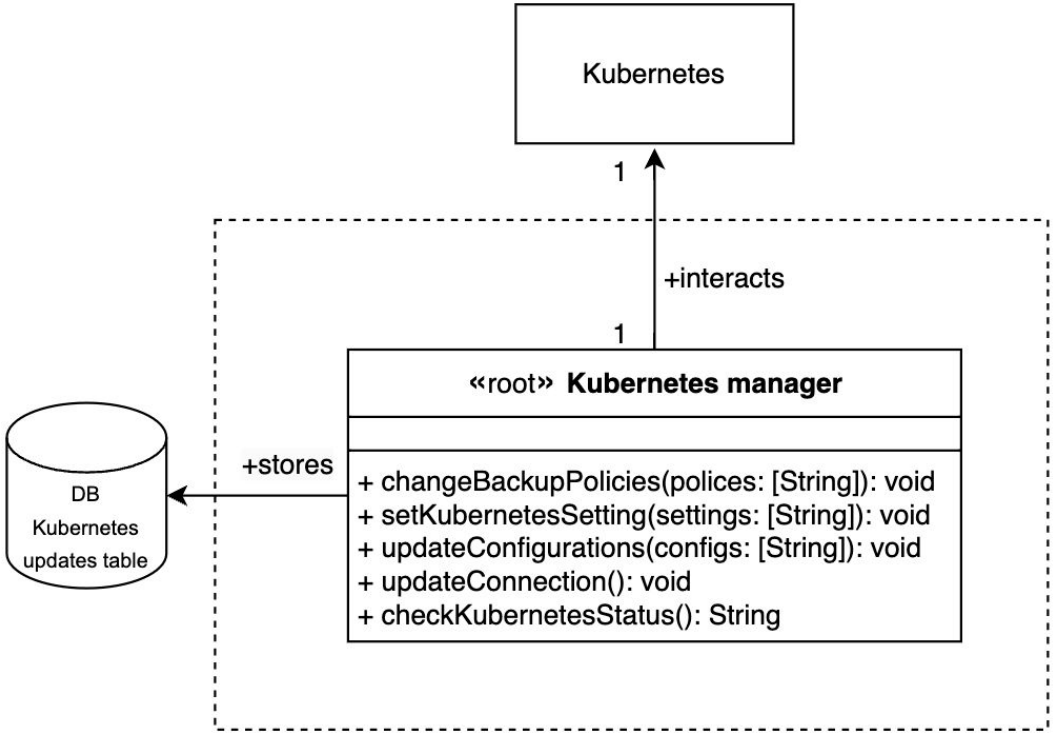
Design case Kubernetes Manager

Problem: Access to kubernetes.
Making changes. Updating policies.

Solution: API Gateway which abstracts the work with k8s



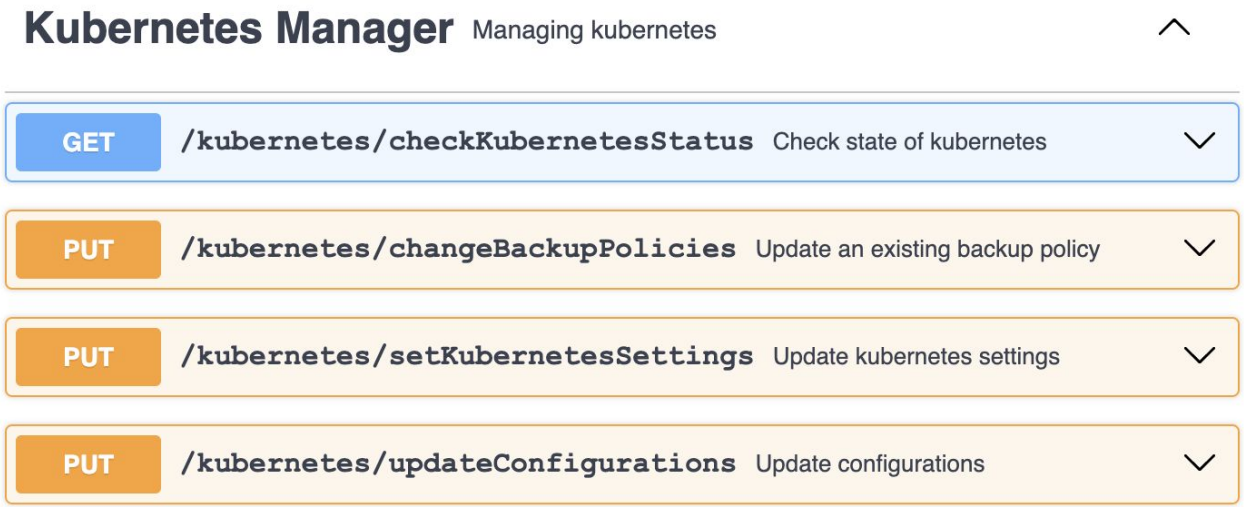
Logical data model Kubernetes Manager



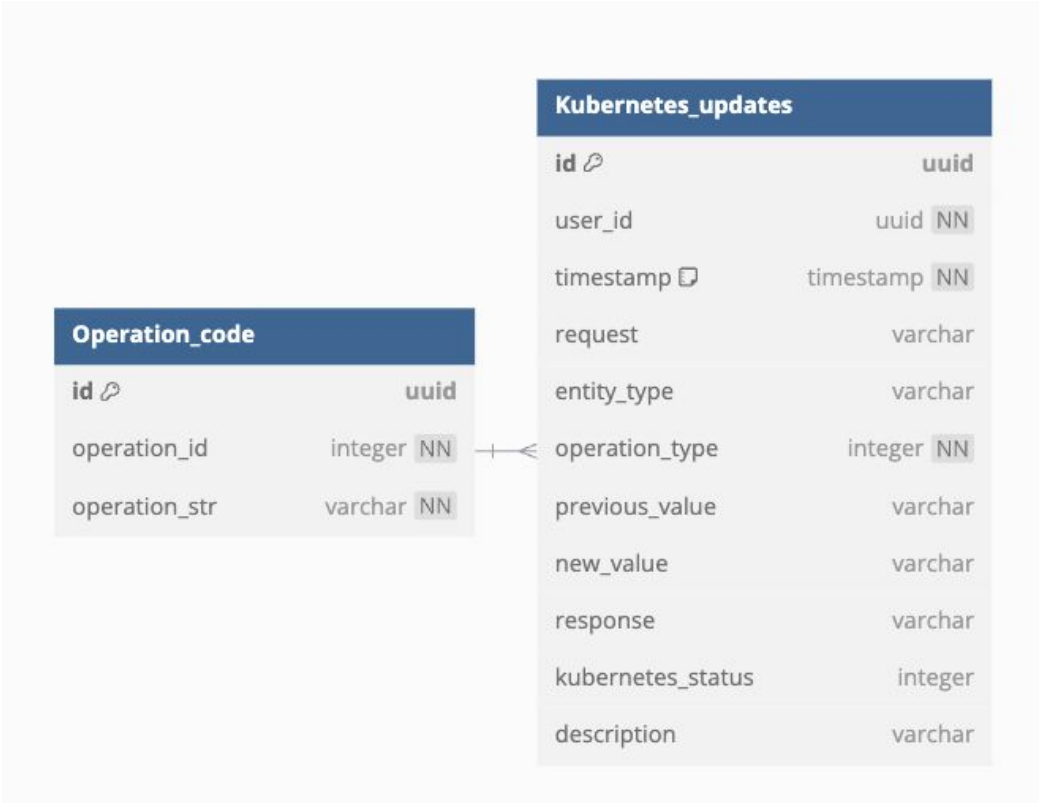
API usage for Kubernetes Manager

Use cases:

- Checking kubernetes status
- Changing backup policies
- Updating kubernetes settings



Physical schema for Kubernetes Manager



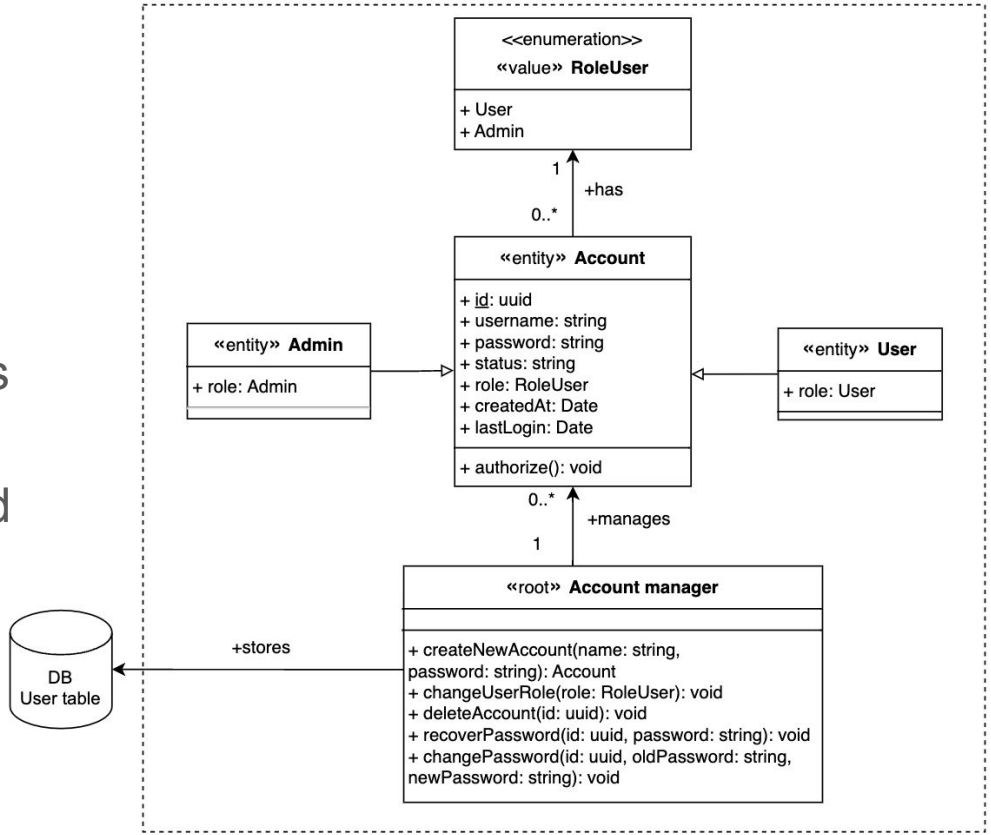

```
CREATE TABLE "Kubernetes_updates" (
  "id" uuid PRIMARY KEY,
  "user_id" uuid NOT NULL,
  "timestamp" timestamp NOT NULL DEFAULT (now()),
  "request" varchar,
  "entity_type" varchar,
  "operation_type" integer NOT NULL,
  "previous_value" varchar,
  "new_value" varchar,
  "response" varchar,
  "kubernetes_status" integer,
  "description" varchar
);

CREATE TABLE "Operation_code" (
  "id" uuid PRIMARY KEY,
  "operation_id" integer NOT NULL,
  "operation_str" varchar NOT NULL
);
```

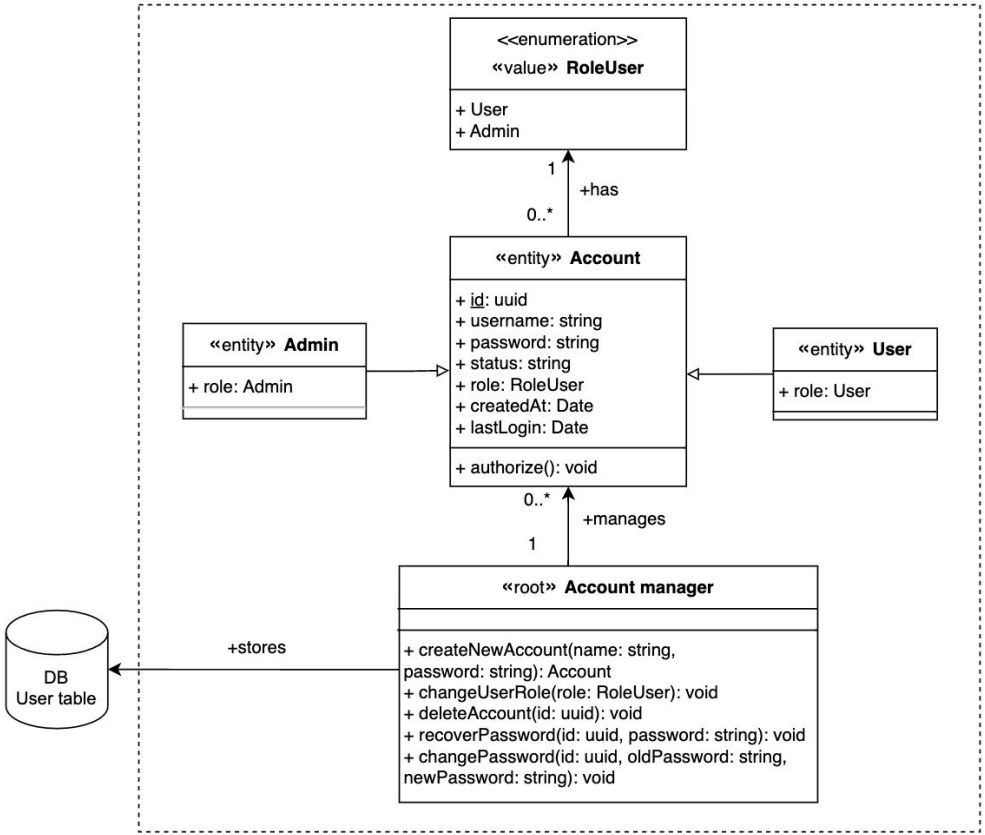
Design case Account Manager

Problem: required to manage user accounts (create, delete, change roles and passwords) based on their roles.

Solution: The "Root Entity" pattern is used, where the Account Manager is the control point for all actions related to accounts.



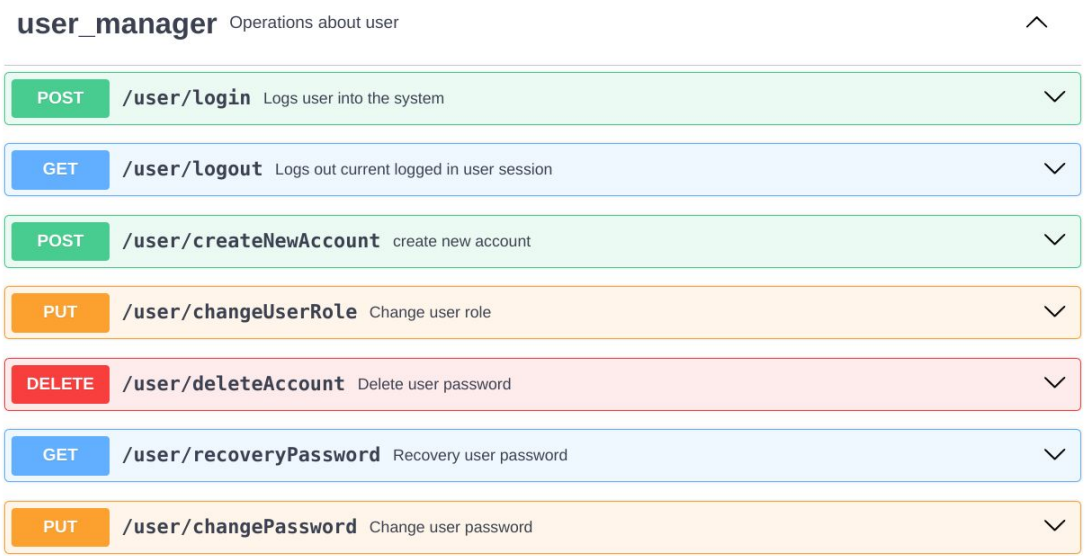
Logical data model for Account Manager



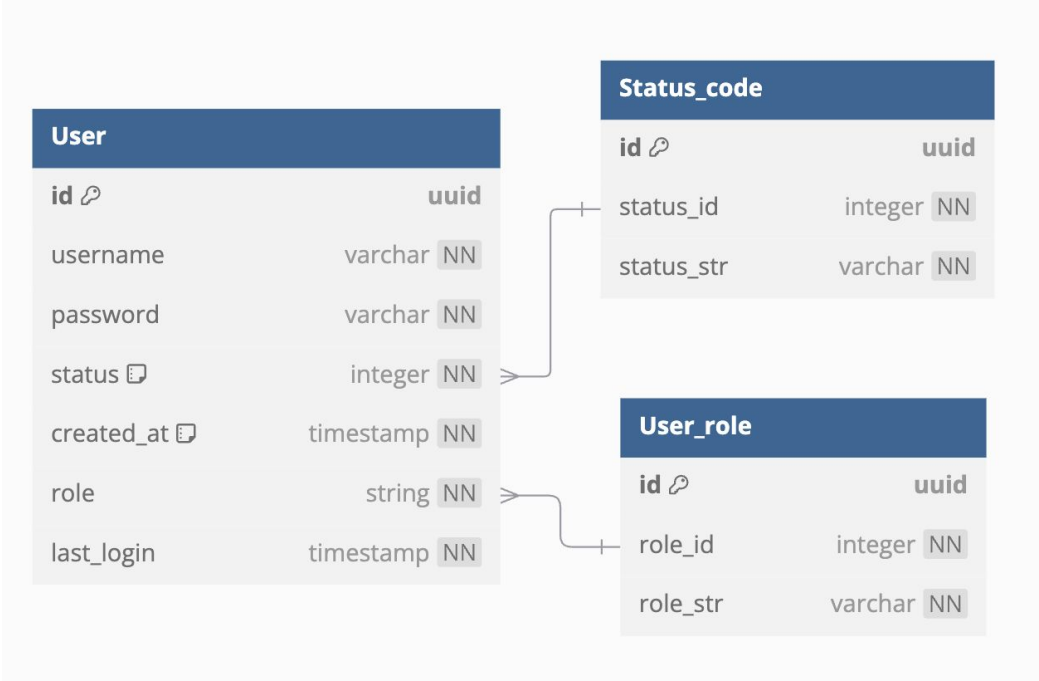
API usage for Account Manager

Use cases:

- User creating an account
- User managing account
- Admin changing role



Physical schema for Account Manager




```
CREATE TABLE "User" (
  "id" uuid PRIMARY KEY,
  "username" varchar UNIQUE NOT NULL,
  "password" varchar NOT NULL,
  "status" integer NOT NULL DEFAULT 0,
  "created_at" timestamp NOT NULL DEFAULT (now()),
  "role" string NOT NULL,
  "last_login" timestamp NOT NULL
);

CREATE TABLE "User_role" (
  "id" uuid PRIMARY KEY,
  "role_id" integer NOT NULL,
  "role_str" varchar NOT NULL
);

CREATE TABLE "Status_code" (
  "id" uuid PRIMARY KEY,
  "status_id" integer NOT NULL,
  "status_str" varchar NOT NULL
);
```

Design complexity

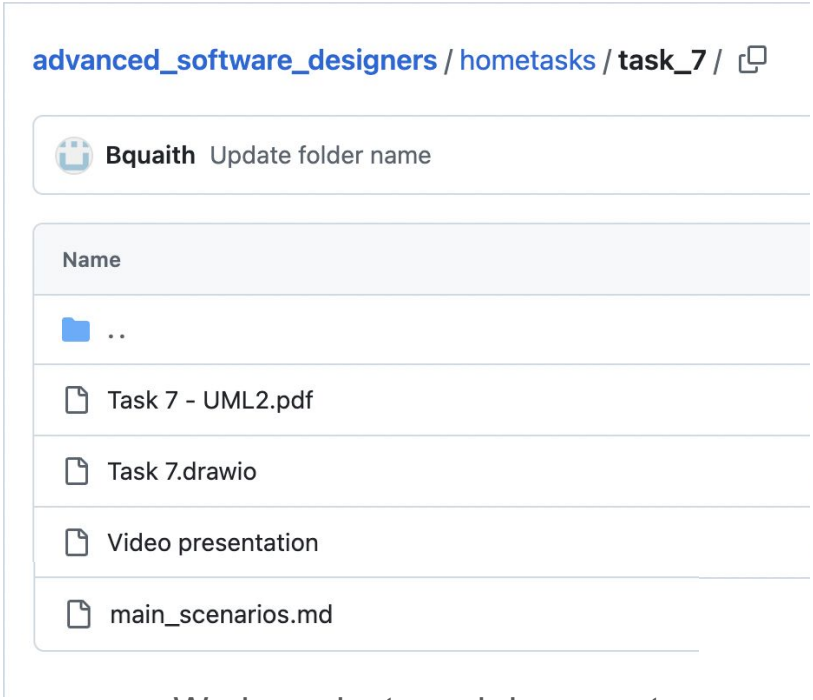
Chidamber-Kemerer suite metrics for Account manager:

Account		**Account manager**		**RoleUser**		**Admin**		**User**	
CBO	4	CBO	1	CBO	1	CBO	1	CBO	1
NOC	0	NOC	0	NOC	2	NOC	0	NOC	0
DIT	1	DIT	0	DIT	2	DIT	2	DIT	2
WMC	1	WMC	5	WMC	0	WMC	0	WMC	0

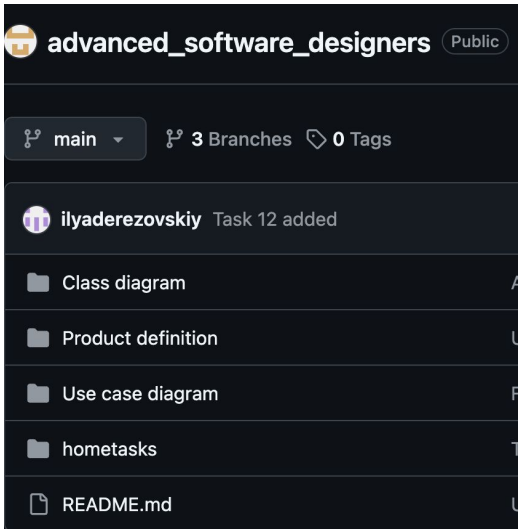
Service dependency metrics:

Application		**Version**		**Account**		**Kubernetes**	
SIY	0.7	SIY	0.4	SIY	2.3	SIY	0.6
AIS	0.3	AIS	0	AIS	1.3	AIS	0.2
ADS	0.4	ADS	0.4	ADS	1	ADS	0.4

Repository structure



Work products and documents



Presentation (slides)

Code (diagrams)

Video presentation

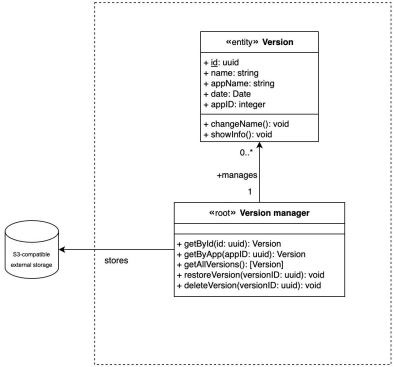
Textual description

Repository structure

Used tools:

1. Repository and Docs storage: GitHub
2. Modeling tool:
   a. draw.io - creating diagrams (data flow diagram, domain model, logical models and others)
   b. miro - Interaction analysis (Event storming)
   c. swagger - creating documentation
   d. dbdiagram.io - creating database diagrams and physical schema
3. Team chat: Telegram

miro usage for creating Event storming cards



draw.io usage for creating domain model

Team and roles

Lukashin Daniil
@bquaithe

Project Manager

Gorelyi Mikhail
@MichaelGorelyi

Backend
Developer

Derezovskiy Ilya
@ilya_derezovskiy

System Architect

Sigal Lev
@Levandou

Frontend
Developer