POLITECNICO DI TORINO

Master's Degree in Engineering and Management

Master's Degree Thesis

Development of REST API to support the Project Management object and process model

Supervisor:
Prof. Paolo Eugenio Demagistris

Candidate:
Simone Ranno

In collaboration with: PM-Lab of PoliTo

Academic Year 2021/2022

Abstract

This thesis has been carried out with the collaboration of the Project Management Laboratory of Politecnico di Torino.

Project Management processes are already well defined, and usually take a semi-standardized path in well-established environments. On the other hand, only in recent times companies began to introduce the use of digital tools in support to the project management process, making this new branch still wild and open to enhancement. Many tools are being developed for the new agile methodologies, but they lack the classic project management document-oriented approach. The purpose of this thesis is to demonstrate that the technology is ready for the next step towards the digitalization of Project Management through the development of an open-source platform to enable the digital project delivery. The software will be composed by three components, the front-end, the back-end and the API, where only the latter will be the one researched in this paper.

Contents

Introduction

1.1 Where the need comes from

The Project Management Laboratory of the Polythecnic University of Turin, that researches cutting-edge processes and methodologies of Project Management, has found that the current project management industry stands behind in terms of digitalization. Given this findings, the PM Lab has issued a research in order to demonstrate that the technology is mature enough to guide the Project Management towards the next step.

1.2 What is needed

For the purpose of creating a working platform for supporting the Project Management objects and processes, it has been individuated the need for a three component project:

Front-end the part visible to the user.

Back-end the component responsible for delivering the front-end's User Interface to the user, managing authentication and authorization.

API that serves both front-end and back-end, delivering a suite of interface-less resources and services, including authorization and authentication.

1.3 What this thesis is focusing on

The focus of this thesis will be demonstrating the technology readiness for the development of the API required by the project.

Definitions

2.1 What is an API

API stands for **A**pplication **P**rogramming **I**nterface
But what does this mean? API means that the system allows a way to access some informatic service, through a well-defined suite of protocols and methods, which require to communicate with a predefined set of rules.
The purpose of an API is to allow multiple different systems to communicate with the desired resources, without redefining an ad-hoc system for every informatic system. Examples of different systems are websites, mobile applications, automated services and more.

2.2 What is an end-point

In the API environment, it is defined as an endpoint a specific string that the server recognizes without ambiguity and that corresponds to a specific action or resource. Example of an endpoint: example.com/documents is an endpoint If the endpoint is interrogated with GET method, it will return a collection of documents, while if interrogated with POST it could allow the insertion of a new document

2.3 The project management object and process model

Project management is defined by documents and processes, which produce data in a structured or unstructured way. This necessity for documents has always been

satisfied by paper and ink, but we can upgrade to a digital document management, allowing a fine-grained control over authentication and authorization.

2.4 How a API satisfies the Project Management needs

Every project defines its own processes and documents and needs a way to store them, each associated with its own authorizations for the users. A API can satisfy those needs, providing a standardized way of accessing resources and limiting them according to custom rules defined by the project owner. In this way other platforms/services/websites/applications can access and edit in the same way the resources.

2.5 The classic approach versus the modern approach

Ten years ago, the best choices for web development were Apache web server, that executed PHP code, Apache Tomcat for Java and ASP.NET for C# and Visual Basic. Nowadays new tools are readily available, offering a much simple and fast deployment with good performances. To name a few, Python has Django, Flask and FastApi; Kotlin has KTor; Golang has Gin; Rust has Rocket Framework.

2.6 What is a Framework

In computer science, it is defined as a framework a suite of code that satisfies a need by calling some user-written code. The project will make use of frameworks designed for the delivery of web services.

2.7 Data interchange formats

In order to transfer meaningful data across the web, many interchange formats have been developed. For the purpose of this thesis, only two are being considered.
JSON: (JavaScript Object Notation) one of the most popular formats, has found wide adoption and many tools are defined around it, making it compatible with

nearly every system.
YAML: (Yet Another Markup Language) another popular format, it has a structure similar to JSON but is more human-readable, making it suitable for direct user input.

Case study

3.1 Requirements

The platform was required to have the following features and constraints:

3.1.1 Expose a REST API to add, retrieve, edit and delete documents

This requirement states that the way to interact with the documents must be a REST API over Http protocol and must support the Http methods (GET, POST, PUT, PATCH, DELETE). REST is a type of architecture for APIs that defines the methods and constraints for the communications. There exist other types of API (SOAP, RPC, GraphQL) but REST is the most popular and one of the easiest to learn, making it the best candidate for an easy-accessible API.

3.1.2 Enable row level permission to documents

The API must authenticate user and limit their actions based on the permissions the user has on the object resource. Row-level states that authorization policies are enforced to the level of the single record.

3.1.3 Allow the reuse of existing components to ease the implementation of already defined structures (Microsoft Project)

In order to not reinvent the wheel, the software should use already popular tools, since many users already have familiarity with wide-known softwares and programming libraries. In particular, the API has to allow the transfer of Microsoft Project

files, which already incorporate structured data about the project scheduling and resources.

3.1.4 Use python as programming language

Being the most popular programming language across university environments, python has been dictated as requirement for this software, in order to be easily readable and accessible. Although there exist programming languages that carry incredibly better performance (Java, GoLang, C++, .Net), the use of Python allows an easier development and requires less programming knowledge, making the project more accessible to people who do not possess high-level programming skills.

3.2 Design

After extensive research about the documents used in project management, it came the conclusion that there is no exact industry standard and every company defines its own rules. This could be acceptable in non digital environments, but informatic systems work with exact schemas and data so approximate approaches are not viable for computer-based solutions. Beside that, every project is different from the others, in size and complexity, so hard-coding a one-size-fits-all solution may not result in the most versatile approach. It has been decided to implement a platform that allows the user to define its document structures, constraints and authorizations. Using this paradigm, the software can cover more project needs than having a rigid structure.

3.2.1 Authentication

Authentication is carried out through the use of an internet standard called JWT Tokens (JSON Web Token). JWT Tokens are time-dependent tokens carrying a payload. They provide basic safety for the purpose of this thesis, have a solid foundation since it is based on cryptographic encryption but it is not exempt from flaws.

3.2.2 The project structure

The project has been identified as the top-level container for all the documents. Users with the appropriate authorization on the platform can create projects.

3.2.3 The document structure

Documents can be inserted by authorized users and are composed by the validation schema and the document content. Each time the document is edited, the system keeps track of the edit user, content and time. In order to ensure compatibility with other systems, the JsonPatch standard shapes the format of the edits.

3.2.4 Processes

Processes are defined as activities having documents as inputs and outputs. The system will prevent the insertion of the content of a document if there exists any defined process where the document to be inserted is listed in the outputs and there is at least one document listed as input that has no content.

3.2.5 Permissions

Permissions are defined at system, project and document level. Each call to the API requires a certain permission, that can be granted by users who have the "edit permissions" authorization for that resource.

3.2.6 Computed fields

In many cases documents need to incorporate a summary of other documents. For this event, the system allows to define computed fields. Computed fields are defined as a special string, whose value is ensured by the JsonPath standard. JsonPath allows to retrieve a subset of data through filtering, making it possible to summarize other documents.

3.2.7 Microsoft Project Files

Microsoft Project is one of the most popular software for scheduling, resource and cost allocation, and its inclusion in the software allows to interoperate with its files, making easier the communication with the API for users that already have familiarity with it.

3.2.8 MS Project Computed Fields

Like the document computed fields, users can define computed fields that summarize content from Microsoft Project files.

3.3 Technologies

3.3.1 Python

Python is a general-purpose programming language, that is wide utilized due to its easy syntax and fast implementation. Its structure is more similar to a pseudocode, famous for the use of indentation instead of brackets. However, the use of Python has its drawbacks, due to its notorious slowness in running time, compared to many other languages.

3.3.2 FastApi

One of the "Big Three" in the python web development environment, has seen its popularity rise thanks to the fast prototyping and fast response times, from both of which claims its name. Specialized in API, FastApi can cover also other cases such as HTML responses and GraphQL API.

3.3.3 SQLModel, SQLAlchemy and Pydantic

Developed by the creator of FastApi, SQLModel is both an ORM (Object-Relational Mapping) and an interface for database interaction. It helps with the connection to a database and it transforms the results from the database into Python usable data and structures. It is based on both SQLAlchemy and Pydantic and combines their features. SQLAlchemy provides the relational mapping while Pydantic offers simple ways to return data, making it easy to return JSON in FastApi calls.

3.3.4 JSON

JavaScript Object Notation. Based on the JavaScript programming language, has seen widespread use in every environment. Many standards, languages and tools have been developed for JSON and in this project three of them are being used:

JsonSchema

Standard for schema validation of JSON documents, allows to define fine-grained constraints for fields and types.

JsonPatch

Format based on JSON which defines edits to a JSON document, allowing to record changes without having to store every variation.

JsonPath

Query language created for JSON, allows to define queries for a particular field or subset, with the option to apply a filtering logic.

3.3.5 MPXJ and JPype

MPXJ is a Java-based library that can work with Microsoft Project files. Being Java-based means that it needs a special porting and it has to run with the JPype library, that bridges Java code to make it interoperable with Python code.

Implementation

4.1 Python code

4.1.1 default module

main.py

This python file is the first file called on startup and sets up the FastApi app instance, sets up the database tables and defines the basic endpoints for the user authentication.

Listing 4.1: App instance

```
app = FastAPI()
```

Listing 4.2: Set up the database on startup

```
@app.on_event("startup")
def on_startup():
    """
    Called on startup, creates the database tables if they do not exist
    """
    create_db_and_tables()
```

Listing 4.3: Login for access token

```
@app.post("/token", response_model=Token)
async def login_for_access_token(form_data: OAuth2PasswordRequestForm = Depends(),
                                 session  : Session = Depends(get_session)):
    """
    Reads data from the login form and returns a token if the user is valid

    :param form_data: data sent by the client
    :param session: database session from dependencies
    :return: token
    """
    user = crud.get_user(session, form_data.username)
    if user is None \
```

```
            or \
            not verify_password(form_data.password, user.password):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.user_name}, expires_delta=access_token_expires
    )
    return {"access_token": access_token, "token_type": "bearer"}
```

Listing 4.4: Register user

```
@app.post("/register", response_model=UserBase)
def register(user: User, session: Session = Depends(get_session)):
    """
    Registers a new user

    :param user: user to add to the database
    :param session: database session from dependencies
    :return: user
    """
    if crud.get_user(session, user.user_name) is not None:
        raise HTTPException(status_code=status.HTTP_409_CONFLICT, detail="User could not
     be registered")
    user.password = get_password_hash(user.password)
    print(crud.get_users_count(session))
    if crud.get_users_count(session) == 0:
        user.system_permissions = [SystemPermission(permission=permission) for
    permission in SysPermissions]
    session.add(user)
    session.commit()
    session.refresh(user)
    if not user:
        raise HTTPException(status_code=409, detail="User could not be registered")
    return user
```

app_config.py

Contains the unique secret key used in the authentication process.

Listing 4.5: Secret key

```
SECRET_KEY = "991c37e0b2867aa8f7ab5028ee26fd185c2d1a92ab178c3e15330297c8931e2f"
```

The secret key should be private and changed from the one provided. A new key can be generated using the following command: openssl rand -hex 32

auth.py

Defines the methods necessary to the authorization of the user. The following are the permissions that can be checked for system, project and documents.

Listing 4.6: Permissions

```python
class Permissions(str, enum.Enum):
    """
    Enum for permissions
    """
    create = "create"
    view   = "view"
    edit   = "edit"
    delete = "delete"
    edit_permissions = "edit_permissions"
```

The permissions check is called from the dependencies trough the methods has_system_permission, has_project_permission and has_document_permission

Listing 4.7: Check if user has permission on document

```python
def has_document_permission(session   : Session,
                            user      : User,
                            project   : Project,
                            document  : Document,
                            permission: Permissions):
    """
    Checks if a user has a permission on a document

    :param session: session to use
    :param user: user to check
    :param project: project of document
    :param document: document to check
    :param permission: permission to check
    :return: True if user has permission, False otherwise
    """
    if project is None:
        return False
    if user.user_name == project.owner_name:
        return True
    if document is None:
        perm = PermissionUtils(session, user.user_name, project.project_name)
    else:
        if user.user_name == document.author_name:
            return True
        perm = PermissionUtils(session, user.user_name, document.project_name, document.
    document_name)

    if permission not in document_map.keys():
        return False
    p = document_map[permission]
    return (
        perm.has_sys_perm(p[0])
        or perm.has_proj_perm(p[1])
        or perm.has_doc_perm(p[2]))
```

dependencies.py

Defines the methods used in the FastApi's dependency injection system. One particular use of this dependency injection system is the check of authentication and authorization before the method execution, removing the need of checking inside it.

Listing 4.8: Retrieve current authenticated user

```
async def get_current_active_user(token  : str     = Depends(oauth2_scheme),
                                  session: Session = Depends(get_session)):
    """
    Gets the current user from the token

    :param token: token
    :param session: database session from dependencies
    :return: current user
    """

    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
        token_data = TokenData(username=username)
    except JWTError:
        raise credentials_exception
    user = crud.get_user(session, token_data.username)
    if user is None:
        raise credentials_exception
    return user
```

Retrieve project, document or msprojects by name Thanks to dependency injection the method gets the project name from the path parameters

Listing 4.9: Retrieve project

```
def get_project(project_name: str | None = None,
                session     : Session    = Depends(get_session)):
    """
    Gets the project named project_name from the database
    If project_name is None, returns None
    If project is not found, raises 404 exception

    :param project_name: project name, from path params
    :param session: database session from dependencies
    :return: project
    """
    if project_name is None:
        return None
    db_project = crud.get_project_by_name(session, project_name)
    if db_project is None:
```

```
16          raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="Project not
        found")
17      return db_project
```

In the dependencies there is also the method for checking if the user has certain permissions. With the help of a callable class, if the user does not have the required permission the method will raise an exception and return the 401 unauthorized http status

Listing 4.10: Check Project Permission

```
1  class CheckProjectPermission:
2      """
3      Callable class that checks if user has permission on project
4      """
5      def __init__(self, permission: Permissions):
6          self.permission = permission
7
8      def __call__(self,
9                   session   : Session        = Depends(get_session),
10                  user      : User           = Depends(get_current_active_user),
11                  db_project: Project | None = Depends(get_project)):
12
13         if not has_project_permission(session, user, db_project, self.permission):
14             raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="
       Unauthorized")
15
16
17 def require_project_permission(permission: Permissions):
18     """
19     Checks if user has permission on project, raises 401 if not authorized
20
21     :param permission: permission to check
22     """
23     return CheckProjectPermission(permission)
```

The dependencies also help to retrieve the body content from different content-types.

Listing 4.11: Get request body

```
1  async def get_request_body(request: Request):
2      """
3      Gets request body from request and returns it as dict, parsing both json and yaml
4
5      :param request: request from body
6      :return: dict with request body
7      """
8      if "content-type" not in request.headers:
9          raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Missing
       Content-Type in header")
10     content_type = request.headers['content-type']
11
12     match content_type:
13         case "application/json":
```

```
            try:
                request_body = await request.json()
            except json.decoder.JSONDecodeError as err:
                raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail=str(
    err))
        case "application/x-yaml":
            try:
                request_body = yaml.safe_load(await request.body())
            except Exception as err:
                raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail=str(
    err))
        case _:
            raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST)
    return request_body
```

4.1.2 database module

crud.py

Defines the queries required for retrieval of records identified by multiple primary keys. The method uses the database session in order to retrieve the requested document.

Listing 4.12: Get document of project

```
def get_document_of_project(session: Session, project_name: str, document_name: str):
    """
    Get document of project

    :param session: session to use
    :param project_name: project name
    :param document_name: document name
    :return: document if found, None otherwise
    """
    return session.exec(select(Document)
                        .where(Document.project_name  == project_name)
                        .where(Document.document_name == document_name)
                        ).first()
```

4.1.3 datatypes module

models.py

Defines the SQLModel models, composed by fields and relationships. Example of a Model:

Listing 4.13: Document model

```
class Document(SQLModel, table=True):
```

```
    # field name   | type            | options
    project_name   : str             = Field(default=None, primary_key=True, foreign_key=Project.Fields.project_name)
    document_name  : str             = Field(default=None, primary_key=True)
    author_name    : str             = Field(default=None,                   foreign_key=User.Fields.user_name)
    jsonschema     : Dict            = Field(default={}, sa_column=Column(JSON))
    first          : Dict            = Field(default={}, sa_column=Column(JSON))
    last           : Dict            = Field(default={}, sa_column=Column(JSON))
    schema_add_date: datetime        = Field(default_factory=datetime.utcnow)
    creation_date  : datetime | None = Field(default=None)
    updated_date   : datetime | None = Field(default=None)

    # relationship name | type                          | options
    patches             : List["Patch"]                 =
        Relationship(back_populates="document")
    project             : Project                       =
        Relationship(back_populates="documents")
    permissions         : List["DocumentPermission"]    =
        Relationship(back_populates="document", sa_relationship_kwargs={ "cascade": "all, delete, delete-orphan"})

    documents_processes : List["DocumentProcess"]       =
        Relationship(back_populates="document",
        sa_relationship_kwargs={ "cascade": "all, delete, delete-orphan"})

    computed_fields     : List["ComputedField"]         =
        Relationship(
        sa_relationship_kwargs={"primaryjoin": 'Document.document_name==ComputedField.field_document_name', "lazy": "joined", "cascade": "all, delete, delete-orphan"})

    computed_fields_reference: List["ComputedField"]    =
        Relationship(back_populates="reference_document",
        sa_relationship_kwargs={"primaryjoin": 'Document.document_name==ComputedField.reference_document_name', "lazy": "joined"})

    ms_computed_fields : List["MSProjectComputedField"] =
        Relationship(
        sa_relationship_kwargs={"primaryjoin": 'Document.document_name==MSProjectComputedField.field_document_name', "lazy": "joined", "cascade": "all, delete, delete-orphan"})
```

schemas.py

Defines pydantic models for the swagger documentation.

Listing 4.14: Schema models

```
"""
Module for defining schemas for Swagger request body
"""

from typing import List, Dict
from pydantic import BaseModel, Extra
```

```python


class ProcessSchema(BaseModel):
    inputs : List[str]
    outputs: List[str]

    class Config:
        extra = Extra.forbid


class PermissionSchema(BaseModel):
    documents: Dict

    class Config:
        extra = Extra.forbid


class ProjectCreateSchema(BaseModel):
    project_name: str
    documents   : Dict[str, Dict]
    processes   : List[ProcessSchema]
    permissions : Dict[str, PermissionSchema]

    class Config:
        extra = Extra.forbid
```

4.1.4 routers module

projects.py

Defines the endpoints for the operations on projects, which include creation, retrieval, edit of permissions and deletion.

Listing 4.15: Permissions

```python
@router.get("/", response_model=List[str])
def get_projects(session: Session = Depends(get_session),
                 user   : User    = Depends(get_current_active_user)):
    """
    Returns a list of all projects that the user has access to

    :param session: session from dependencies
    :param user: current user from dependencies
    :return: list of project names
    """

    return [project.project_name for project in crud.get_projects(session)
            if has_project_permission(session, user, project, Permissions.view)]
```

documents.py

Defines the endpoints for the operations on documents, which include creation, insertion and edit content, retrieval, edit of permissions.

Listing 4.16: Set document content

```python
@router.put("/{document_name}",
            response_model=DocumentReturn,
            dependencies=[Depends(require_document_permission(Permissions.edit)),
                          Depends(check_document_process),
                          Depends(validate_document)])
async def put_document_to_project(document_body: Dict     = Depends(get_request_body),
                                  user         : User     = Depends(get_current_active_user),
                                  db_project   : Project  = Depends(get_project),
                                  db_doc       : Document = Depends(get_document),
                                  session      : Session  = Depends(get_session)):
    """
    Add or update content of document
    Requires the authenticated user to have the permission to edit the document

    :param document_body: document content to add or update
    :param user: current user from dependencies
    :param db_project: project of path from dependencies
    :param db_doc: document of path from dependencies
    :param session: session from dependencies
    :return: the document
    """

    last = db_doc.last
    time = datetime.utcnow()
    db_doc.last = document_body
    db_doc.updated_date = time

    if not bool(db_doc.first):
        db_doc.first = document_body
        db_doc.creation_date = time
        db_doc.author_name = user.user_name
    else:
        patch = json.loads(jsonpatch.JsonPatch.from_diff(last, db_doc.last).to_string())
        diff = Patch(project_name=db_project.project_name, patch=patch, user_name=user.user_name)
        db_doc.patches.append(diff)

    for computed_field in db_doc.computed_fields_reference:
        jsonpath_expr = jsonpath_ng.ext.parse(computed_field.jsonpath)
        computed_field.field_value = list(map(lambda a: a.value, jsonpath_expr.find(db_doc.last)))

    session.add(db_doc)
    session.commit()
    session.refresh(db_doc)

    db_doc.update_forward_refs()
```

```
46
47      return db_doc
```

msprojects.py

Defines the endpoints for the operations on Microsoft Project files, which include the upload, retrieval and deletion.

Listing 4.17: Get microsoft project file

```
1 @router.get("/{ms_project_name}",
2             dependencies=[Depends(require_project_permission(Permissions.view))])
3 async def get_ms_file_of_project(db_ms_project: MSProject = Depends(get_ms_project)):
4     """
5     Get ms file of a project
6     :param db_ms_project: ms project from dependencies
7     :return: ms project if found, 404 otherwise
8     """
9     return db_ms_project
```

4.1.5 security module

utils.py

Defines the methods for hashing and verifying passwords

Listing 4.18: Password hashing and verifying methods

```
1  ALGORITHM = "HS256"
2  ACCESS_TOKEN_EXPIRE_MINUTES = 3000
3  pwd_context = CryptContext(schemes=["bcrypt"], deprecated="auto")
4
5
6  def verify_password(plain_password, hashed_password):
7      """
8      Verifies password against a hash
9
10     :param plain_password: password to verify
11     :param hashed_password: hash to verify against
12     :return: True if password matches, False otherwise
13     """
14     return pwd_context.verify(plain_password, hashed_password)
15
16
17 def get_password_hash(password):
18     """
19     Returns a hash of the password
20
21     :param password: password to hash
22     :return: hashed password
23     """
24     return pwd_context.hash(password)
```

schemas.py

Defines the pydantic schemas for the JWT token

Listing 4.19: Token Schemas

```python
"""
Token schemas
"""

from pydantic import BaseModel


class Token(BaseModel):
    access_token: str
    token_type  : str


class TokenData(BaseModel):
    username: str | None = None
```

4.1.6 utils module

Contains the common utility functions as the retrieval of the current timestamp.

Listing 4.20: Utility functions

```python
"""
Module for utility functions
"""

import time


def current_milli_time():
    """
    Returns the current time in milliseconds

    :return:
    """
    return round(time.time() * 1000)
```

4.2 Schemas

Schemas are fundamental for the correct communication with the API. Every endpoint has defined input and output schemas, that the client must respect. This API accepts JSON and YAML schemas to define projects and documents. Documents fields are user-defined and follow constraints based on a jsonschema the user uploads

4.2.1 Project Schema

Listing 4.21: Project creation schema

```
project_name: example_project
documents:
  ...
processes:
  ...
permissions:
  ...
```

Created project must provide a unique name. The other fields (documents, processes and permissions) are optional. The "documents" field specifies the documents that the project will contain. Documents can be added also after the project creation and the schema for insertion is the same. As the name suggests, "Processes" defines the processes for the project. Each process has two fields: inputs and outputs.

Listing 4.22: Processes field in Project creation schema

```
processes:
  develop_work_breakdown_structure:
    inputs: [ project_charter ]
    outputs: [ work_breakdown_structure ]
```

The "Permissions" field defines the authorization on the project and documents for the users.

Listing 4.23: Permissions field in Project creation schema

```
permissions:
  doge:
    documents:
      project_charter: [ view, edit, delete ]
      work_breakdown_structure: [ view, edit, delete ]
```

4.2.2 Document schema

Documents are created specifying name and schema.

Listing 4.24: Document creation schema

```
project_charter:
  jsonschema:
    properties:
      overview: { type: string }
      impact: { type: string }
      organization: { type: string }
    additionalProperties: false
  computed_fields:
    scope:
      reference_document: work_breakdown_structure
      jsonpath: $.elements[?(@.level == 1)].name
```

```
12  ms_computed_fields:
13    scope:
14      ms_project_name: example
15      field_from: tasks
16      jsonpath: $[?(@.level < 3)].name
```

As the name explains, “jsonschema” defines the schema which the document will be validated against. “computed_fields” is a list of fields that refer to other documents of the project. To provide a summary and not the entire document, the field “jsonpath” defines which parts of the other document to include. “ms_computed_fields” works like “computed_fields” only that the referred document is a Microsoft Project document. “field_from” specifies from which of the three fields (tasks, resources, proj_info) the document is referring to.

Version Control, Deployment and Documentation

5.1 Version control repository

In order to track changes, it has been selected the GitHub repository platform, that offers reliable tools for software versioning.
The source code is publicy available at
github.com/pm-lab-polito/EnvForDigitalProjectDelivery/tree/main/project-management-api

5.2 Deployment

Docker has been selected for the deployment. The Dockerfile in the code files takes a python 3.10 image, installs the required packages, installs java and runs uvicorn.

```
FROM python:3.10
RUN apt-get update && \
    DEBIAN_FRONTEND=noninteractive \
    apt-get -y install default-jre-headless && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*
RUN pip install --upgrade pip
WORKDIR /code
COPY ./app/requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
COPY ./app /code/app
ENV PYTHONPATH /code/app
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "80"]
```

In order to run the API, three steps are required: build the image, create a volume and finally run the image in a container specifying the volume for the ”db” folder.

```
sudo docker build -t myimage .
sudo docker volume create myvolume
sudo docker run -d --name mycontainer -v myvolume:/db -p 80:80 myimage
```

5.3 Documentation

Documentation has been carried out through the Sphinx library, which with the help of some tools automatically generates the documentation from the docstrings in the code. The documentation is available at
https://project-management-api.web.app

Explanatory use case

6.1 First step: Registration

User sends a POST request to /register with the following body:

Listing 6.1: register form

```
{
  "user_name": "doge",
  "password" : "doge_pass"
}
```

If no user exists with the specified name, the system adds the user to the database and returns:

Listing 6.2: register response

```
{
    "user_name": "doge"
}
```

If the user is the first registered user, the system grants the user all the authorizations.

6.2 Second step: Login

In order to use the API endpoints, the user must be authenticated. The user sends a POST request to /token with the same body as the register:

Listing 6.3: Login form

```
{
  "user_name": "doge",
  "password" : "doge_pass"
}
```

If the system finds correspondence of the username and password in the database, it will return a valid JWT token that the user can use. Example of response:

Listing 6.4: Token response

```
{
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
    eyJzdWIiOiJkb2dlIiwiZXhwIjoxNjU0MTg0MTM2fQ.
    SToW2wOkh2KrVRdQDfTaSZeG5Qajy9DzoXwB4HQNf0k",
    "token_type": "bearer"
}
```

6.3 Third step: Project creation

If the user has the permission to create projects, it can send a POST request to /projects Example request:

Listing 6.5: Project creation request body

```
# Project name
project_name: example_project
documents:
  project_charter:
    # jsonschema of the document
    jsonschema:
      properties:
        overview: { type: string }
        impact: { type: string }
        organization: { type: string }
      additionalProperties: false
    computed_fields:
      scope:
        reference_document: work_breakdown_structure
        jsonpath: $.elements[?(@.level == 1)].name
      scope:
        ms_project_name: example
        field_from: tasks
        jsonpath: $[?(@.level < 3)].name
  work_breakdown_structure:
    jsonschema:
      properties:
        elements:
          type: array
          items:
            properties:
              level: { type: integer }
              code: { type: string }
              name: { type: string }
              description: { type: string }
            additionalProperties: false
        additionalProperties: false
processes:
  # process name
```

```
  develop_project_charter:
    inputs:
    outputs: [ project_charter ]
  develop_work_breakdown_structure:
    inputs: [ project_charter ]
    outputs: [ work_breakdown_structure ]
permissions:
  # user name
  doge:
    documents:
      project_charter: [ view, edit, delete ]
      work_breakdown_structure: [ view, edit, delete ]
```

The server would return:

Listing 6.6: Project response

```
{
    "project_name": "example_project",
    "owner_name": "doge",
    "documents": [
        {
            "project_name": "example_project",
            "document_name": "project_charter",
            "author_name": "doge",
            "jsonschema": {
                ...
            },
            "first": {},
            "last": {},
            "creation_date": null,
            "patches": [],
            "computed_fields": {
                "scope": []
            },
            "ms_computed_fields": {}
        },
        {
            "project_name": "example_project",
            "document_name": "work_breakdown_structure",
            "author_name": "doge",
            "jsonschema": {
                ...
            },
            "first": {},
            "last": {},
            "creation_date": null,
            "patches": [],
            "computed_fields": {},
            "ms_computed_fields": {}
        }
    ]
}
```

6.4 Fourth step: Document content insertion

The endpoint for the document content insertion is
PUT /projects/example_project/documents/project_charter

Listing 6.7: Document content insertion

```
{
  "overview": "Academic research for implementing an environment for digital delivery of
     projects",
  "impact": "Creation of the environment...",
  "organization": "PoliTo Project Management Lab"
}
```

Listing 6.8: Document response

```
{
    "project_name": "example_project",
    "document_name": "project_charter",
    "author_name": "doge",
    "jsonschema": {
        ...
    },
    "first": {
        "overview": "Academic research for implementing an environment for digital
    delivery of projects",
        "impact": "Creation of the environment...",
        "organization": "PoliTo"
    },
    "last": {
        "overview": "Academic research for implementing an environment for digital
    delivery of projects",
        "impact": "Creation of the environment...",
        "organization": "PoliTo"
    },
    "creation_date": "2022-04-11T16:05:06.575202",
    "patches": []
    "computed_fields": {
          "scope": []
    },
    "ms_computed_fields": {
          "scope": []
    }
}
```

6.5 Fifth step: Edit document content

The endpoint for document editing is
PATCH /projects/example_project/documents/project_charter
In this example the user wants to edit the field “organization” from “PoliTo” to “PoliTo Project Management Lab”

Listing 6.9: document patch request body

```
{
  "organization": "PoliTo Project Management Lab"
}
```

Then the response would be

Listing 6.10: document patch response

```
{
    "project_name": "example_project",
    "document_name": "project_charter",
    "author_name": "doge",
    "jsonschema": {
        ...
    },
    "first": {
        "overview": "Academic research for implementing an environment for digital delivery of projects",
        "impact": "Creation of the environment...",
        "organization": "PoliTo"
    },
    "last": {
        "overview": "Academic research for implementing an environment for digital delivery of projects",
        "impact": "Creation of the environment...",
        "organization": "PoliTo Project Management Lab"
    },
    "creation_date": "2022-04-11T16:05:06.575202",
    "patches": [
        {
            "id": 1,
            "user_name": "doge",
            "updated_date": "2022-04-11T16:05:57.498469",
            "patch": [
                {
                    "op": "replace",
                    "path": "/organization",
                    "value": "PoliTo Project Management Lab"
                }
            ]
        }
    ],
    "computed_fields": {
          "scope": []
    },
    "ms_computed_fields": {
          "scope": []
    }
}
```

6.6 Sixth step: Adding a second document

Adding the work breakdown structure will provide a summary of the tasks to the project charter.

Listing 6.11: Work breakdown structure insertion body

```
{
  "elements": [
    {
      "level": 1,
      "code": "1",
      "name": "Task 1",
      "description": "descr"
    },
    {
      "level": 2,
      "code": "1.1",
      "name": "Task 1.1",
      "description": "descr"
    },
    {
      "level": 3,
      "code": "1.1.1",
      "name": "Task 1.1.1",
      "description": "descr"
    },
    {
      "level": 1,
      "code": "2",
      "name": "Task 2",
      "description": "descr"
    }
  ]
}
```

In the project charter document, the “computed_fields” value will be:

Listing 6.12: computed_fields value

```
"computed_fields": {
          "scope": [
              "Task 1",
              "Task 2"
          ]
    }
```

6.7 Seventh step: Adding a Microsoft Project file

The endpoint for adding Microsoft Project Files is POST /projects/example_project/msprojects/

Listing 6.13: MS Project file insertion response

```
{
  "project_name": "example_project",
  "ms_project_name": "example",
  "author_name": "doge",
  "update_author_name": "doge"
  "proj_info": {
    "baseline_start": "Thu May 16 08:00:00 CEST 2019",
    "baseline_finish": "Wed Dec 30 17:00:00 CET 2020",
    "currency_code": "EUR"
  },
  "resources": [
    {
      "name": "Workers IT",
      "id": "1"
    },
    {
      "name": "Workers IND",
      "id": "2"
    },
    ...
  ],
  "tasks": [
    ...
    {
      "name": "C1 - Start of Activities",
      "level": 2,
      "duration": "0.0d",
      "predecessors": [
        {
          "target_task": "M1 - Effective Date",
          "target_task_id": "2",
          "lag": "42.0ed",
          "type": "FS"
        }
      ],
      "ef": "Thu Jun 27 08:00:00 CEST 2019",
      "es": "Thu Jun 27 08:00:00 CEST 2019",
      "lf": "Thu Jun 27 08:00:00 CEST 2019",
      "ls": "Thu Jun 27 08:00:00 CEST 2019",
      "start": "Thu Jun 27 08:00:00 CEST 2019",
      "finish": "Thu Jun 27 08:00:00 CEST 2019",
      "cost": "0.0",
      "id": "3"
    },
    ...
  ],
}
```

The value of “ms_computed_fields” will be:

Listing 6.14: ms_computed_fields value after insertion

```
"ms_computed_fields": {
          "scope": [
              "2018-PMTermProject_BID_baseline",
              "New production line",
              "M1 - Effective Date",
```

```
            "C1 - Start of Activities",
            "Design",
            "C2 - Design completed",
            "Purcahse",
            "Manifacturing",
            "Transportation",
            "Testing",
            "M2 - Owner's Taking Over",
            "Civil Works",
            "O1 - Permits completed",
            "O2 - Civil works substantial completion"
        ]
    }
```

Findings

7.1 Conclusion

It has been demonstrated to be rather accessible to design and implement a platform to support the enabling of a digital transformation. At this development level, the API offers a sufficient number of features and allows the customization and addition of features with ease.

7.2 (TLR) Technology Readiness Level

It is possible to assert that the TLR for this development has reached level TRL6, since the platform has been tested to be ready to accept complex cases.