POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Politecnico di Torino

Master's Degree Thesis

Enabling Digital Project Management – Development of Backend Libraries

Supervisor:

Prof. Paolo Eugenio Demagistris

Candidate:

Emin Suleymanov

In collaboration with: PM-Lab of PoliTo

A. Y. 2021/2022

Abstract

Various digital tools exist for Project Management which applies various project management methodologies, and new tools are created or existing ones are improved due to needs of Project Management industry. Whereas, they may have some gaps. This thesis will develop a unified storable model to support financial flow of a project by applying object-oriented programming methodologies to project management software. OOP provides some benefits such as reusability, flexibility. The thesis has been accomplished in collaboration with Project Management Laboratory of Politecnico di Torino and purpose of the thesis is to develop software that presents reusable object models for project finance. This platform is considered as Python based web-application which enables digital environment for Project Management. The software will contain following elements: backend, frontend and API endpoints for interaction between them.

Table of Contents

Introduction

The Problems in Managing Projects

Many of the real issues a project manager faces are: requirement elucidation, senior management involvement, change control, risk management, user involvement, work breakdown and allocation, team management, technology and process change management, and tracking and monitoring [1].

According to many researches, the conventional approaches of project management have some problems. For instance, significantly large part of projects isn't finished on time or budget management problems exist. As a solution, Object-Oriented Project Management (O2PM) technique is provided.

Object-Oriented Project Management (O2PM)

The objective of object-oriented management is to provide a clear set of principles set into a framework that enables all participants while minimizing management overhead [2].

Object-Oriented Project Management applies Object-Oriented Programming paradigm to Project Management. Object-Oriented approach contains some main concepts such as classes and members, objects and attributes and so on. The five main activities of the object-oriented approach listed below.

1. finding classes and objects
2. identifying structures
3. identifying subjects
4. defining attributes
5. defining services

In a typical project context, the project manager uses software to establish a project plan, allocates tasks, deadlines, and effort to the team members, whereas any appropriate mechanism doesn't exist to encapsulate work and set boundaries. This makes accountability a significant problem, also problems occur with measuring the amount of work completed and producing effective reporting.

Most of these matters are resolved by O2PM. The core components are as follows:

- **Encapsulation** – team members can effectively work within specified bounds when work deliverables are encapsulated and boundaries are made apparent. Stronger governance and,

more significantly, accountability will result from encapsulating work in this way. Additionally, it makes identification of problems and their solution easy.

- **Inheritance** – establishes the project's architecture, rules, standards, and processes, which every team member and the work deliverables have to inherit them.

- **Polymorphism** – presents the meaning that a unique object can have multiple alternative representations

- **Communication** –the interfaces may be created that the encapsulated deliverables will use to communicate with others.

Object-Oriented Programming (OOP)

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or *properties*), and code, in the form of procedures (often known as *methods*) [3].

The following elements make up the Object-Oriented Programming structure:

- **Classes** – data types defined by user, which is used to construct separate objects, attributes, and methods.

- **Objects** – are instances of a class that are generated using specified data. Objects can be abstract entities or things in the real world.

- **Methods –** define the behavior of an object and placed inside a class. Methods are attached to class instances (objects) and allow to access and alter the object's data fields.

- **Attributes –** are described in the class template. When a new object of the class is created, attributes field contains particular data of the object.

Object-oriented programming is supported by the majority of the most widely used programming languages, including C++, Java, Python, etc. OOP provides many benefits such as modularity, reusability, productivity, security, flexibility.

Four key principles form the foundation of object-oriented programming:

- **Encapsulation** – information is encapsulated inside an object and only a limited amount of data is revealed. This aspect of data concealing contributes to increased program security.

- **Inheritance** – makes codes of other classes reusable. Programmers can reuse common logic by keeping a distinct hierarchy when assigning relationships and subclasses between objects.

- **Polymorphism** – objects and methods may share same behaviors more than one form. Polymorphism decreases the demand to duplicate code.

- **Abstraction** – is an object-oriented programming paradigm that "shows" only relevant properties and "hides" extraneous data.

Implementation

Used tools and technologies

Python

As a programming language, it's general-purpose and high-level. High-level programming language means that development of a computer program with python is simpler, allows to automate significant domains of computing systems, a code is more understandable, and more readable than using low-level programming languages. A meaning of general-purpose programming language is that in contrast to domain-specific programming languages, python is not considered for a solution of any specific problem, it can be applied on a solution of various areas. It supports a variety of programming paradigms, including functional programming and object-oriented programming. A simplicity of its syntax makes python popular rather than other high-level programming languages. It is used in various fields of software development, for instance, desktop applications, web application, data analysis etc.

Django

Django is a python-based web framework which allows to develop web applications rapidly, makes development of complex, database-driven web applications easy, takes care of security and assists developers to get rid of common security mistakes such as SQL injection, cross-site request forgery etc. It pursues MVT (model-view-template) architectural pattern. MVT is a software design pattern and consists of three main components Model, View and Template. The Model is a source of information about our data and corresponds a database table. The View is responsible for executing business logic, interacting with a carry data, and rendering a template. The Template is a presentation layer that takes care of the entire user interface. Moreover, django uses ORM (object-relational mapper) which intercedes between data models (defined as Python class) and relational database (Model). Django also presents an optional administrative panel which allows CRUD operations over data models and helps developers to save time during development.

REST API

API stands for application programming interface which is a set of protocols and definitions for computers or computer programs to communicate with one another.

REST (Representational State Transfer) is not a protocol or a standard, it's just a software architectural style that allows users to communicate with RESTful web services. It delivers a representation of the state of the resource to the requester or endpoint when a client sends request using A RESTful API. This data may be transferred via HTTP in one of several formats such as plain text, JSON, HTML. Although, multiple possible formats, the most preferred file format is JSON among them, because of its language-agnostic feature, it has good readability by either humans or machines.

REST API requests can be made by following HTTP methods:

GET – the most used HTTP method, returns a representational view of the contents and data of a resource.

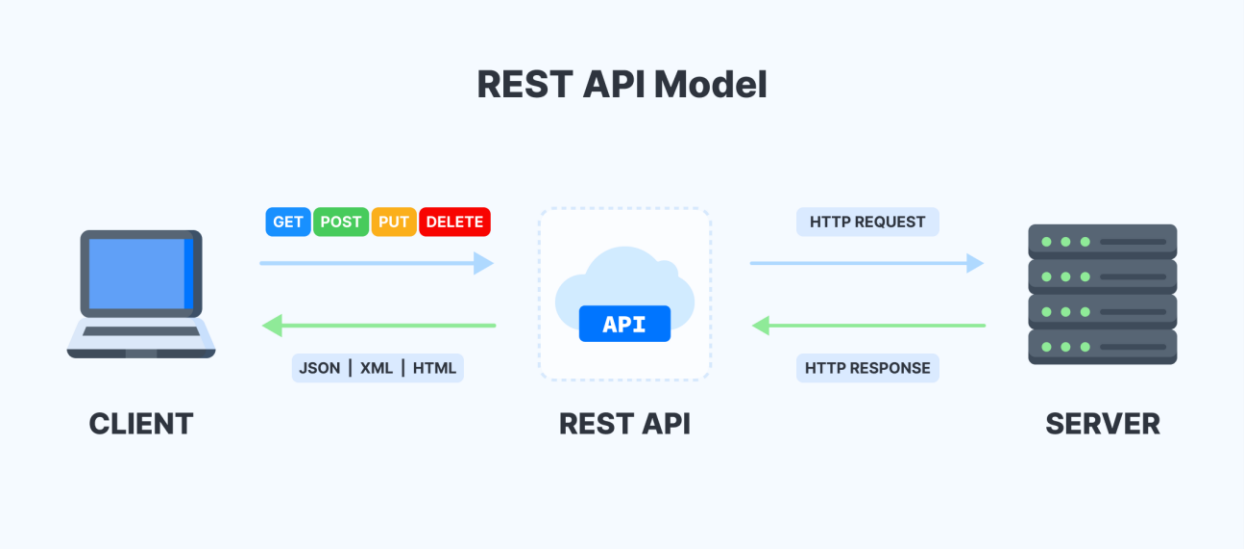
POST – the only HTTP method in the RESTful API that primarily works with resource collections. Creates a new resource and associate it with the proper hierarchy.

PUT – modifies a resource by completely replacing its content.

PATCH – another HTTP mechanism for updating resources, it just alters resource contents rather than replacing them, like the PUT technique does.

DELETE – deletes a single resource completely

Below diagram describes how data transferred between a client and a server using REST API.



Django REST framework

Django Rest Framework (DRF) is a robust and adaptable toolset that allows us to quickly create RESTful APIs using Django models. It represents class based generic API views and serializers, so with a few lines of code we can rapidly create views for api endpoints by avoiding common mistakes. Also, it provides “ApiTestCase” class to test api’s by integration test.

Django REST Knox

While creating a web application, security is one of the important domains that has been taken care of that. At this point, Knox framework offers easy to use authentication for API endpoints built with DRF. Knox is token-based authentication and fixes various issues with DRF's built-in Token Authentication. For instance, token is generated per one call in login views with Knox or it presents expiration of tokens etc.

JSON

JSON (JavaScript Object Notation) is a format for data exchange that is simple to use. Reading and writing are simple tasks for humans. Machines can easily parse and generate it. JSON is a language-independent text format that incorporates standards common to programmers of the C family of languages, like C, JavaScript, C++, Python, C#, and many others. JSON is an ideal data-transfer language because of these characteristics.

JSON is made up of two basic structures:

- Name/value pairs are grouped together as a collection. This is represented as an object, record, struct, dictionary, hash table, keyed list, or associative array in many languages.
- An ordered list of values. This is implemented as an array, vector, list, or sequence in most programming languages.

These are data structures that are universal. They are supported in some form or another by almost all modern programming languages. It's logical that a data format that can be used in various programming languages is based on these structures.

SQLite

SQLite is a compact software library that presents a relational database management system. It implements a transactional SQL database engine that is self-contained, serverless, and requires no configuration. Self-contained means it only needs little support of the operating system or a third-party library. Thus, it is applicable in any environment, including embedded devices, for instance, game consoles or Android phones. SQLite doesn’t use a separate server process like other SQL databases. A data is read and written directly to ordinary disk files. So, a single disk file contains

a whole SQL database, including many tables, indices, triggers, and views. All these features make it so popular among other database management systems.

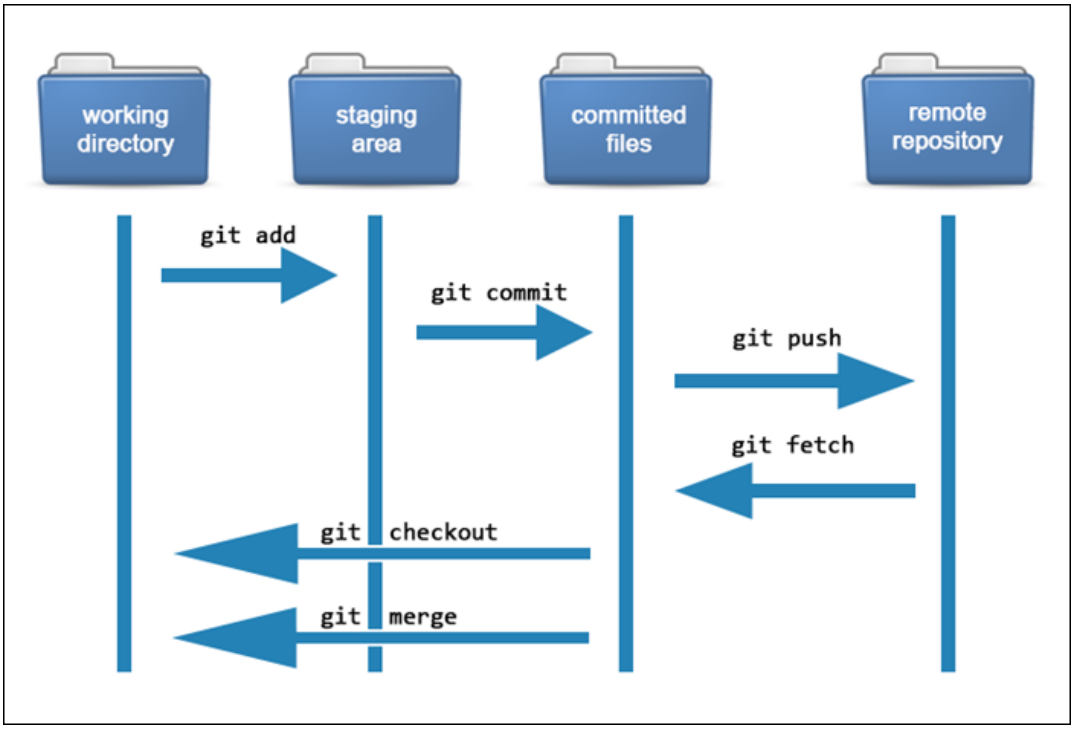
Git

In the world, Git is the most popular decentralized version control system which stands for Global Information Tracker. It's a piece of software that allows us to track changes made to any group of files. Git is built to manage a wide range of jobs, from tiny to extremely large in a rapid and efficient way. It's typically used to coordinate work among developers who are working on source code together while software development. An essential element of Git is a repository that contains a project. A repository might be locally saved or hosted on a website like GitHub. Commits are used to save the project's progress throughout development. All commits of the project are listed in the commit history. Furthermore, a commit creates a possibility of reverting or forwarding the code to any prior commit in the history.

The files of each Git project go through various phases:

- Working directory – files that have been modified but are not yet ready to commit because they are untracked.
- Staging directory – when updated files are added to the staging environment, it's indicated that they're ready to commit.
- Committed – the commit history contains snapshots of files from the staging area.

The following diagram indicates the Git workflow.



Jupyter notebook

Jupyter Notebook is an interactive computational environment for generating notebook documents that is accessible through the web. A Jupyter Notebook document is a web-based REPL (read-eval-print-loop) with an ordered list of input/output cells that may consist of code, rich media, text (Markdown), plots and mathematics. A notebook is a JSON document that follows a versioned format and commonly ends with the ".ipynb" extension beneath the interface. It may link to a variety of kernels for allowing to programming in many languages.

Project Management Software

Backend API's

It is a web application which is responsible for enabling of a digital environment for the project management. Main focus of the thesis is a backend side (server-side) of the web application that allows to manage funding and spending for steering of a project. Thus, this digital environment provides various functionalities such as control over stakeholder accounts, management of project budget, tracking of project spending etc. Moreover, various parts of a project are accessible by stakeholders within certain permissions. All of these functionalities are provided as API endpoints (REST API's) that allow a frontend side of the web application to communicate with backend. Frontend side is represented as a Jupyter notebook document to present outputs of API's.

Structure of Django project

The backend API's are developed as a Django project. Each Django project may have several python packages which one of them is a main package contains configurations of entire project and rest of them are django applications. In our case, main package is called *dpm_env* and contains some files such as settings, urls and so on. *Settings.py* file demonstrates default configurations of the project which consists of implemented django applications (python packages) including local and third-party packages, database configurations, REST framework configurations, security settings etc. Urls.py file plays a role of gateway inside the project. Because, redirecting of each api request to a related endpoint is realized by this url settings.

/dpm_env/urls.py

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('accounts.urls'), name='accounts'),
    path('projects/', include('projects.urls'), name='projects'),
```

```
    path('api-schema/', schema_view.with_ui(
        'swagger', cache_timeout=0), name='schema-swagger-ui'),
    path('redoc/', schema_view.with_ui(
        'redoc', cache_timeout=0), name='schema-redoc'),
]
```

Figure 1. URL patterns of the project

Structure of Django application

Our project consists of six main django applications. Django application is a Python package that allows us to split a django project into small packages due to business logic.

An application may use common Django conventions, such as having models, tests, urls, and views submodules [4].

Model – Django's built-in tool for creating tables, their fields, and other constraints.

View – responsible for accepting web request and returning response and describes business logic. Django views can be created as function-based or class-based.

Serializer – used in DRF to convert objects into data types that are understandable by front-end frameworks and vice versa.

Urls – a set of URL patterns are used by Django to determine dedicated view for the requested URL.

Test – Django offers a framework for tests that allows unit and integration tests.

Implemented django applications are:

- Accounts application
- Projects application
- Project Charter application
- Project Resources application
- Project Procurements application
- Project Budget application

Permissions

Access to each API endpoint requires a certain permission in the system. Permissions are defined using permissions module of DRF and Django guardian implementation. Some methods, such as *assign_perm, remove_perm, get_user_perms* are offered by Django Guardian which allows to assign certain permissions to each stakeholder for each project. Within these permissions a stakeholder can edit certain part of the project such as spending, project charter, resources, project

budget. For instance, a*dd_project_resource* permission allows a stakeholder to add a new resource to the project.

/project_resources/permissions.py

```
class hasAddProjectResourcePermission(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        permissions = get_user_perms(request.user, obj)
        return 'add_project_resource' in permissions
```

Figure 2. Object level permission example

Accounts application

It's responsible for administration of stakeholder accounts. Accounts application contains "User" model which is a Python class and corresponds a table with same name in the database. The properties of the User model demonstrated below.

/accounts/models.py

```
class User(AbstractBaseUser, PermissionsMixin):
    first_name = models.CharField(max_length=255, blank=False)
    last_name = models.CharField(max_length=255, blank=False)
    email = models.EmailField(max_length=255, unique=True)
    user_role = models.CharField(
        max_length=3,
        choices=[('PMO', 'PMO'), ('PM', 'PM'), ('PS', 'PS'), ('PC', 'PC'),
                 ('PSC', 'PSC'), ('PP', 'PP'), ('AS', 'AS'), ('U', 'U')],
        default='U'
    )
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    is_superuser = models.BooleanField(default=False)
```

Figure 3. User model

The properties *first_name, last_name* and *email* defined as required fields that must be provided, *email* field is also unique identifier that defines each user can have one account while registration of a new stakeholder. The field of *user_role* is considered to manage permissions and responsibilities of a stakeholder in the system. It represents several choices to be set as value which are:

- PMO – project management office
- PM – project manager
- PS – project sponsor
- PC – project controller
- PSC – project scheduler
- PP – project planner
- AS – administrative staff

- U – unknown

is_active field defines account status to conduct an access of a stakeholder to the system. Last two properties serve for accounts of administration staff.

Accounts application provides seven API endpoints to execute operation on the User data model.

- **Register API** – any specific permission is not required. Everybody can register by providing his credentials such as name, surname, email. Dedicated view class will handle request, forward data to linked serializer class, it will check validity of email, password etc. and a new account will be created if all credentials are valid. In response, it returns details of registered user.

- **Login API** – by providing email and password stakeholders can sign in to the system. Related serializer class finds user data from database due to given email, if account is active forwards data to linked view class. View class creates a new authentication token for user and returns them as response.

- **User Details API** – represents details of specified user account. Just account owner and pmo type user have access to the endpoint.

- **User List API** – only pmo type user can get an entire list of users by sending request to the API endpoint.

- **Activate User API** – pmo type user may re-activate suspended user account using the API endpoint to allow access to the system. PMO just needs to include user id in request URL.

- **Deactivate User API** – pmo type user may prevent login of a user by deactivating his account. It's enough to include specified user in the request.

- **Update User Role API** – directly after registration user role is set as unknown, however, user needs to be defined his user role to become a stakeholder of any project. So, this can be done by pmo type user by sending request to the API endpoint with related credentials.

Projects application

Projects application contains some basic data about the project. The *Project* model of projects application serves for generation of a new project instance and consists of data fields present primary information about a project which are *project_name, author of a project, starting date of a project and involved stakeholders.*

/projects/models.py

```
class Project(models.Model):
    project_name = models.CharField(max_length=255, blank=False)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created = models.DateTimeField(auto_now_add=True)
    stakeholders = models.ManyToManyField(User, blank=True,
         related_name="stakeholders")

    def actual_cost(self):
        resource_spending = project_budget.models.ResourceSpending.objects.all()
           .filter(project=self).filter(approval_status='approved')
           .aggregate(Sum('amount')).get('amount__sum')
        contract_spending = project_budget.models.ContractSpending.objects.all()
           .filter(project=self).filter(approval_status='approved')
           .aggregate(Sum('amount')).get('amount__sum')
        if not resource_spending:
            resource_spending = 0.0
        if not contract_spending:
            contract_spending = 0.0
        actual_cost= resource_spending + contract_spending

        return {
            "actual_cost": actual_cost,
            "resource_spending": resource_spending,
            "contract_spending": contract_spending
        }

    def __str__(self):
        return self.project_name

    class Meta:
        ordering = ('project_name',)
        permissions = (

            ('add_project_charter', 'Can add project charter/to project charter'),
            ('change_project_charter', 'Can change project charter'),
            ('delete_project_charter', 'Can delete project charter'),
            ('view_project_charter', 'Can view project charter'),

            ('add_additional_budget', 'Can add additional budget request'),
            ('change_additional_budget', 'Can change additional budget'),
            ('view_additional_budget', 'Can view additional budget'),

            ('add_project_resource', 'Can add project resource'),
            ('change_project_resource', 'Can change project resource'),
```

```
            ('delete_project_resource', 'Can delete project resource'),
            ('view_project_resource', 'Can view project resource'),

            ('add_project_contract', 'Can add project contract'),
            ('change_project_contract', 'Can change project contract'),
            ('delete_project_contract', 'Can delete project contract'),
            ('view_project_contract', 'Can view project contract'),

            ('add_project_spending', 'Can add project spending'),
            ('change_project_spending', 'Can change project spending'),
            ('delete_project_spending', 'Can delete project spending'),
            ('view_project_spending', 'Can view project spending'),
        )
```

Figure 4. Project model

The *Project* model provides a method named *actual_cost* that allows to track actual cost of entire project including spending for resources and procurements. Management of various operations over models is administrated via permissions (*project_charter, resource, contract, spending*) defined in addition to the default project permissions (*add_project, change_project, delete_project, view_project*) in the *Meta* class of *Project* model.

Thirteen API endpoints are defined in the Projects application to edit user permissions, involved stakeholders and project data.

- **Project API** – only pmo type users has access to the endpoint. Authorized *pmo* can generate a new project instance by sending primary credentials: *project_name and his id*. First stakeholder of project is an author of the project and all permissions are assigned automatically directly after creation of project.

- **Edit Project API** – allows to edit project name. Requires a permission of *change_project*.

- **Delete Project API** – a stakeholder who has *delete_project* permission can delete entire project using project's id.

- **Project Details API** – requires a permission of *view_project* and provides all details of a project including budget, project charter, business case swots etc.

- **Get projects of stakeholder API** – each authorized stakeholder can get a list of his projects with brief information about per project. It doesn't require any individual permission.

- **Add stakeholder to project API** – a user with *add_project* permission may add a new stakeholder to the project. Presenting of project id and stakeholder account id is enough to make user a stakeholder of the project.

- **Remove stakeholder from project API** – requires to have *delete_project* per mission. A stakeholder may be deleted from stakeholder list of the project by sending API request to the endpoint. Project id and stakeholder account id have to be inserted to the body of API request.

- **Get stakeholders of project API –** a user who has *view_project* permission may get a list of stakeholders of the project. Just required parameter for the request is id of specified project.

- **Get actual cost of project API –** provides brief information about actual cost of entire project including resource spending, procurement spending. A user who has *view_project* permission may access to the endpoint.

- **Assign project permissions to stakeholder API –** an author of a project may assign project permissions to a stakeholder of the project using the endpoint. He needs to specify needed permissions for assignment in the API request. The system has various kinds of permissions (such as budget permissions, spending permissions) in addition to four main project permissions: *add_project, change_project, delete_project, view_project*. These permissions allow to access the endpoints of applications.

- **Assign all project permissions to stakeholder API –** all permissions can be assigned to a stakeholder in a single request using this API endpoint without need to specify permissions list. Only author of the project has access to the endpoint.

- **Delete project permissions of stakeholder API –** only an author of the project can access to the endpoint to remove project permissions of a stakeholder. He must include a list of permissions which there is need to delete.

- **Get permissions of stakeholder API –** an author of the project or owner of the user account may get permissions list of given stakeholder. So, this list contains all permissions of the stakeholder for the project.

Project Charter application

It is in charge of regulating the project charter including business case swot (strengths, weaknesses, opportunities, threats). This application consists of two data models, namely, *project charter* and *business case swot.*

/project_charter/models.py

```
class ProjectCharter(models.Model):
    project = models.OneToOneField(Project, related_name='project_charter',
      on_delete=models.CASCADE)
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created = models.DateTimeField(auto_now_add=True)
```

```
    last_updated = models.DateTimeField(auto_now=True)
    sow = models.CharField(max_length=1024, blank=True, null=True)
    contract = models.CharField(max_length=1024, blank=True, null=True)
    business_case = models.CharField(max_length=1024, blank=True, null=True)

    def __str__(self):
        return self.project.project_name

    class Meta:
       ordering = ('project',)
```

Figure 5. Project Charter model

project field of project charter model identifies that project charter belongs to which project. Each project can have just one project charter in the system. Furthermore, as it appears from their names *author, created, last_updated* fields declare some fundamental data about a project charter. *sow (project statement of work), contract* and *business_case* properties are described as primary inputs to generate project charter.

/project_charter/models.py

```
class BusinessCaseSWOT(models.Model):
    project_charter = models.ForeignKey(ProjectCharter,
       related_name='bus_case_swot', on_delete=models.CASCADE)
    swot_type = models.CharField(
        max_length=11,
        choices=[('strength', 'strength'), ('weakness', 'weakness'),
            ('opportunity', 'opportunity'), ('threat', 'threat'),]
    )
    content = models.CharField(max_length=1024, blank=True)

    def __str__(self):
        return self.swot_type

    class Meta:
        ordering = ('project_charter',)
```

Figure 6. Business Case SWOT model

Every project charter may have multiple business case swot instances. *project_charter* property identifies linked project charter instance for generated business case swot item. SWOT is an acronym for strengths, weaknesses, opportunities and threats. Thus, business case swot model includes a property named *swot_type* that defines a category of business case swot item among strengths, weaknesses, opportunities and threats. *content* field describes a text body of business case swot item.

Eight API endpoints listed below provides a possibility for editing project charter and business case swot items.

- **Project Charter API –** a user who has *add_project_charter* user permission or author of the project can generate project charter for the project by sending required credentials to the endpoint. These credentials are project statement of work, contract, business case etc.

- **Edit Project Charter API –** allows to edit project charter partially. Project statement of work, contract, business case fields may be edited by declaring in the body of the request. The endpoint requires to have *change_project_charter* user permission.

- **Project Charter Details API –** provides detailed information about the project charter of specified project. Thus, entire information about budget, business case swot items etc. are included in the response body. A user who sends API request just need to have *view_project_charter* user permission.

- **Delete Project Charter API –** simply deletes a project charter including all data which is linked to the project charter such as business case swot instances. Requires a user permission of *delete_project_charter*.

- **Business Case SWOT API –** inserts a new business case swot instance to the project charter. A user needs *add_project_charter* user permission to add a new business case swot item to the project charter.

- **Business Case SWOT Details API –** presents a single business case swot instance in details using its database id. To access to the endpoint a user must have *view_project_charter* permission.

- **Business Case SWOT List of Project Charter API –** provides a list of all business case swot items inserted in the project charter. It requires *view_project_charter* user permission to access to the endpoint.

- **Delete Business Case SWOT API –** removes a business case swot item from the project charter. A user must have *delete_project_charter* user permission to remove it from the project charter.

Project Resources application

The application represents API endpoints to take care of project resources. Project Resources application contains *Resource* model that is data structure for project resources instance.

/project_resources/models.py

```
class Resource(models.Model):
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField(max_length=1024, default=None)
    cost = models.FloatField(default=0.00)
    unit = models.CharField(max_length=255, default='euro')
    category = models.CharField(max_length=255)

    def __str__(self):
        return self.name

    class Meta:
        ordering = ('project', 'name')
```

Figure 7. Resource model

A resource item is linked to a project which is identified by *project* property in the *Resource* model. The *name* property determines a title for a project resource item. The *description* property is considered for explanatory information about a project resource item. c*ost* and *unit* fields define respectively, unit cost of resource and unit as measure which default value is *euro* for *unit* property. In a project, resources may be classified in various categories such as human, material and so on. The *category* property allows to categorize project resources.

The application consists of five API endpoints to assist users for handling project resources.

- **Add Resource API –** each stakeholder of the project may access to the endpoint for insertion of a new resource to the project if the stakeholder has *add_project_resource* permission.

- **Update Resource API –** presents a possibility to edit *name, description, cost* and *category* properties of a project resource item. Required user permission is *change_project_resource* for the API endpoint.

- **Get Resource API –** a stakeholder may get details of a single project resource item. A stakeholder needs *view_project_resource* permission to request information about details of project resource.

- **Get Resource List of Project API –** provides a list of all resources inserted to the project. A stakeholder must have *view_project_resource* permission to get the list.

- **Remove Resource API –** requires *delete_project_resource* permission to allow access to the API endpoint. It deletes mentioned project resource permanently.

Project Procurements application

Project Procurements application covers all agreements for third-party purchases in a project. Thus, the application presents *Contract* model that is declared as data structure for purchase agreements.

/project_procurements/models.py

```
class Contract(models.Model):
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    product = models.CharField(max_length=255)
    description = models.TextField(blank=True, max_length=1024)
    unit_price = models.FloatField(default=0)
    unit = models.CharField(max_length=255, default='euro')
    assignment = models.IntegerField(default=0)
    supplier = models.CharField(max_length=255)
    date = models.DateField()


    def total_cost(self):
        return self.unit_price * self.assignment

    def __str__(self):
        return self.product

    class Meta:
        ordering = ('project', 'date')
```

Figure 8. Contract model

The *project* property of the model defines that contract belongs to which project. The field of *product* determines a title for purchased product, *description* property describes details of purchase. *unit_price* and *unit* properties define cost of purchased product for each unit and unit measure, and *assignment* property expresses amount of product agreed for purchase with third-party supplier which is demonstrated in the *supplier* field of the model. *date* field defines creation date of the contract. Additionally, *Contract* model contains *total_cost* method that is responsible for computing total payment amount for the purchase.

Management of procurement contracts are carried out by five API endpoints of Project Procurements application.

- **Add Contract API –** generates a new procurement contract based on described contract details in the request. A stakeholder must have a user permission *add_project_contract* to access to the API endpoint.

- **Update Contract API –** allows to edit contract data and requires *change_project_contract* permission for given contract to update the contract details.

- **Get Contract Details API –** an authorized stakeholder may obtain all details of the requested contract. *view_project_contract* permission is required for the API endpoint.

- **Get All Contracts of Project API –** returns entire list of created contracts of the project declared in the request. It's mandatory to have *view_project_contract* permission for accessing to the endpoint.

- **Delete Contract API –** removes a specified contract from the system. The endpoint requires *delete_project_contract* user permission to proceed with operation.

Project Budget application

The application covers operations on project budget and spending. Four data models, namely, *ProjectBudget, AdditionalBudget, ResourceSpending, ContractSpending* are replaced in the Project Budget application.

Budget of a project is defined as a collection of annual budgets during the project progress. Thus, *ProjectBudget* model is considered as data model for annual budget.

/project_budget/models.py

```
class ProjectBudget(models.Model):
    project_charter = models.ForeignKey(ProjectCharter,
       related_name='project_budget', on_delete=models.CASCADE)
    year = models.IntegerField(('year'), choices=year_choices(),
        default=current_year())
    budget = models.FloatField(default=0.00)

    def actual_cost(self):
        resource_spending = ResourceSpending.objects.all().filter(budget=self)
           .filter(approval_status='approved').aggregate(Sum('amount'))
           .get('amount__sum')
        contract_spending = ContractSpending.objects.all().filter(budget=self)
           .filter(approval_status='approved').aggregate(Sum('amount'))
           .get('amount__sum')
        if not resource_spending:
            resource_spending = 0.0
        if not contract_spending:
            contract_spending = 0.0
        actual_cost= resource_spending + contract_spending
        return {
            "year": self.year,
            "budget": self.budget,
            "actual_cost": actual_cost,
```

```
            "resource_spending": resource_spending,
            "contract_spending": contract_spending
        }

    def __str__(self):
        return str(self.year)

    class Meta:
        ordering = ('project_charter', 'year')
```

Figure 9. Project Budget model

Project budget is described in a simple structure. It has three properties that express linked project charter, budget year and amount of budget. Although, its structure is simple, it provides a method called *actual_cost* which is so practical. The method computes total spending for annual budget and returns information about annual spending with details.

Sometimes project spending may exceed annual budget which project manager may need to request additional funds to cover these expenses. *AdditionalBudget* model is considered for management of additional budget.

/project_budget/models.py

```
class AdditionalBudget(models.Model):
    budget = models.ForeignKey(ProjectBudget, on_delete=models.CASCADE)
    amount = models.FloatField(default=0.0)
    date = models.DateTimeField(auto_now_add=True)
    status = models.CharField(
        max_length=8,
        choices=[('waiting', 'waiting'), ('approved', 'approved'),
            ('denied', 'denied')],
        default=('waiting')
    )

    def __str__(self):
        return str(self.amount)

    class Meta:
        ordering = ('budget',)
```

Figure 10. Additional Budget model

budget property identifies annual budget that additional fund belongs. Amount of request fund is determined by *amount* property. Current date and time of additional fund request is automatically inserted to the *date* field to identify creation date and time of the request. *status* property expresses confirmation status of created additional budget request which default value is *waiting* and may be replaced with *approved* or *denied* during acceptance of the request.

Project expenses are divided into resource spending (internal) and contract spending (procurement), and data models are presented for each of them.

Resource spending is responsible for internal payments based on project resources and presented by *ResourceSpending* model.

/project_budget/models.py

```
class ResourceSpending(models.Model):
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    resource = models.ForeignKey(Resource, on_delete=models.CASCADE)
    budget = models.ForeignKey(ProjectBudget, on_delete=models.CASCADE)
    assignment = models.IntegerField(default=0)
    amount = models.FloatField(default=0.00)
    description = models.TextField(blank=True, max_length=1024)
    date = models.DateField()
    approval_status = models.CharField(max_length=9,
            choices=[('approved', 'approved'), ('denied', 'denied')] ,
            default=('approved')
    )

    def __str__(self):
        return str(self.resource)

    class Meta:
        ordering = ('project', 'budget')
```

Figure 11. Resource Spending model

project, resource and budget properties express linked project, resource and budget. *assignment* property is used to identify total payment amount of a spending which defined by *amount* property and automatically calculated by the system during insertion of a new spending. *description* property contains text data about details of a spending. *date* property identifies date of a spending and allows the system to control expenditure's date that matches with a year of annual budget to avoid wrong budget usage. A spending has *approval_status* property that describes validity of a spending. The system automatically computes actual cost for the budget including current spending, if it's over budget the spending is stored with a status of *denied*, otherwise approval status becomes *approved.*

Procurement spending is described *ContractSpending* data model which determines a payment for a predefined procurement contract.

/project_budget/models.py

```
class ContractSpending(models.Model):
    project = models.ForeignKey(Project, on_delete=models.CASCADE)
    contract = models.ForeignKey(Contract, on_delete=models.CASCADE)
    budget = models.ForeignKey(ProjectBudget, on_delete=models.CASCADE)
    amount = models.FloatField(default=0.00)
```

```
    description = models.TextField(blank=True, max_length=1024)
    date = models.DateField()
    approval_status = models.CharField(max_length=9,
            choices=[('approved', 'approved'), ('denied', 'denied')] ,
            default=('approved')
    )

    def __str__(self):
        return str(self.contract)

    class Meta:
        ordering = ('project', 'budget')
```

Figure 12. Resource Spending model

It has similar properties with *ResourceSpending* model, whereas it's linked to a predefined procurement contract instead of project resource. *Amount* property defines final payment amount and automatically computed by system based on the linked contract.

Functionalities of the Project Budget application are controlled by following twenty-three API endpoints.

- **Project Budget API –** allows to set annual budget of a project. It can be done with *add_project_charter* user permission. Budgets of several years may be allocated at the same time.

- **Edit Project Budget API –** provides functionality to update data of an annual budget. Required user permission is *change_project_charter.*

- **Actual Cost of Project according to Budget API –** computes total spending of a project which related to a mentioned annual budget. A stakeholder needs *view_project_charter* permission to access to the endpoint.

- **Project Budget Details API –** presents details of a mentioned annual budget. It requires *view_project_charter* user permission for access.

- **Total Project Budget API –** returns a list of all budgets related to project charter. An authorized user must have *view_project_charter* user permission to request budget list.

- **Delete Project Budget API –** permanently removes mentioned annual budget instance. It requires *delete_project_charter* user permission to delete budget instance from linked project charter.

- **Delete Total Project Budget API –** deletes all annual budget instances from specified project charter. *delete_project_charter* user permission is needed to access to the endpoint.

- **Request Additional Budget API –** additional funds may be requested from a project sponsor when annual budget is not sufficient to complete tasks. By using the API endpoint, a stakeholder who has *add_additional_budget* permission could place additional budget request in the system to be confirmed by project sponsor. A fund request is created in addition to an annual budget of a project.

- **Update Additional Fund Request Status API –** allows a project sponsor to define a status of additional budget request. He can accept or reject additional fund request by having *change_additional_budget* user permission.

- **Additional Budget Requests API –** requires *view_additional_budget* user permission, and returns a list of all additional fund requests for specified project.

- **Additional Budget Details API –** returns details of a mentioned additional budget request. *view_additional_budget* permission is needed to access to the API endpoint.

- **Add Resource Spending API –** creates a new resource spending instance. Firstly, accepted data is validated to identify validity of spending date, or actual cost including the new spending is under budget (including additional funds), if all conditions are satisfied the spending is stored and approved by the system. Access to the endpoint requires *add_project_budget_spending* permission.

- **Update Resource Spending API –** requires *change_project_budget_spending user permission, and* allows to edit specified resource spending instance. If provided data is valid, the instance is updated, otherwise its status is changed to *denied.*

- **Resource Spending Details API –** presents detailed information about mentioned resource spending instance. Needed user permission for access to the endpoint is *view_project_budget_spending.*

- **Resource Spendings by Budget API –** represents a list of resource spendings related to specified annual budget of a project. A user permission of *view_project_budget_spending* is needed for the endpoint.

- **Delete Resource Spending API –** removes requested resource spending instance. *delete_project_budget_spending* user permission is needed for the operation.

- **Add Contract Spending API –** checks validity of accepted data that date matches with mentioned annual budget, actual cost of project for the budget is not over the annual budget including additional funds, if the data is valid, inserts a new contract spending instance to the system. The API endpoint requires *add_project_budget_spending* permission to add a new spending to the budget.

- **Update Contract Spending API –** allows to edit mentioned contract spending instance. *change_project_budget_spending* user permission is needed for editing.

- **Contract Spending Details API –** is responsible for providing detailed information about specified contract spending instance. A stakeholder needs *view_project_budget_spending* permission to get details of a contract spending instance.

- **Contract Spendings by Budget API –** a list of all contract spending instances are returned based on mentioned annual budget of a project. The endpoint requires *view_project_budget_spending* permission for access.

- **Delete Contract Spending API –** requires *delete_project_budget_spending* permission for access, and removes specified contract spending instance permanently.

- **Forecast Future Spending API –** allows to forecast future spending using *earned value management* model. The application contains a class called *EVA* that applies Earned Value Management model. The EVA class accepts some data such as number of scheduled tasks, task duration etc., and provides a lot of methods such as budgeted cost, cost variance etc. assists to forecast future spending.

/project_budget/eva.py

```
class EVA:
    def __init__(self, **kwargs):
        self._total_scheduled_tasks = kwargs['total_scheduled_tasks']
        self._task_duration = kwargs['task_duration']
        self._employee_cost_day = kwargs['employee_cost_day']
        self._actual_activity = kwargs['actual_activity']
       ...

    def budgeted_cost(self):
        return self._multiply(self.employee_cost_hour(),
            self.scheduled_acticity())
       ...
```

Figure 13. Earned Value Analysis class

The endpoint requires *view_project_budget_spending* permission for access.

- **Forecast Balance API –** forecasts future balance using *EVA* class. It computes *estimate_to_complete* and *estimate_at_complete* values for future balance applying EVM method. *view_project_budget_spending* permission is needed for access to the endpoint.

Frontend

Frontend side of the software is developed as a Jupyter notebook document just to have clear visual presentation of system. API requests are sent from this document according to use case of the system and obtained results are demonstrated as tables, graphs using *pandas* library.

Informative Use Case

Registration

User addresses a POST request to …*/accounts/register/* by including following body:

Registration body

```
{
    "first_name": "Luca",
    "last_name": "Verdi",
    "email": "pmo@email.com",
    "password": "pasword",
    "confirm_password": "password"
}
```

If data is valid, and there is not any registered user with this email address, a new user account is created, result of registration and account data is returned as response.

Registration response

```
{
  "detail": "User resgistered succesfully!",
  "user": {
    "id": 8,
    "first_name": "Luca",
    "last_name": "Verdi",
    "email": "pmo@email.com",
    "user_role": "U",
    "is_active": true
  }
}
```

As default, user role is *U(stands for unknown)*. System administrator defines user role of PMO using administration panel (*.../admin/*), then *project management office* can manage other user accounts and define their role in the system.

Definition of User Role

Project management office sends a PATCH request to */accounts/user-account/user-role/update/* by defining user role in the request body.

User role body for project sponsor

```
{
    "user_role": "PS"
}
```

If operation is completed successfully, it returns 204 HTTP status code.

Deactivating and Activating User Account

Project management office sends a PATCH request to */accounts/user-account/deactivateuser/* with empty body for deactivating user account.

Deactivating User account response

```
{
      "detail": "User deactivated succesfully"
}
```

Project management office sends a PATCH request to */accounts/user-account/activateuser/* with empty body for activating user account.

Activating User account response

```
{
      "detail": "User activated succesfully"
}
```

Login

Login endpoint is responsible for authorization of user to use other API endpoints. User addresses a POST request to */accounts/login/* by including email and password in the body.

Login body

```
{
    "email": "pmo@email.com",
    "password": "password"
}
```

If user account is active, the endpoint generates an authorization token for specified user, and returns auth-token together with account data in response.

Login response

```
{
  "user": {
    "id": 3,
    "first_name": "Luca",
    "last_name": "Verdi",
    "email": "pmo@email.com",
```

```
"user_role": "PMO",
    "is_active": true
  },
  "auth_token": "58dfcc483a385c16e9050ed2f331f256e640e2c334a0d98e8c21d4831d3036f9"
}
```


Project Creation

Project management office addresses a POST request to */projects/create/* with following body.

Project Creation body


```
{
    "project_name": "thesis project",
    "author": user-id
}
```


The endpoint generates a new project instance and makes the author the first stakeholder of the project, and returns generated project data in response.

Project Creation response


```
{
  "detail": "Project created successfully.",
  "project": {
    "id": 2,
    "project_name": "thesis project",
    "author": {
      "id": 3,
      "first_name": "Luca",
      "last_name": "Verdi",
      "email": "pmo@email.com",
      "user_role": "PMO"
    },
    "created": "2022-06-20T16:09:33.234615Z",
    "stakeholders": [
      {
        "id": 3,
        "first_name": "Luca",
        "last_name": "Verdi",
        "email": "pmo@email.com",
        "user_role": "PMO"
      }
    ],
    "project_charter": null
  }
}
```

Adding Stakeholder to Project

Project management office sends a PATCH request to */projects/example-project/stakeholders/add/* by inserting specified users in the request.

Adding Stakeholder body

```
{
    "stakeholders": [5, 6]
}
```

The endpoint informs about result in response.

Adding Stakeholder response

```
{
  "detail": "Stakeholders added successfully"
}
```

Assigning User Permission

User permissions may be assigned in two ways, all together or partly.

Project management office may send a POST request to */projects/permissions/add/* with following request body to assign permissions partly.

Assigning User Permissions partly - body

```
{
    "user_id": 4,
    "project_id": 1,
    "permissions": ["change_project","view_additional_budget","add_project_charter"]
}
```

For assigning of all permissions in one request, *project management office* must send a POST request to */projects/permissions/assign-all/* with following request.

Assigning User Permissions all - body

```
{
    "user_id": 4,
    "project_id": 1
}
```

If assignment is successful, the endpoints return 201 HTTP status code.

Adding Project Charter

Stakeholder has to send a POST request to */projects/project-charter/create/* with following request body.

Adding Project Charter body

```
{
    "project": 4,
    "author": 4,
    "sow": "standard thesis outline",
    "contract": "Guida Studente DIGEP per la tesi",
    "business_case": "Research Case Digital PM"
}
```

If the stakeholder has *add_project_charter* permission, requested project charter is generated and details returned in response with 201 HTTP status code.

Adding Project Charter response

```
{
  "detail": "Project charter created successfully.",
  "project_charter": {
    "id": 2,
    "project": 2,
    "author": 3,
    "created": "2022-06-20T16:56:08.583088Z",
    "last_updated": "2022-06-20T16:56:08.583088Z",
    "sow": "standard thesis outline",
    "contract": "Guida Studente DIGEP per la tesi",
    "business_case": "Research Case Digital PM",
    "bus_case_swot": [],
    "project_budget": []
  }
}
```

Editing Project Charter

Stakeholder sends a PATCH request to */projects/project-charter/example-project-charter/edit/* by inserting new data in the request body to edit project charter.

Editing Project Charter body

```
{
    "sow": "standard thesis outline",
    "contract": "Guida Studente DIGEP per la tesi",
    "business_case": "Research Case Digital PM"
}
```

If the user has *change_project_charter* permission, the project charter is updated and result message returned in response with 200 HTTP status code.

Editing Project Charter response

```
{
  "detail": "Project charter updated successfully"
}
```

Adding Business Case SWOT

Stakeholder addresses a POST request to */projects/project-charter/swot/add/ with following* request body.

Adding Business Case SWOT body

```
{
    "project_charter": 2,
    "swot_type": "opportunity",
    "content": "A standard for PM exists"
}
```

If the stakeholder has user permission to add something to the project charter, a new business case swot is added, result message and business case swot instance returned in response with 201 HTTP status code.

Adding Business Case SWOT response

```
{
  "detail": "Business case swot added successfully.",
  "swot": {
    "id": 5,
    "project_charter": 2,
    "swot_type": "opportunity",
    "content": "A standard for PM exists"
  }

}
```

Budget Allocation

If a stakeholder has user permission to allocate project budget, he may address a POST request to */projects/project-charter/budget/set/example-project-charter/* with a list of budgets request body.

Budget Allocation body

```
[
    {
        "year": 2022,
        "budget": 23000.00
    },
    ...
]
```

After allocation of budget, the endpoint returns a list of allocated budget instances in response with 201 HTTP status code.

Budget Allocation response

```
[
  {
    "id": 7,
    "project_charter": 2,
    "year": 2022,
    "budget": 23000.0
  },
  ...
]
```

Adding Project Resource

To add a new resource to a project, a stakeholder must have certain user permission. POST request is sent to */projects/resources/add/* with details of a new resource in the request body.

Project Resource body

```
{
    "project": 2,
    "name": "engineer",
    "description": "price in eur/hour",
    "cost": 12.00,
    "category": "human"
}
```

The endpoint inserts the new resource to the system and returns following response with 201 HTTP status code.

Adding Project Resource response

```
{
  "detail": "Resource added successfully.",
  "resource": {
    "id": 4,
    "project": 2,
    "name": "engineer",
    "description": "price in eur/hour",
    "cost": 12.0,
    "unit": "euro",
    "category": "human"
  }
}
```

Adding Resource Spending

Stakeholder needs special user permission for spending. He must send POST request to */projects/project-charter/budget/resource-spending/add/* with details of spending in the request body.

Adding Resource Spending body

```
{
    "project": 2,
    "resource": 4,
    "budget": 7,
    "assignment": 10,
    "description": "some information about spending",
    "date": "2022-05-12"
}
```

The endpoint controls actual cost of project, if it's under budget stores the new spending with status of *approved* and returns following response.

Adding Resource Spending response

```
{
  "detail": "Resource Spending added successfully.",
  "resource-spending": {
    "id": 3,
    "project": 2,
    "resource": 4,
    "budget": 7,
    "assignment": 10,
    "amount": 120.0,
    "description": "some information about spending",
    "date": "2022-05-12",
    "approval_status": "approved"
  }
}
```

Adding Procurement Contract

A stakeholder must have *add_project_contract* permission to add new contract. POST request is sent to */projects/procurements/contracts/add/* with details of the contract in the body.

Procurement Contract body

```
{
    "project": 2,
    "product": "wood",
    "description": "wood with good quality",
    "unit_price": 2.50,
```

```
    "assignment": 100,
    "supplier": "New Wood MMC",
    "date": "2022-06-09"
}
```

Returned response consists of result message and created contract instance.

Procurement Contract response

```
{
  "detail": "Contract added successfully.",
  "contract": {
    "id": 2,
    "project": 2,
    "product": "wood",
    "description": "wood with good quality",
    "unit_price": 2.5,
    "unit": "euro",
    "assignment": 100,
    "supplier": "New Wood MMC",
    "date": "2022-06-09"
  }
}
```

Adding Contract Spending

Stakeholder must send POST request to */projects/project-charter/budget/contract-spending/add/* by inserting following request body.

Adding Contract Spending body

```
{
    "project": 2,
    "contract": 2,
    "budget": 7,
    "description": "details of spending",
    "date": "2022-06-12"
}
```

Actual cost of project is controlled by the system, if it's under budget the spending saved with status of *approved*, and following response is returned.

Adding Contract Spending response

```
{
  "detail": "Contract Spending added successfully.",
  "contract-spending": {
    "id": 1,
    "project": 2,
```

```
"contract": 2,
    "budget": 7,
    "amount": 250.0,
    "description": "details of spending",
    "date": "2022-06-12",
    "approval_status": "approved"
  }
}
```


Acquisition of Actual Cost

The system allows to calculate actual cost in two ways: due to annual budget and due to entire project.

Actual cost of entire project is acquired by addressing GET request to */projects/example-project/actual-cost/get/.*

Actual Cost of Project - response


```
{
  "actual_cost": 490.0,
  "resource_spending": 240.0,
  "contract_spending": 250.0
}
```


To get actual cost due to budget, a stakeholder must send GET request to */projects/project-charter/budget/example-budget/actual-cost/get/.*

Actual Cost of Annual Budget - response


```
{
  "year": 2022,
  "budget": 23000.0,
  "actual_cost": 230.0,
  "resource_spending": 110.0,
  "contract_spending": 120.0
}
```


Forecasting Future Spending

Authorized stakeholder may send POST request to */projects/project-charter/budget/example-budget/forecast-future-spending/* by inserting following body in the request to forecast future spending.

Forecasting body


```
{
    "total_scheduled_tasks": 64,
    "task_duration": 8,
    "employee_cost_day": 800,
```

```
    "actual_activity": [4, 5, 3, 4, 5, 7, 6, 3]
}
```

Forecasting is made by applying Earned Value Management model and all details provided in response.

Forecasting response

```
{
  "planned_work_percentage": [0.125, 0.25, ..., 1.0],
  "budget_work": [8, 16, ..., 64],
  "actual_work_percentage": [0.0625, 0.140625, ..., 0.578125],
  "budgeted_cost_of_work_scheduled": [800.0, 1600.0, ..., 6400.0],
  "actual_cost_of_work_performed": [800.0, 1600.0, ..., 6400.0],
  "budgeted_cost_of_work_performed": [400.0, 900.0, ..., 3700.0],
  "earned_value": [400.0, 900.0, ..., 3700.0],
  "schedule_performance_index": [0.5, 0.5625, ..., 0.578125],
  "schedule_variance": [-400.0, -700.0, ..., -2700.0],
  "schedule_variance_percentage": [-0.5, -0.4375, ..., -0.421875],
  "cost_variance": [-400.0, -700.0, ..., -2700.0],
  "cost_performance_index": [0.5, 0.5625, ..., 0.578125],
  "cost_variance_percentage": [-1.0, -0.7777777777777778, ..., -0.7297297297297297],
  "estimate_to_complete": [800.0, 1244.4444444444443, ..., 4670.27027027027],
  "estimate_at_complete": [1600.0, 2844.4444444444443, ..., 11070.27027027027],
  "cost_variance_at_completion": [-800.0,-1244.4444444444443, ...,-4670.27027027027],
  "time_variance": [-4.0, -7.0, ..., -27.0],
  "time_variance_percentage": [-0.04, -0.07, ..., -0.27],
  "earned_schedule": [4.0, 9.0, ..., 37.0],
  "time_performance_index": [0.5, 0.5625, ..., 0.578125],
  "estimated_accomplishment_rate": [50.0, 56.25, ..., 57.8125],
  "time_estimate_at_completion": [16.0, 28.444444444444443, ..., 110.70270270270271],
  "time_variance_at_completion": [8.0, 12.444444444444443, ..., 46.70270270270271]
}
```

Requisition Additional Funds

project manager is able to request additional budget from project sponsor, if he has certain user permission. For additional funds he may send POST request to */projects/project-charter/budget/additional-budget/request/* by inserting additional amount in request.

Request body

```
{
    "project": 2,
    "budget": 7,
    "amount": 2000.00
}
```

When additional budget request is placed in the system, its status becomes *waiting* until it is confirmed by project sponsor.

Request response


```
{
  "detail": "Additional budget request placed successfully.",
  "additional-budget": {
    "id": 7,
    "budget": 7,
    "amount": 2000.0,
    "date": "2022-06-20T19:25:24.768122Z",
    "status": "waiting"
  }
}
```


Project sponsor may accept or reject additional funds request, if he has *change_additional_budget* permission to change status of the request. He needs to address PATCH request to */projects/project-charter/budget/additional-budget/example-request/update-status/* by defining decision in request body.

Request body


```
{
    "status": "approved"
}
```


The endpoint returns additional fund request instance with details after updating its status.

Request body


```
{
  "id": 7,
  "budget": 7,
  "amount": 2000.0,
  "date": "2022-06-20T19:25:24.768122Z",
  "status": "approved"
}
```

Testing

What is Testing

Testing is the process of determining how well something performs. Software testing is a method for assessing whether produced software product meets the expected requirements and ensuring that it doesn't have defect.

Software testing may be performed using a couple of types of testing, for instance, system testing, unit testing or integration testing which each of them have various purposes. So, testing of thesis software is carried out using integration testing.

What is Integration Testing

Integration testing (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group [5].

The target is to test API endpoints that they work as a group without errors. API testing covers testing the APIs directly and as part of integration testing to see if they meet functionality, reliability, performance, and security requirements.

Testing of Backend API's

API tests are realized by using APITestCase module of DRF. Moreover, tests are accomplished with 100% test coverage. A test case example is described below.

/projects/tests.py

```
def test_post_create_project(self):
        """
            Ensure we can create a project as Project Management Office
        """
        user = self.generate_pmo_user_data()
        reg_response = self.post_user_registration(user)
        self.define_user_role(reg_response, 'PMO')
        response = self.post_user_login(user)
        assert response.status_code == status.HTTP_200_OK
```

```
# Authorized Project Management Office can create a new project
        author = response.data['user']['id']
        auth_token = response.data['auth_token']
        project = self.generate_project_data(author=author)
        response = self.post_create_project(project=project)
        assert response.status_code == status.HTTP_401_UNAUTHORIZED

        response = self.post_create_project(project=project, token=auth_token)
        assert response.status_code == status.HTTP_201_CREATED
        assert response.data.get('project').get('project_name') == project.get('project_name')
```

Figure 1. Test case for project creation

The test case checks that register, login and create project API's works as expected together.

Conclusion

The goal of this thesis was to demonstrate a platform that enables digital project management. The digital environment was developed to provide as much as sufficient API endpoints to cover project management, mainly project finance. Moreover, Integration tests has been written to ensure that API endpoints work without problem as a group. A Jupyter notebook document has also been developed to have a clear represent with table, graphs as frontend side of the platform.

References

[1] K. R. Chandrashekar. Object-Oriented Project Management. *Project Management Institute, 2010*.

[2] Object-Oriented Management. https://en.wikipedia.org/wiki/Object_Oriented_Management

[3] Object-Oriented Programming. https://en.wikipedia.org/wiki/Object-oriented_programming

[4] Django Documentation. Django apps. https://docs.djangoproject.com/en/4.0/intro/reusable-apps

[5] Wikipedia. Integration Testing. https://en.wikipedia.org/wiki/Integration_testing