

**Програмиране в мрежова среда**

**Курсов проект**

Изготвил:

Илия Веселинов Чакъров, СИ, МАГ 1к, 24651225

1. **Задание на проекта и описание**

Да се създаде контейнеризирана RESTful API услуга, използвайки Quarkus framework, написана на Java или Kotlin. Да се използва релационна БД, като за връзка с нея се използват Hibernate и Panache. Услугата да използва service, repository и други шаблони. Да се използва и DTO шаблон, за да не достигат идентификаторите от базата данни до клиента. Да се използва SwaggerUI за визуализация и JavaDoc за описание на крайните точки в него. Допълнително да има JavaDoc за всеки клас и метод. Приложението да съдържа тестове, които да покриват 35% от кода.

1. **Разглеждане на заданието**

Quarkus е **модерен** framework, оптимизиран за контейнеризирани приложения и облачни среди (Cloud Native). Той е особено подходящ за микросървиси и работи добре с GraalVM за компилация в native executables.

Използването на Hibernate (JPA) с Panache улеснява работата с релационни бази данни и намалява boilerplate кода. Panache предлага удобен начин за работа с entity класове и repository pattern.

Проектът следва Service, Repository и DTO **шаблони**, което осигурява:

* По-добра структурираност и разпределение на отговорностите.
* Изолация между слоевете на приложението.
* По-лесна поддръжка и тестване.

Добри практики за документация и визуализация:

* **SwaggerUI** – модерен начин за представяне на API-то, позволява тестване директно през браузър.
* **JavaDoc** – осигурява детайлна документация за разработчиците.

35% тестове е добър минимум за осигуряване на стабилност на кода. В днешно време се използват **JUnit, Quarkus Test,** Mockito за unit и integration тестове. Като най-удобна алтернатива за съвместимост с Quarkus-базирана API услуга е Quarkus Test.

Проектът ще бъде оптимизиран за работа в **Docker**/Kubernetes, което е стандарт за съвременните приложения. За база данни е избрана PostgreSQL.

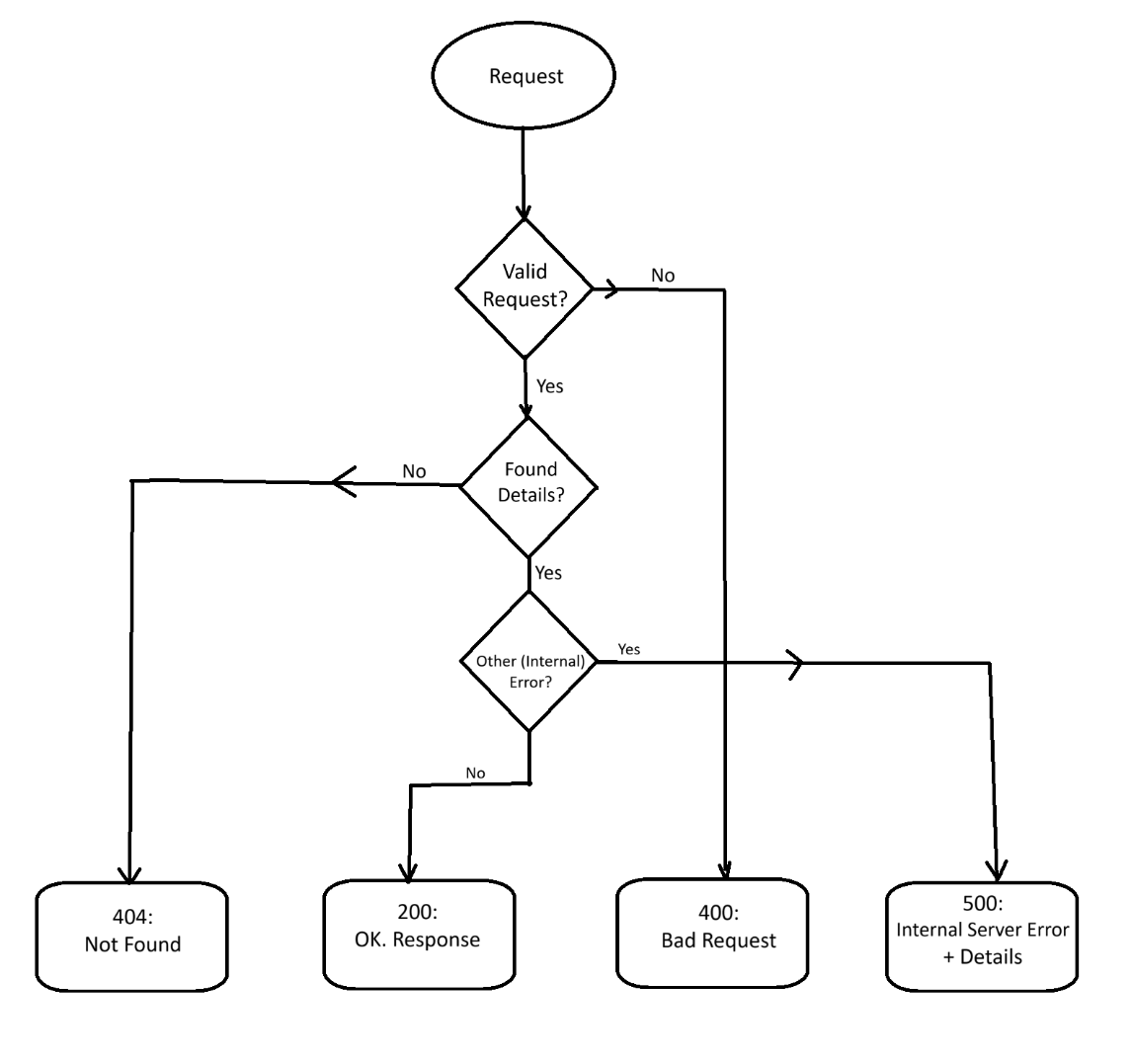
Това е напълно **актуален проект**, който използва модерни технологии и добри практики за разработка на RESTful API.

1. **Резлизация**
   1. Описание на приложението

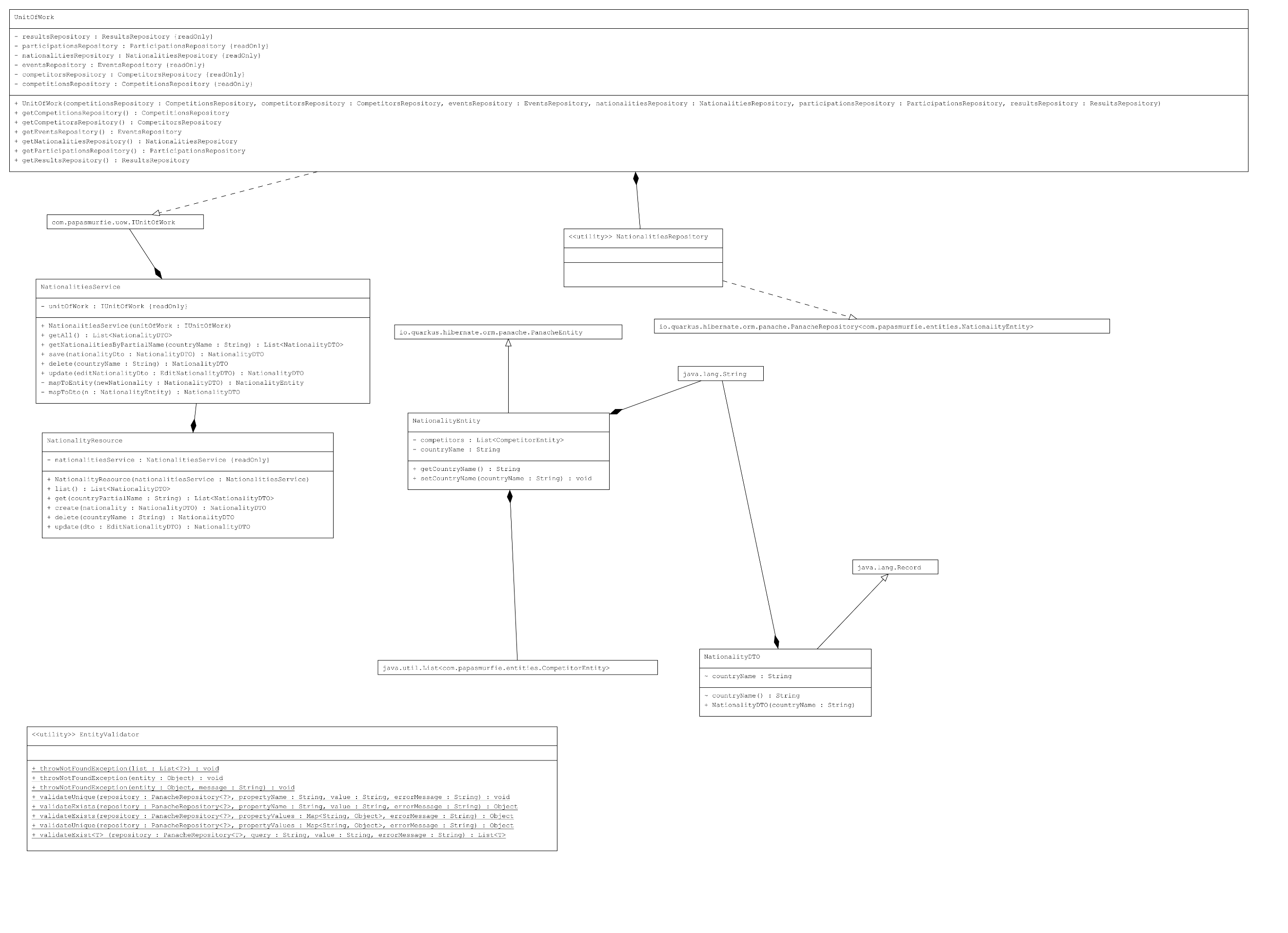
За реализация е избрана темата: **„Спортен рекордьор“**. Изработеното Web API записва състезатели, техните националости, състезания с дата напровеждане и пази записи за участия в съответното състезание, класиранията на участниците, както и постиженията. Услугата може да филтрира по множество различни критерии, например по дата на състезание, по дата от-до, по класиране, по резултат, по имена на участниците, по националност и други. Записите също така могат да бъдат редактирани и изтривани. Сорс кодът може да бъде разгледан тук: <https://github.com/OGSmurfen/PNU_Project>

* 1. Последователност

**Flowchart** диаграмата по-долу показва потока на данни и взаимодействието между компонентите в една типична архитектура на уеб микросървис. Диаграмата включва различни слоеве и процеси, които се случват, когато потребителят изпраща заявка и получава отговор. Отговорът независимо дали успешен или не, връща съответен код, както и json тяло с описание на проблема или ако е успешно – с исканите данни или с резултат от изпълнена процедура или други.



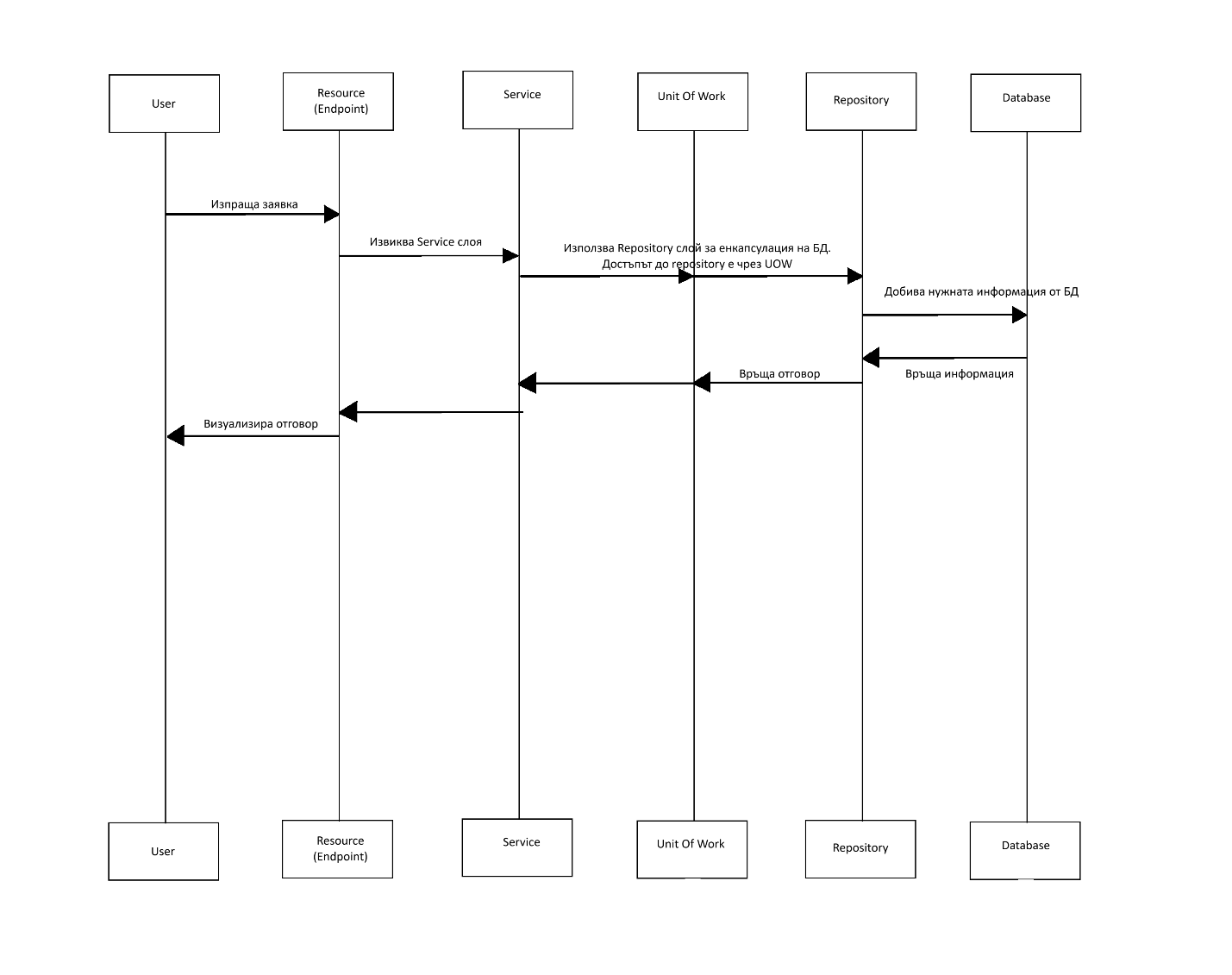
* 1. Връзки между обектите

С цел по-лесно проследяване в **UML клас диаграмата** е включена логиката на Nationality. Представени са класове NationalityResource, NationalityEntity, NationalityDTO, EditNationalityDto, NationalityRepository, Unit Of Work. Включен е и клас EntityValidator.

Ресурсът е крайната точка, тя взаимодейства със сървисът, който в себе си съдържа инстанция на Unit of Work, инжектирана от DI контейнера и предоставя инстанции на repository, също инжектирани от DI контейнера. Тези репозитори от своя страна заимодействат с БД и представляват готова имплементация от Panache.

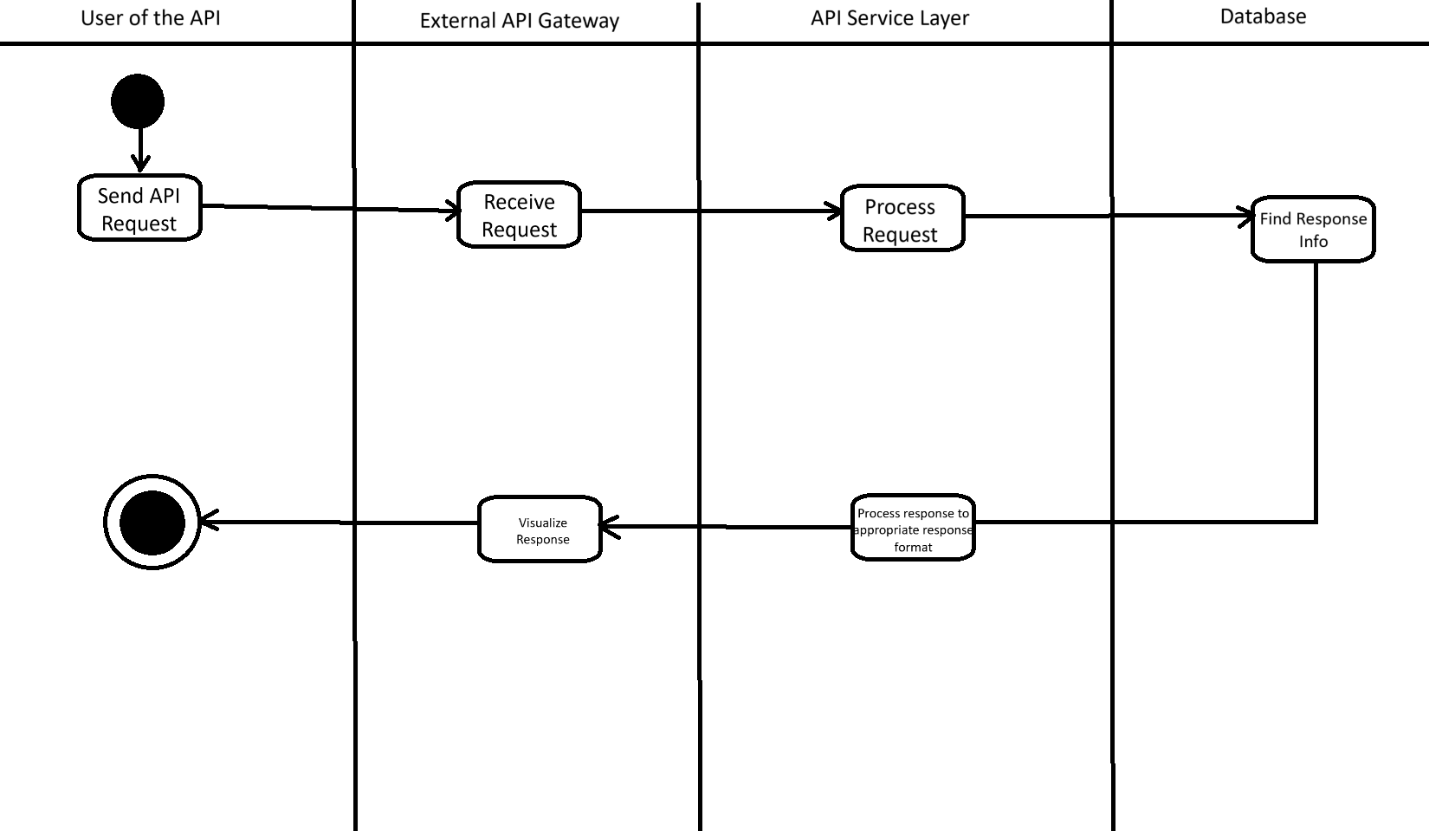
* 1. Взаимодействия

От всяка крайна точка се изпраща заявка от някой от следните типове: GET, POST, UPDATE, DELETE, PUT. Заради това, че е имплементирано RESTful API е нужно да се спази принципът за промяна и изтриване на запис да са в отделни заявки: DELETE и PUT. След подаване на информацията от входната точка към базата данни в зависимост от типа заявка и дали е изпълнена успешно, се очакват различни отговори в тялото на json-отговора. Опростено може да се проследи следващата **Sequence** диаграма:



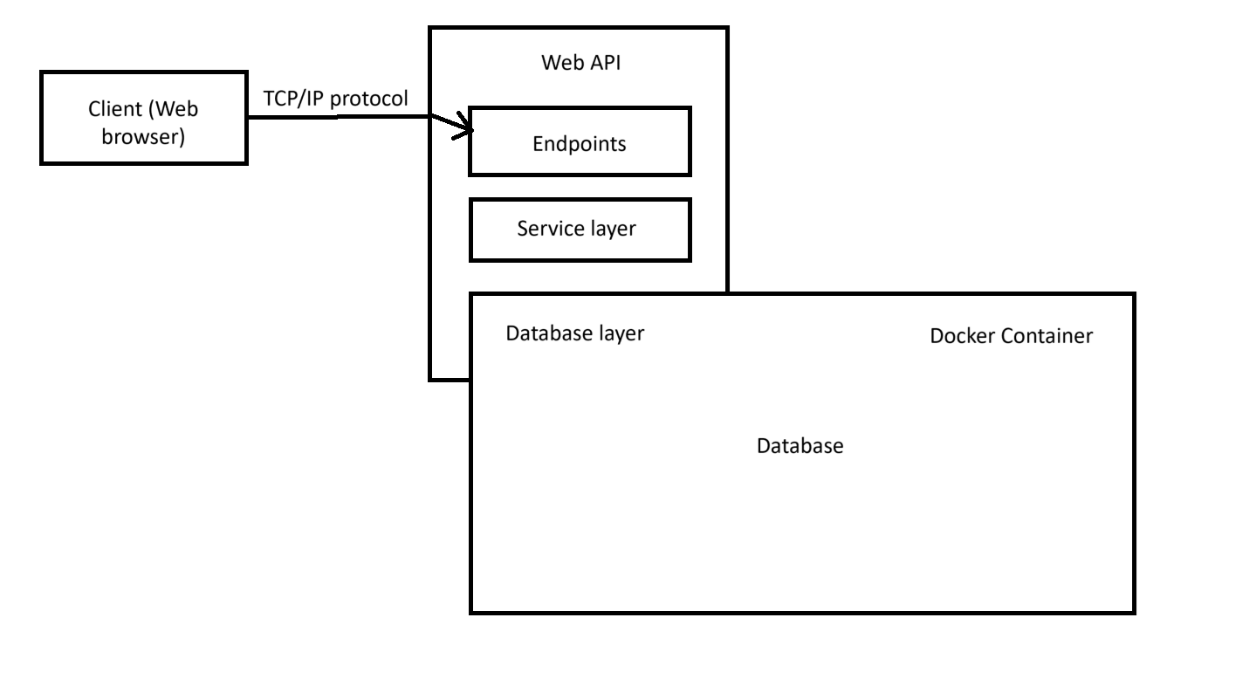
* 1. Процес на работа

Клиентът изпраща заявка към Endpoint (API Gateway), който маршрутизира заявката към API Service Layer, където се обработва бизнес логиката и чрез Repository Layer се осъществява достъп до Database Layer за извличане или запис на данни. След това резултатът се връща обратно по същия път – от база данни към Service Layer, през API Gateway и обратно към клиента като отговор. Всички repository са включени и в unit of work layer, който прави извикването им по-централизирано, усъществявайки по-сложни заявки като тези на Participation endpoint-а, който изисква да направи сложен запис, първо проверявайки информацията в няколко различни таблици от БД, после записвайки в няколко от тях. Затова и други подобни случаи този unit of work се явява полезен. За опростяване в **Activity** диаграмата service, unit of work и repository, са представени като един слой – API Service Layer.



* 1. Експлоатация на продукта

Самото уеб API е пуснато на хост машината, а Postgres базата данни е контейнеризирана в Docker. **Deployment** диаграма:



1. Източници:

<https://bump.sh/blog/api-architecture-diagrams>

<https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/>

<https://www.geeksforgeeks.org/deployment-diagram-unified-modeling-languageuml/>

<https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/>

<https://www.geeksforgeeks.org/unified-modeling-language-uml-activity-diagrams/>

<https://play.google.com/console/about/guides/releasewithconfidence/>

<https://play.google.com/console/u/0/developers/>

<https://support.google.com/googleplay/android-developer/answer/9859152?hl=en>