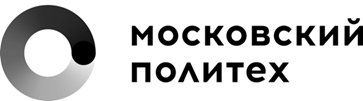
Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

(МОСКОВСКИЙ ПОЛИТЕХ)



ЛАБОРАТОРНАЯ РАБОТА 7

**ПРОЕКТИРОВАНИЕ API. ФРЕЙМВОРК gRPC.**

По курсу

**Проектирование Веб-сервисов**

ВыполнилКилеев С.И**.**

**студент группы 211-321**

Проверил

Пардаев А.А.

Москва, 2024

**Лабораторная работа 7**

**Цель работы:** Получить представление об основных концепциях gRPC и протокола Protobuf.

**Задачи:**

Индивидуальная работа

1. Перечислить аспекты, на основании которых делается выбор в пользу применения REST API или gRPC API.  
2. Разработать файл протокола Protobuf по выбранной функции из лабораторной работы 4.  
3. Создать спецификацию gRPC API в формате Protobuf для авторизации и аутентификации пользователей для сервиса из лабораторной работы 4.. Определить методы для регистрации нового пользователя и входа в систему.  
4. \*(необязательно) Реализовать спроектированный gRPC API на любом языке и фреймворке.

**Отчёт по выполнению:**

**Производительность и эффективность**

* **gRPC**: Использует бинарный протокол (Protocol Buffers), что обеспечивает более высокую производительность и меньший объем передаваемых данных.
* **REST**: Обычно использует текстовые форматы (JSON, XML), что может быть менее эффективным по сравнению с бинарными.

**Совместимость и поддержка языков**

* **gRPC**: Имеет широкую поддержку различных языков программирования через Protocol Buffers, облегчая межъязыковую интеграцию.
* **REST**: Поддерживается практически всеми языками, но может требовать дополнительных усилий для обеспечения совместимости.

**Сложность разработки и обучения**

* **REST**: Более прост и интуитивен, что снижает кривую обучения для новых разработчиков.
* **gRPC**: Требует знания Protocol Buffers и специфических инструментов, что может увеличить время обучения.

**Поддержка стриминга и реального времени**

* **gRPC**: Отлично подходит для приложений, требующих двунаправленного стриминга и обмена данными в реальном времени.
* **REST**: Ограничен моделью запрос-ответ и не поддерживает стриминг нативно.

**Управление версиями API и совместимость**

* **gRPC**: Позволяет более гибко управлять версиями API через Protocol Buffers, обеспечивая обратную совместимость.
* **REST**: Часто требует явного управления версиями API, что может усложнять поддержку.

**Безопасность**

* **Оба подхода**: Могут использовать стандартные механизмы безопасности, такие как TLS/SSL. Реализация специфических требований может варьироваться в зависимости от используемого подхода.

**Масштабируемость и распределенные системы**

* **gRPC**: Часто предпочтительнее в микросервисных архитектурах благодаря высокой производительности и эффективному использованию ресурсов.
* **REST**: Тоже подходит для масштабируемых систем, но может столкнуться с ограничениями по производительности при высоких нагрузках.

**Поддержка мобильных и веб-клиентов**

* **REST**: Обычно проще для интеграции с веб-браузерами и мобильными приложениями, благодаря широкому распространению и поддержке.
* **gRPC**: Поддерживает мобильные платформы, но может требовать дополнительных усилий для реализации на стороне клиента.

**Отладка и мониторинг**

* **REST**: Легче отлаживать и мониторить с помощью стандартных HTTP-инструментов и прокси.
* **gRPC**: Отладка может быть более сложной из-за использования бинарных протоколов, требующих специализированных инструментов.

**Кэширование**

* **REST**: Поддерживает стандартные механизмы кэширования HTTP, что может улучшить производительность.
* **gRPC**: Кэширование не поддерживается нативно и требует дополнительной реализации.

Функция: Назначать и управлять правами других модераторов

Функциональные требования:

1. Добавить нового модератора: на вход подается пользователь, а на выходе обновляем статус пользователя до модератора

2. Удалить модератора: на вход подается пользователь-модератор, а на выходе обновляем статус пользователя до участника

2.

syntax = "proto3";

package api;

option go\_package = "./";

service Moderator {

rpc PromoteToModerator(ModeratorRequest) returns (ModeratorResponse) {}

rpc DemoteToUser(ModeratorRequest) returns (ModeratorResponse) {}

}

message ModeratorRequest {

int32 participantId = 1;

}

message ModeratorResponse {

uint32 id = 1;

string name = 2;

string email = 3;

string password = 4;

uint32 status = 5;

uint32 role = 6;

string avatar = 7;

}

3.

syntax = "proto3";

package api;

option go\_package = "./";

service Auth {

rpc Register (RegisterRequest) returns (RegisterResponse);

rpc Login (LoginRequest) returns (LoginResponse);

}

message RegisterRequest {

string email = 1;

string password = 2; .

}

message RegisterResponse {

int64 user\_id = 1;

}

message LoginRequest {

string email = 1;

string password = 2;

int32 app\_id = 3;

}

message LoginResponse {

string token = 1;

}