Техническое задание

Представьте, что после изучения сложной темы и успешного выполнения всех заданий вы решили отдохнуть и провести вечер за просмотром фильма. Вкусная еда уже готовится, любимый плед уютно свернулся на кресле — а вы всё ещё не выбрали, что же посмотреть! Фильмов много — и с каждым годом становится всё больше. Чем их больше, тем больше разных оценок. Чем больше оценок, тем сложнее сделать выбор. Однако не время сдаваться! Вы напишете бэкенд для сервиса, который будет работать с фильмами и оценками пользователей, а также возвращать топ-5 фильмов, рекомендованных к просмотру. Теперь ни вам, ни вашим друзьям не придётся долго размышлять, что посмотреть вечером.
В этом спринте вы начнёте с малого, но очень важного: создадите каркас Spring Boot приложения `Filmorate` (от англ. *film* — «фильм» и *rate* — «оценивать»). В дальнейшем сервис будет обогащаться новым функционалом и с каждым спринтом становиться лучше благодаря вашим знаниям о Java. Скорее вперёд!

Предварительная настройка проекта

В появившемся репозитории создайте ветку `controllers-films-users`. Разработку решения для первого спринта нужно вести в ней.
Создайте заготовку проекта с помощью Spring Initializr. Некоторые параметры вы найдёте в этой таблице, остальные заполните самостоятельно.

Ура! Проект сгенерирован. Теперь можно шаг за шагом реализовать приложение.

Модели данных

Создайте пакет `model`. Добавьте в него два класса — `Film` и `User`. Это классы — модели данных приложения.
У `model.Film` должны быть следующие свойства:

- целочисленный идентификатор — `id`;
- название — `name`;
- описание — `description`;
- дата релиза — `releaseDate`;
- продолжительность фильма — `duration`.

Свойства `model.User`:

- целочисленный идентификатор — `id`;
- электронная почта — `email`;
- логин пользователя — `login`;
- имя для отображения — `name`;
- дата рождения — `birthday`.

Подсказка: про аннотацию @Data
Используйте аннотацию `@Data` библиотеки Lombok — с ней будет меньше работы по созданию сущностей.

Хранение данных

Сейчас данные можно хранить в памяти приложения — так же, как вы поступили в случае с менеджером задач. Для этого используйте контроллер.
В следующих спринтах мы расскажем, как правильно хранить данные в долговременном хранилище, чтобы они не зависели от перезапуска приложения.

REST-контроллеры

Создайте два класса-контроллера. `FilmController` будет обслуживать фильмы, а `UserController` — пользователей. Убедитесь, что созданные контроллеры соответствуют правилам REST.
Добавьте в классы-контроллеры эндпоинты с подходящим типом запроса для каждого из случаев.
Для `FilmController`:

- добавление фильма;
- обновление фильма;
- получение всех фильмов.

Для `UserController`:

- создание пользователя;
- обновление пользователя;
- получение списка всех пользователей.

Эндпоинты для создания и обновления данных должны также вернуть созданную или изменённую сущность.

Подсказка: про аннотацию @RequestBody
Используйте аннотацию `@RequestBody`, чтобы создать объект из тела запроса на добавление или обновление сущности.

Валидация

Проверьте данные, которые приходят в запросе на добавление нового фильма или пользователя. Эти данные должны соответствовать определённым критериям.
Для `Film`:

- название не может быть пустым;
- максимальная длина описания — 200 символов;
- дата релиза — не раньше 28 декабря 1895 года;
- продолжительность фильма должна быть положительной.

Для `User`:

- электронная почта не может быть пустой и должна содержать символ `@`;
- логин не может быть пустым и содержать пробелы;
- имя для отображения может быть пустым — в таком случае будет использован логин;
- дата рождения не может быть в будущем.

Подсказка: как обработать ошибки
Для обработки ошибок валидации напишите новое исключение — например, `ValidationException`.

Логирование

Добавьте логирование для операций, которые изменяют сущности — добавляют и обновляют их. Также логируйте причины ошибок — например, если валидация не пройдена. Это считается хорошей практикой.

Подсказка: про логирование сообщений
Воспользуйтесь библиотекой `slf4j` для логирования и объявляйте логер для каждого класса — так будет сразу видно, где в коде выводится та или иная строка.

```
Скопировать кодJAVA
private final static Logger log = LoggerFactory.getLogger(Example.class);
```

Вы также можете применить аннотацию `@Slf4j` библиотеки Lombok, чтобы не создавать логер вручную.

Тестирование

Добавьте тесты для валидации. Убедитесь, что она работает на граничных условиях.

Подсказка: на что обратить внимание при тестировании
Проверьте, что валидация не пропускает пустые или неверно заполненные поля. Посмотрите, как контроллер реагирует на пустой запрос.

Проверьте себя

Так как у вашего API пока нет интерфейса, вы будете взаимодействовать с ним через веб-клиент. Мы подготовили набор тестовых данных — Postman коллекцию. С её помощью вы сможете протестировать ваше API: postman.json

Дополнительное задание*

А теперь необязательное задание для самых смелых! Валидация, которую мы предлагаем реализовать в основном задании, — базовая. Она не покрывает всех возможных ошибок. Например, всё ещё можно создать пользователя с такой электронной почтой: `это-неправильный?эмейл@`.
В Java есть инструменты для проверки корректности различных данных. С помощью аннотаций можно задать ограничения, которые будут проверяться автоматически. Для этого добавьте в описание сборки проекта следующую зависимость.

```
Скопировать код XML
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

Теперь вы можете применить аннотацию `@NotNull` к полю класса-модели для проверки на `null`, `@NotBlank` — для проверки на пустую строку, `@Email` — для проверки на соответствие формату электронного адреса. Полный список доступных аннотаций можно найти в документации.

💡 Чтобы Spring не только преобразовал тело запроса в соответствующий класс, но и проверил корректность переданных данных, вместе с аннотацией `@RequestBody` нужно использовать аннотацию `@Valid`.

```
Скопировать код JAVA
public createUser(@Valid @RequestBody User user)
```

Поздравляем: первый шаг навстречу уютным киновечерам сделан.
Интересного вам программирования!