# REST-API сервис для управления фильмотекой.

### Предисловие

Перечисленные ниже требования рекомендуется выполнять параллельно с изучением основного материала по Flask и NGINX. Например, вы изучили, как создавать модели - примените эти знания к проекту, создав необходимые модели, после чего создайте миграции к моделям и примените их.
По ходу изучения вы будете сталкиваться с тем, что полученных знаний недостаточно, чтобы применить их к проекту, но не волнуйтесь, очень скоро (по ходу изучения следующих пунктов) это изменится.

### Функциональные требования

1. Поиск фильмов должен давать результаты по частичному совпадению.
2. Для вывода результатов поиска необходимо использовать пагинацию (по-умолчанию - 10 результатов на одну страницу).
3. Операции с фильмами:
    a. добавлять может только авторизованный пользователь
    b. удалять может только авторизованный пользователь, который его добавил или администратор
    c. редактировать может только авторизованный пользователь, который его добавил или администратор
    d. запрашивать может кто угодно
4. Фильмы можно фильтровать по:
    a. жанрам
    b. диапазону годов выхода
    c. режиссёру
5. Фильмы можно сортировать по:
    a. рейтингу
    b. дате выхода
6. Атрибуты фильма:
    a. название
    b. жанры
    c. дата выхода
    d. режиссёр
    e. описание (необязательное для заполнения поле)
    f. рейтинг (0-10)
    g. постер
    h. пользователь, добавивший фильм
7. При удалении режиссёра фильм НЕ должен быть удален, вместо этого нужно установить director = ‘unknown’.
8. Загрузка данных в БД должна сопровождаться валидациями по здравому смыслу (например, год выхода - должен быть числом, а не строкой).

9. При ошибках нужно выдавать адекватные сообщения и коды ошибок, чтобы пользователь API мог понять, что является причиной ошибки.
10. Должна присутствовать авторизация (рекомендуется использовать Flask-Login).

## Требования к проекту

1. Основа приложения - Flask Framework.
2. Основной функционал должен быть покрыт тестами.
3. Обязательным пунктом является применение Docker Compose.
4. В качестве базы данных должен быть использован PostgreSQL.
5. База данных должна быть нормализована и находиться минимум в 3-й нормальной форме.
6. В качестве виртуального окружения предпочтительней использовать один из этих инструментов: pipenv / poetry / pyenv.
7. Запросы на ваш backend должны проксироваться через Nginx.
8. При разработке использовать любой статический анализатор кода на выбор: pylint / prospector / black и т. д.
9. В ходе разработки использовать git flow.
10. При написании кода должен соблюдаться code-style.
11. Проект должен сопровождаться логированием.
12. Проект должен быть документирован при помощи swagger.
13. Использование type hint приветствуется.