Разработать веб-приложение с использованием Express и Sequelize. БД – любая.

*Сущности, операции*

К сущностям из ЛР01 добавить: «Пользователь», «Роль» + придумать еще одну сущность из предметной области (например, Книга, Автор, Библиотека).

Приложение должно уметь добавлять, изменять, удалять сущности, а также читать конкретную сущность и все сущности.

Приложение должно уметь осуществлять поиск по сущностям. Реализовать как глобальный поиск (по всему), так и локальный (по конкретному типу сущностей).

*Сортировка, пагинация*

Чтение всех сущностей и поиск должно поддерживать сортировку и разбиение на порции.

Сортировка по возрастанию по primary key и разбиение на порции, например, по 10 записей должны работать по умолчанию.

Клиент может дополнительно указать по какому полю хочет отсортировать сущности и в каком порядке.

Клиент может дополнительно указать по сколько записей должно быть в порции.

Ответ на чтение всех сущностей и поиск помимо самих сущностей должен содержать номер порции и доступное количество порций.

*Аутентификация, авторизация*

Приложение должно поддерживать вход, регистрацию и проверку прав пользователей.

После входа пользователя необходимо помечать через cookie.

Роли – «Администратор», «Редактор», «Пользователь».

Пользователь может читать сущности.

Редактор может тоже что и Пользователь + создавать, добавлять, удалять сущности.

Администратор может все что и Редактор + удалять пользователей, назначать им роли, просматривать лог файлы, восстанавливать удаленные сущности из предметной области.

Пароли должны хранится в виде хэша. Для хэширования использовать bcrypt (https://www.npmjs.com/package/bcrypt).

*Статика*

Статические файлы должны отдаваться с помощью express.static и храниться в отдельной директории.

*Маршрутизация*

Все запросы (кроме статики) должны приходить на /api/*. Имеется ввиду, что для работы с сущностями вы построите API. Например, маршрут для чтения всех книг - /api/books, для чтения конкретной книги - /api/books/:id.

Запросы на чтение – GET, запросы на создание, изменение, удаление – POST.

Все запросы должны быть разделены между роутерами, например, роутер книг (операции с книгами), роутер авторов и т.д.

*Архитектура*

Все таблицы должны иметь столбцы createdAt, updatedAt, deletedAt.

Код приложения должен быть разбит по слоям (модели, сервисы, контроллеры, хелперы), а внутри слоев на блоки (сервис книг, сервис авторов и т.д.).

Каждый блок должен быть помещен в отдельный модуль.

Внутри блока все должно быть разбито на как можно меньшие функции (в идеале на одну функцию строго одна ответственность). Например, контроллер запрашивает у сервиса книги отсортированные по имени автора. У нас будет цепочка методов, которая этим занимается. Проверяем входные параметры – номер порции, поле и порядок сортировки (1 функция), запрашиваем данные у БД (2), формируем ответ контроллеру (3).

Информация для подключения к БД должна хранится в config.json.

*Логирование*

Все запросы к /api/* должны выводится на консоль + асинхронно записываться в JSON файлы. Если размер текущего лог-файла X Кб (должно задаваться в config.json), создается новый файл. Приложение не должно ожидать завершения записи в лог-файлы.

*Кэширование*

Запросы на чтение должны кэшироваться на время X (задается в config.json). Например, клиент запросил книги, отсортированные по году по убыванию. После того как отправили ответ клиенту, записываем результат в кэш на X секунд. Если за X секунд кто-то еще обращается с точно таким же запросом, мы отдаем ему ответ из кэша, не продлевая время нахождения данных в кэше.

*Обработка ошибок*

Если что-то пошло не так по вине клиента, сообщить понятной ошибкой. Например, запросили несуществующую книгу, вернуть ошибку 404 с сообщением. Ошибки возвращать в формате JSON { code: 'not_found', message: 'Entity not found', status: 404 }. Также написать middleware которое будет обрабатывать все непредвиденные ошибки и возвращать в таком случае 500 ошибку сервера.

*Время на выполнение:* 10 часов / 5 занятий