Финальный проект JavaRush университета

Модуль 5. Spring
19 уровень, 0 лекция

ОТКРЫТА

Этот проект будет отличаться от всех предыдущих, которые ты уже сделал на протяжении курса обучения. Его главное отличие – то, что это относительно большой, работающий проект.

Его не нужно писать с нуля, а нужно вносить изменения в код и ничего в нем не сломать, добавлять еще не написанный функционал, настраивать инфраструктуру. Именно с таким видом задач вы будете, в основном, сталкиваться на работе.

Это половина всей работы программиста – чтение и разбор чужого кода.

На работе вы будете все время тратить на 3 вещи:

- Разбор и чтение чужого кода – 50% времени.
- Представить как вашу задачу описать в виде кода-решения – 25% времени.
- Добавление вашего кода в существующую логику проекта – 25% времени.

Когда я начинаю знакомиться с новым проектом, я стараюсь придерживаться такой последовательности:

1. Узнаю какую задачу решает (или будет решать в будущем) проект.
2. Смотрю какие зависимости есть у проекта.
3. Смотрю на структуру БД.
4. Смотрю на структуру Entity.
5. Смотрю API и/или прочие контроллеры.
6. Вникаю какая задача у каждого сервиса.

Только после знакомства с данным списком можно смотреть «а что, собственно говоря, я должен сделать». Давай по этому плану и будем двигаться.

Основная задача

Название проекта – JiraRush.

GitHub проекта

Думаю, из названия можно догадаться, что это некая доска задач как Jira или Trello. Проект нужен для отслеживания какой-либо активности. Это может быть как ведение проекта, так и список покупок от жены к мужу. **Клонируй проект себе**, и давай смотреть что в нем есть.

Если не знаком с Jira, вот короткое ознакомительное видео по теме

Технологии

- Spring Boot
- Spring JPA
- Hibernate
- PostgreSQL
- Liquibase (система управления версиями БД, в основном, ее структурой)
- Spring Security
- Spring MVC
- Thymeleaf
- jQuery
- Swagger (документирование API)
- Caffeine (кэш)
- Lombok

- Mapstruct (мапперы для преобразования между entity & DTO)
- Spring Test
- JUnit
- Docker [1], [2]

Структура БД

Как запустить приложение:

1. Клонировать себе на машину проект.
2. Запустить локально сервер БД (PostgreSQL). Рекомендую это делать через docker.

   Сделаем 2 контейнера с БД. Первый для запуска приложения (профиль prod), второй для билда и тестов (профиль test). Соответствующие команды:

   ```
   docker run -p 5432:5432 --name postgres-db -e POSTGRES_USER=jira -e

   docker run -p 5433:5432 --name postgres-db-test -e POSTGRES_USER=j
   ```

3. Сбилдить приложение: `mvn clean install`
4. Запустить Spring Boot приложение (JiraRushApplication) с профилем `prod`

Во время билда приложения произойдет популяция тестовой БД. Во время запуска приложения произойдет популяция БД для работы. Если точнее – накатится структура и словари. Чтоб «посмотреть» как работает приложение нужно выполнить скрипт `data.sql` из `resources/data4dev`.

Теперь можем посмотреть структуру БД:

`activity` – таблица активностей, в которой хранятся действия пользователей относительно других сущностей. Например «пользователь написал комментарий», «пользователь изменил статус задачи» …

`attachment` – таблица для хранения информации о прикреплённых файлах.

`contact` – словарь «пользователь – тип контакта – значение контакта».

`mail_case` – таблица куда сохраняются данные, если не удалось отправить письмо (email).

`profile` – профиль пользователя. Настройки хранятся в поле `mail_notifications` в виде битовых флагов.

`project` – Проекты. Самая крупная логическая единица доски задач. Например, если у компании есть микросервисы, каждый микросервис ведет команда, тогда для такого микросервиса будет создан проект.

`reference` – таблица разных сущностей по типам. `ref_type` соответствует **Enum** `com.javarush.jira.ref.RefType`. Колонка **aux** – значение для вычисления битовых флагов.

`task` – задачи. Самый базовый логический структурный элемент доски задач JiraRush.

`task_tag` – словарь связей «задача– тег». У одной задачи может быть ноль или более тегов.

`user_belong` – связь пользователя с сущностями типа «задача», «спринт», «проект».

`user_role` – словарь связей «пользователь – роль». У одного пользователя может быть несколько ролей.

`users` список зарегистрированных пользователей. Колонки `endpoint` и `startpoint` отвечают за то, активен ли этот пользователь, или был активен, но сейчас уже нет. В поле `password` в начале пароля в виде «{…}» хранится метод шифрования (Spring Security, помнишь?).

Структура Entity

Этот пункт я оставлю на самостоятельное ознакомление

Смотрю API и/или прочие контроллеры

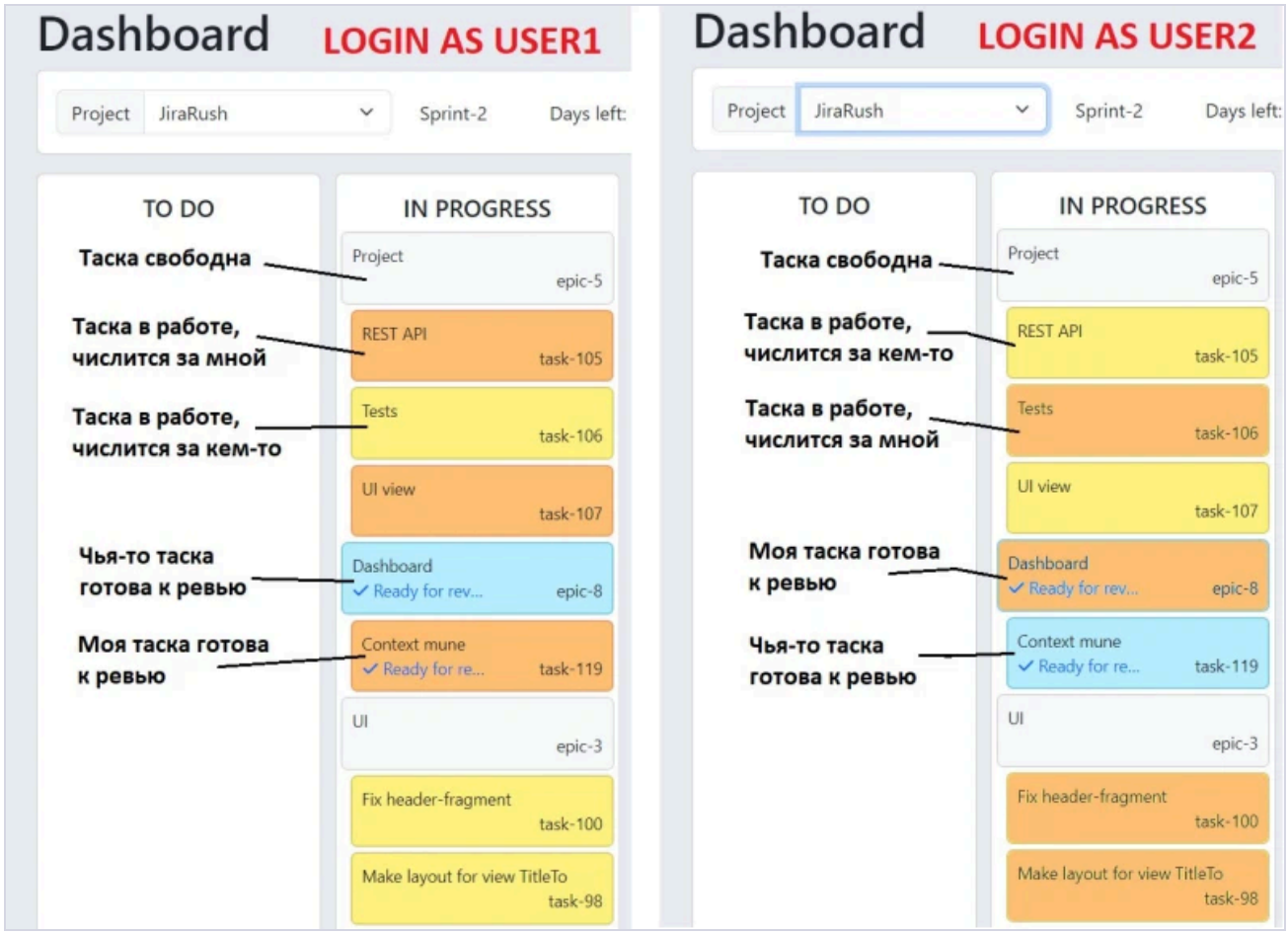
В проект подключен Swagger, который отвечает за документирование API. Для того чтоб посмотреть – при запущенном приложении зайди на http://localhost:8080/swagger-ui/index.html

Здесь можно посмотреть какие есть контроллеры, какие методы у этих контроллеров, какие ожидаются данные на вход, типы методов и т.д. Также можно отправлять самому запрос. Не лишним будет авторизоваться от разных пользователей (тестовые данные написаны на UI-е) и смотреть какие права нужны на выполнение какого метода.

За что отвечает какой сервис

С этим пунктом тебе также придётся знакомиться самостоятельно. Вместо этого рассмотрим уже реализованный функционал. Смотреть будем от пользователя с ролью **ADMIN**, так как у него наиболее широкие полномочия. Но тебе нужно будет посмотреть и от **MANAGER**, и от **DEV**.

- Дерево проектов. Есть список проектов. Админ или Менеджер (смотри энам Role) может заниматься добавлением новых проектов и подпроектов. У проекта есть беклог (список задач (таск), которые нужно сделать), и спринты. Также у проекта может быть подпроект. Задача может относиться к спринту непосредственно, или через эпик (некая большая группа задач). Смотри http://localhost:8080/view/tree
- Доска задач (дешборд). Можно выбрать по какому проекту смотреть дешборд. По проекту будет показан активный (или последний) спринт. Задачи из спринта разбиты по колонкам. Цветом выделены те, на которые нужно обратить внимание. Изменение статусов задач делается через райт-клик по задаче. Смотри http://localhost:8080/ui/dashboard



- Отчет по спринту: минимальный набор статистик по завершенным спринтам с возможностью выбора проекта и спринта. Смотри http://localhost:8080/ui/reports

- Список пользователей. Список всех зарегистрированных пользователей с возможностью поиска, сортировки, добавления, редактирования и удаления. Страница http://localhost:8080/view/admin/users
- Ссылки - функционал настройки JiraRush (какие поля доступны какому виду сущностей. Например, какие соц. сети показывать в контакте пользователя, или какие есть приоритеты у задачи). Смотри http://localhost:8080/ui/admin/references
- Редактирование профиля пользователя (своего профиля).
- Регистрация нового пользователя через `email + password`.
- Регистрация нового пользователя через социальные сети (Google, GitLab, GitHub).
- Отправка писем на почту при сбросе пароля.
- Отправка писем на почту для ее подтверждения.
- Создание новой задачи в беклоге (с возможностью прикреплять файлы как вложения).

Еще некая информация о проекте «из уст в уста»

Структура проекта сделана на подобии к проекту **Spring Modulith**. То есть, есть пакеты `internal`, к которым не может обращаться код извне пакета верхнего уровня.

Это сделано с «заделом на будущее», чтоб можно было легко поделить сервер на микросервисы. По крайней мере, такая была задумка архитектора, а реализация может отличаться (да, так бывает в реальных проектах почти всегда).

Статические файлы для фронта раздаются через **nginx** сервер. По крайней мере так должно быть на «проде», но при локальном запуске все проще.

PostgreSQL. Ты не будешь писать нативные запросы. Нужно будет использовать запросы на JPQL. Из нового для тебя будет только подключение к PostgreSQL, чтобы смотреть структуру таблицы и их данные. Рекомендую использовать вкладку **database IDEA**.

Liquibase — это открытая независимая от БД библиотека для отслеживания, управления и применения изменений схемы базы данных. В проекте он используется для автоматического выполнения скрипта инициализации базы данных и заполнения ее тестовыми данными. Если ты меняешь структуру таблиц, нужно удалить 2 служебные таблицы Liquibase (`databasechangelog` и `databasechangeloglock`). После этого при следующем запуске приложения, Liquibase выполнит заново скрипт инициализации бд. Тебе никакие настройки делать не нужно, все работает из коробки.

Swagger — позволяет через UI визуально смотреть какие методы есть в контролерах приложения и выполнять к ним тестовые запросы. Задач по Swagger не будет. Он

добавлен для удобства работы над проектом.

Caffeine cache — это высокопроизводительная библиотека кеша для Java. Рекомендую посмотреть как он подключается. Заданий на использование не будет.

Твои задачи на выбор

Начни с легких задач и только потом приступай к остальным. Постарайся выполнить как можно больше задач.

1. Разобраться со структурой проекта (onboarding).
2. Удалить социальные сети: vk, yandex. `Easy task`
3. Вынести чувствительную информацию в отдельный проперти файл:
   - логин
   - пароль БД
   - идентификаторы для OAuth регистрации/авторизации
   - настройки почты

   Значения этих проперти должны считываться при старте сервера из переменных окружения машины. `Easy task`
4. Переделать тесты так, чтоб во время тестов использовалась **in memory БД (H2)**, а не PostgreSQL. Для этого нужно определить 2 бина, и выборка какой из них использовать должно определяться активным профилем Spring. H2 не поддерживает все фичи, которые есть у PostgreSQL, поэтому тебе прийдется немного упростить скрипты с тестовыми данными.
5. Написать тесты для всех публичных методов контроллера `ProfileRestController`. Хоть методов только 2, но тестовых методов должно быть больше, т.к. нужно проверить success and unsuccess path.
6. Сделать рефакторинг метода `com.javarush.jira.bugtracking.attachment.FileUtil#upload` чтоб он использовал современный подход для работы с файловой системмой. `Easy task`
7. Добавить новый функционал: добавления тегов к задаче (REST API + реализация на сервисе). Фронт делать необязательно. Таблица `task_tag` уже создана.
8. Добавить подсчет времени сколько задача находилась в работе и тестировании. Написать 2 метода на уровне сервиса, которые параметром принимают задачу и возвращают затраченное время:
   - Сколько задача находилась в работе (ready_for_review минус in_progress).
   - Сколько задача находилась на тестировании (done минус ready_for_review).

Для написания этого задания, нужно добавить в конец скрипта инициализации базы данных `changelog.sql` **3 записи** в таблицу `ACTIVITY`

```
insert into ACTIVITY ( ID, AUTHOR_ID, TASK_ID, UPDATED, STATUS_CODE
```

Со статусами:

- время начала работы над задачей – **in_progress**
- время окончания разработки - **ready_for_review**
- время конца тестирования - **done**

9. Написать `Dockerfile` для основного сервера
10. Написать `docker-compose` файл для запуска контейнера сервера вместе с БД и nginx. Для nginx используй конфиг-файл `config/nginx.conf`. При необходимости файл конфига можно редактировать. `Hard task`
11. Добавить локализацию минимум на двух языках для шаблонов писем (mails) и стартовой страницы `index.html`.
12. Переделать механизм распознавания «свой-чужой» между фронтом и беком с `JSESSIONID` на `JWT`. Из сложностей – тебе придётся переделать отправку форм с фронта, чтоб добавлять хедер аутентификации. `Extra-hard task`

При сдаче проекта в **README.md** напиши какие пункты задания ты выполнил.

---

1. Обзор финального проекта (Onboarding)
2. Подсказка по Docker Compose

‹ Предыдущая Следующая ›

Введите текст комментария

Мария Уровень 111 EXPERT 16 мая, 21:44

не получалось запустить через докер. оказалось нужно было зайти в менеджер задач > сервисы > остановить postgresql. занимал порт, так как был установлен ранее

Ответить 0

Павел Уровень 106 EXPERT 5 августа, 21:39

Я не стал создавать контейнер для jira db, просто в системном postgres создал базу. Если у кого-то были трудности с ошибкой

```
1 java.lang.NoSuchFieldError: Class com.sun.tools.javac.tree.JCTr
```

после команды **mvn clean install** попробуйте посмотреть на https://stackoverflow.com, а то Олександр как-то быстро проскочил этот момент.

Ответить 0

Жасаров Рысбек Уровень 111 EXPERT 13 мая, 16:08

ура, тоже долистался до этого)))

Ответить 0

Павел Уровень 106 EXPERT 28 июля, 21:26

Я вот листал-листал и в Security увидел заголовок лекции **Remember Me**. Подумал, это javarush университет напоминает, что материалы курса подходят к концу. Даже грустно стало.

Ответить 0

Антон Уровень 110 EXPERT 20 марта, 20:09

ура я долистался до этого

Ответить +1

Max Dudin Уровень 108 EXPERT 9 января, 12:14

Ура .. тут есть подсказки =))

Ответить +1

ОБУЧЕНИЕ

Курсы программирования

Курс Java

Помощь по задачам

Подписки

Задачи-игры

СООБЩЕСТВО

Пользователи

Статьи

Форум

Чат

Истории успеха

Активности

КОМПАНИЯ

О нас

Контакты

Отзывы

FAQ

Поддержка

RUSH

JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ