Лабораторная работа №3.

Тема:  Создание API. Простое серверное приложение.

Задание

Реализуйте функционал API.

1. Теория

**API** – набор команд, с помощью которых серверное приложение понимает, что вы хотите сделать. На практике реализуется набором функций.

**Spring** - один из самых популярных фреймворков для создания веб-приложений на Java. **Фреймворк** — это что-то похожее на библиотеку (возможно этот термин вам более знаком), но есть один момент. Грубо говоря, используя библиотеку, вы просто создаете объекты классов, которые в ней есть, вызываете нужные вам методы, и таким образом получаете нужный вам результат. То есть, тут более императивный подход: вы четко указываете в своей программе в какой конкретный момент надо создать какой объект, в какой момент вызвать конкретный метод, и тд. С фреймворками дела обстоят слегка иначе. Вы просто пишете какие-то свои классы, прописываете там какую-то часть логики, а создает объекты ваших классов и вызывает методы за вас уже сам фреймворк. Чаще всего, ваши классы имплементируют какие-то интерфейсы из фреймворка или наследуют какие-то классы из него, таким образом получая часть уже написанной за вас функциональности. Но не обязательно именно так. В спринге например стараются по максимуму отойти от такой жесткой связности (когда ваши классы напрямую зависят от каких-то классов/интерфейсов из этого фреймворка), и используют для этой цели аннотации.

*Важно понять*, что спринг — это просто набор каких-то классов и интерфейсов, которые уже написаны за вас.

**JSON** - один из наиболее удобных форматов для сериализации данных при взаимодействии различных систем между собой. Если нужно с сервера взять объект с данными и передать его клиенту, то в качестве промежуточного формата – для передачи по сети, почти всегда используют именно его.

**Запросы HTTP** - протокол HTTP описывает взаимодействие между двумя компьютерами (клиентом и сервером), построенное на базе сообщений, называемых запрос (Request) и ответ (Response). Тип HTTP-запроса (также называемый HTTP-метод) указывает серверу на то, какое действие мы хотим произвести с ресурсом. Для разграничения действий с ресурсами на уровне HTTP-методов были придуманы следующие варианты:

* GET — получение ресурса
* POST — создание ресурса
* PUT — обновление ресурса
* DELETE — удаление ресурса

Обратите внимание на тот факт, что спецификация HTTP не обязывает сервер понимать все методы (которых на самом деле гораздо больше, чем 4) — обязателен только GET, а также не указывает серверу, что он должен делать при получении запроса с тем или иным методом. А это значит, что сервер в ответ на запрос DELETE /index.html HTTP/1.1 *не обязан* удалять страницу index.html на сервере, так же как на запрос GET /index.html HTTP/1.1 *не обязан*возвращать вам страницу index.html, он может ее удалять, например.

**Подходы к созданию серверных приложений.**

Есть множество подходов к созданию архитектур приложений типа клиент-сервер (которую мы с вами и напишем по итогу лабораторных работ). Наиболее распространенная – многоуровневая архитектура (или многослойная). Заключается в том, что все приложение делится на слои – слой представления данных, слой обработки, слой хранения…

1. Создание приложения

Далее как пример будет описано создание одной функции API для просмотра записи оценки из таблицы оценок (*mark*, БД и сама таблица описаны во 2 лабораторной работе).

Реализуем многослойное представление – слой хранения данных (model), слой взаимодействия с БД (dao), слой контроллеров для взаимодействия с клиентским приложением(controller).

Создадим следующую структуру приложения (рис. 1).

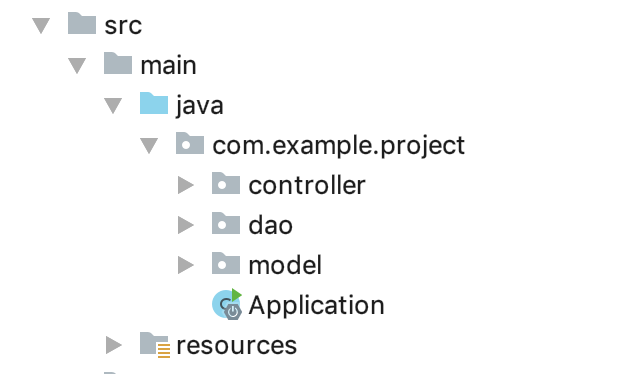


Рисунок 1.

Название пакета у вас может отличатся в зависимости от того, как вы создали шаблон ранее.

Создадим в папке model класс под хранение данных таблицы mark (рис. 2). Через этот класс мы будем взаимодействовать с таблицей  mark.

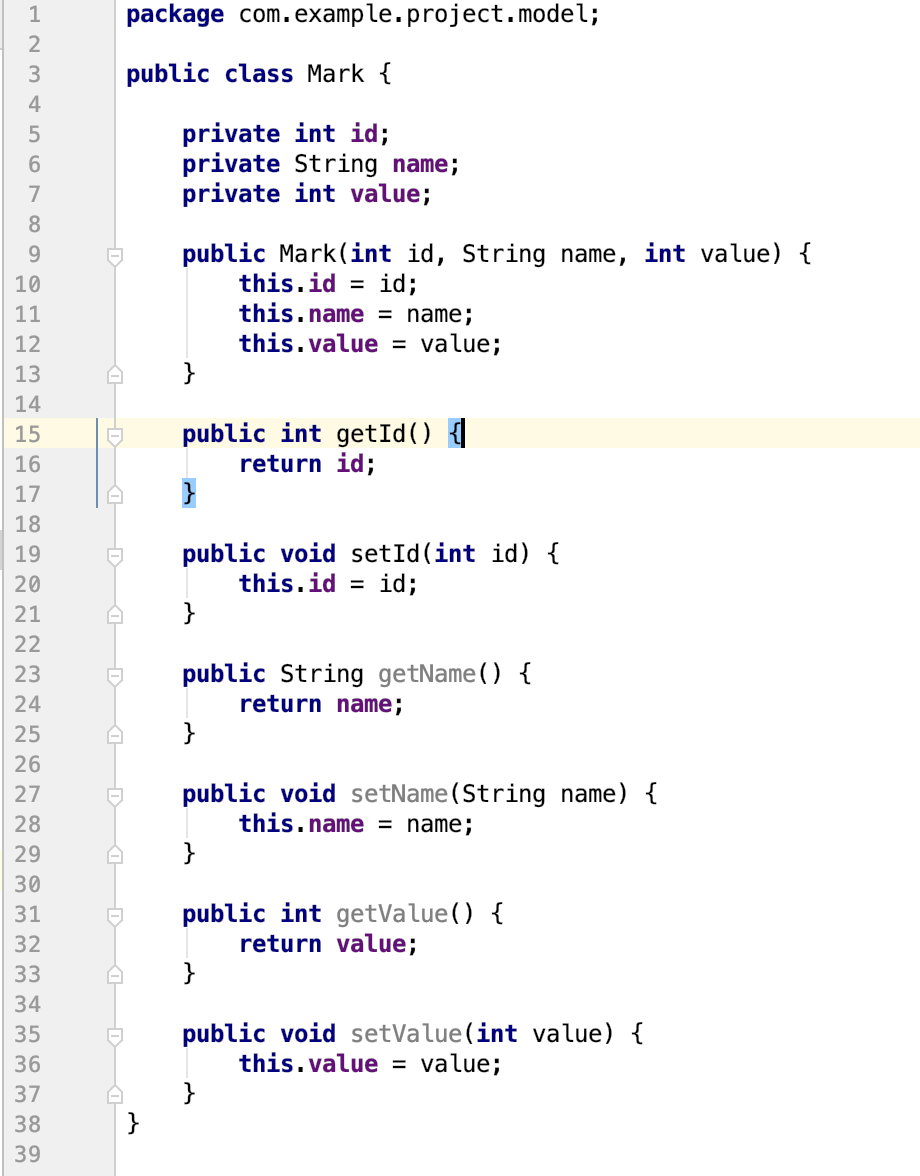


Рисунок 2.

Обратите внимание:

* Есть конструктор со всеми полями и так называемые методы getter/setter для получения/записи полей. Это необходимо для того, чтобы впоследствии фреймворк мог грамотно отправить объект вашего класса на клиент (об этом ниже).

Создадим класс слоя взаимодействия с БД. Для этого в папке dao создадим класс MarkJdbc (рис. 3).



Рисунок 3.

Обратите внимание:

* Используется спринговая аннотация на класс @Repository – таким образом мы «говорим» фреймворку, что этот класс отвечает за работу с данными.
* Используется спринговый класс JdbcTemplate – необходим для связи с вашей БД. При этом, сам объект класса JdbcTemplate *НЕ НУЖНО* создавать – за вас это сделает спринг. Вам лишь необходимо указать его в полях класса и конструкторе, а также правильно использовать его методы для получения данных(в данном примере используется метод queryForObject()).
* Метод mapMark нужен для того, чтобы программа понимала в какое поле класса Mark вы хотите записать данные из ответа на SQL-запрос.

Однако параметры для подключения к БД все-таки необходимо уточнить для фреймворка. Для этого переходим в папку resources, открываем файл application.properties и пишем необходимые строки (рис. 4).

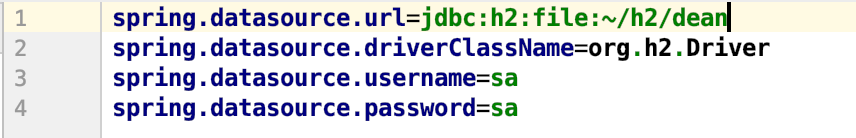


Рисунок 4.

Обратите внимание, что имя, пароль и путь вы указываете конкретно к своей БД (посмотреть параметры можно через вкладку программы databases – см. Лабораторную 1). Также необходимо задать имя БД для корректного подключения к ней при запуске приложения (происходит через ту же вкладку databases).

Создадим класс взаимодействия с клиентом. Для этого в папке controller создадим класс MarkController (рис. 5). Именно он связывает url адрес запроса с необходимым нам функционалом.

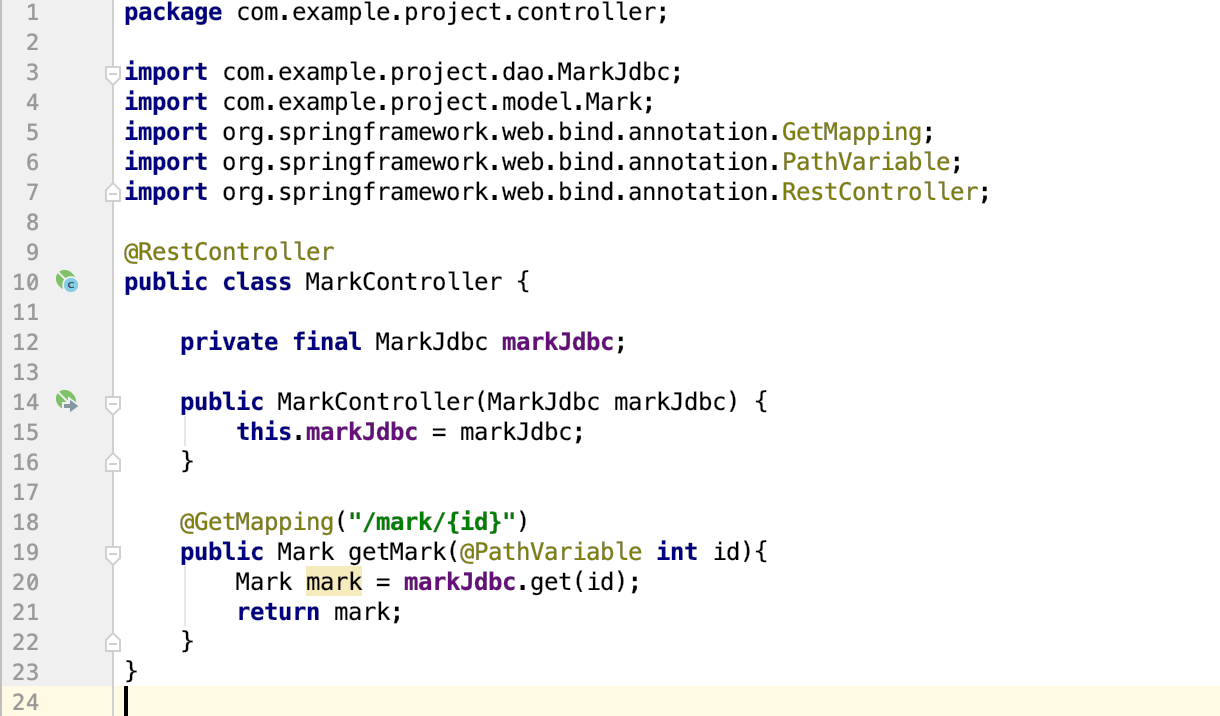


Рисунок 5.

Обратите внимание:

* Используется спринговая аннотация @RestController – так мы «говорим» фреймворку о предназначении класса.
* Используется спринговая аннотация @GetMapping – так мы «говорим» фреймворку о предназначении нашего метода, а именно:
  + Связать данную функцию с GET запросом по адресу /mark/{id оценки}
* Используется спринговая аннотация @PathVariable – так мы «говорим» фреймворку о том, чтобы он взял параметр id из строки запроса (имена должны совпадать!).

Запустим наше приложение. Для этого нажмем на кнопку запуска (рис. 6).

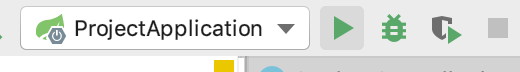


Рисунок 6.

После запуска приложения (рис. 7) переходим в браузер и вбиваем адрес <http://localhost:8080/mark/1> .

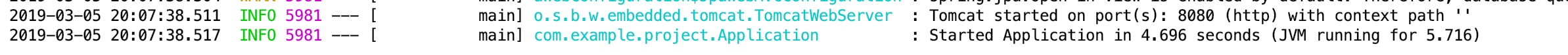


Рисунок 7.

.

По итогу мы получаем сообщение в формате JSON об отметки в БД под id = 1 (рис. 8).

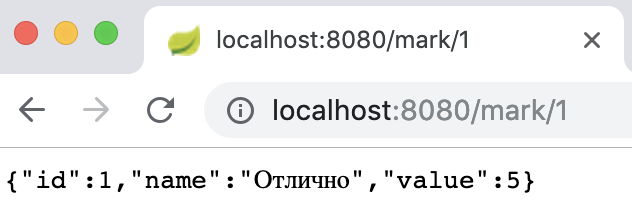


Рисунок 8.

**Что только что произошло?**

1. Выполнение запроса отсылает GET запрос “/mark/1”в наше приложение.
2. Этот запрос обрабатывается в объекте класса MarkController (который заблаговременно создал спринг), который вызывает метод объекта класса MarkJdbc (спринг заблаговременно создал объект MarkJdbc и передал его в поле объекта класса MarkConroller).
3. В объект класса MarkJdbc спринг передает в поле объект класса JdbcTemplate, который он заблаговременно создал и настроил через параметры из файла application.properties.
4. В объекте класса MarkJdbc происходит взаимодействие с БД, которое по итогу записывает ответ на SQL-запрос в объект класса Mark.
5. Этот объект передается в контроллер.
6. Контроллер пытается вернуть объект класса Mark, который спринг преобразует в JSON-строку и отсылает нам.
7. В итоге мы видим ответ на наш запрос в виде строки {"id":1,"name":"Отлично","value":5}.
8. Совет

Для проверки и более плотного взаимодействия с API рекомендуется установить программу Postman.

Скачать: <https://www.getpostman.com>

Пример того же самого запроса выше в постмане (рис. 9).

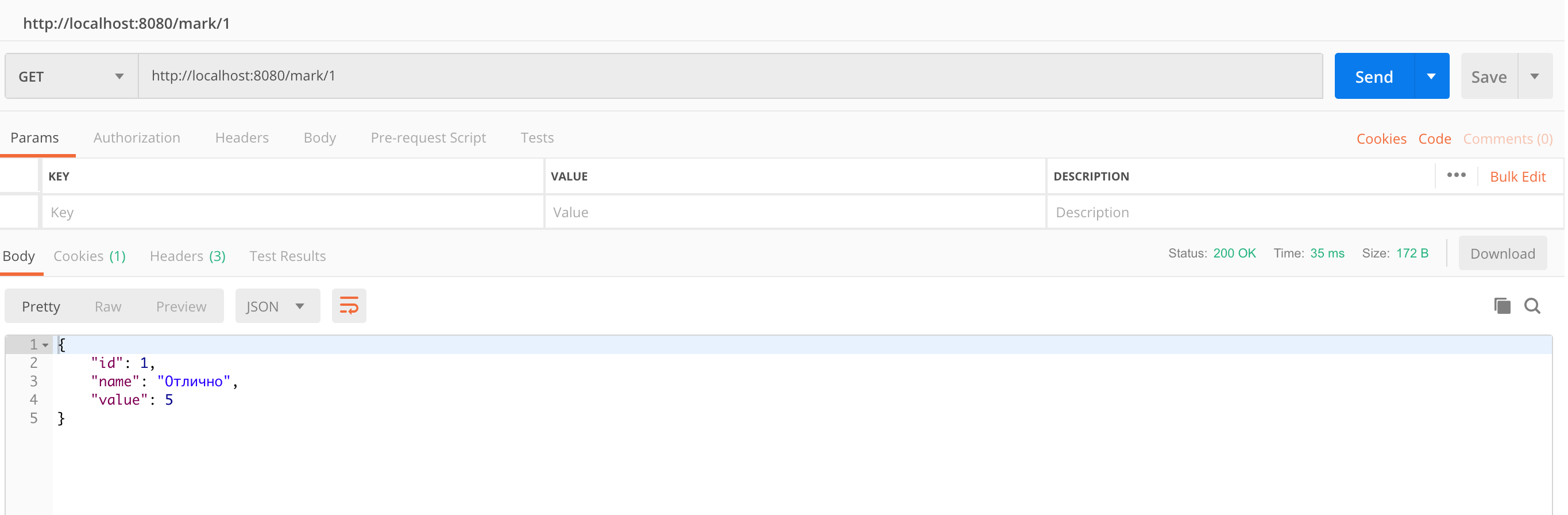


Рисунок 9.

1. Задание

Реализуйте следующий функционал:

* Создание студента
* Просмотр студента
* Просмотр всех студентов
* Просмотр студентов по группе
* Редактирование студента
* Удаление студента
* Просмотр группы
* Просмотр всех групп
* Редактирование группы
* Создание группы
* Удаление группы
* Просмотр записи журнала
* Просмотр записей журнала по студенту
* Просмотр записей журнала по группе
* Редактирование оценок в журнале