МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

Специализация 1-40 05 01 03 «Информационные системы и технологии (издательско-полиграфический комплекс)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Веб-приложение «Управление задачами и проектами»

Выполнил студент Процукович К.М

(Ф.И.О.)

Руководитель проекта асс. Денисова А.М.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Денисова А.М.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Денисова А.М.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2019

Оглавление

[Введение 4](#_Toc26772706)

[1. Постановка задачи 5](#_Toc26772707)

[1.1. Обзор аналогов 5](#_Toc26772708)

[1.2. Актуальность задачи 9](#_Toc26772709)

[1.3. Выбор инструментов разработки 9](#_Toc26772710)

[1.4. Постановка целей 11](#_Toc26772711)

[2. Разработка 13](#_Toc26772712)

[2.1. Разработка модели базы данных 13](#_Toc26772713)

[2.2. Разработка архитектуры проекта 15](#_Toc26772714)

[2.3. Структура проекта 15](#_Toc26772715)

[2.3.1. Backend 16](#_Toc26772716)

[2.3.2. FrontendAPI 21](#_Toc26772717)

[2.3.3. Frontend 21](#_Toc26772718)

[2.4. Разработка таблиц БД 22](#_Toc26772719)

[3. Руководство пользователя 24](#_Toc26772720)

[Заключение 29](#_Toc26772721)

[Список используемых источников 30](#_Toc26772722)

[Приложения А. Листинг создания БД и таблиц. 31](#_Toc26772723)

[Приложения Б. Листинг Микросервиса Backend 32](#_Toc26772724)

[Приложения В. Листинг Микросервиса FAPI 38](#_Toc26772725)

Введение

Центральной частью любого бизнес-приложения являются данные. Именно на различных манипуляциях с данными и строится большая часть пользовательских сценариев современных приложений. В частности, это сценарии описываются акронимом «CRUD» [1]. Он подразумевает под собой основные возможные пользовательские действия с данными – их создание (create), чтение (read), обновление (update) и удаление (delete). Однако, зачастую, разнородные сами по себе данные, являются связанными друг с другом какими-либо отношениями, и любая операция, производимая на одним набором или типом данным должна так или иначе влиять и на связанные с ними другие наборы. Отсюда появляется необходимость в структурировании информации в модели, которые будут достаточно полно описывать, как и их самих, так и связи, между разными сущностями, а также их влияние друг на друга. На сегодняшний день, самой распространённой является реляционная модель структурирования данных. Она подразумевает под собой описание каждой сущности набором атрибутов, которые хранятся в соответствующих, для этих сущностей, таблицах. Здесь появляется и понятие базы данных, как совокупности данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных.

Таким образом, для корректного выполнения операций над данными, с которыми работает приложение, необходимо чётко и правильно составить реляционную (в частности) структуру, которая будет наиболее полно отражать то, как данные связанные друг с другом и определять начальный интерфейс доступа к ним.

Программное обеспечение для управления проектами — комплексное программное обеспечение, включающее в себя приложения для планирования задач, составления расписания, распределения ресурсов, совместной работы, общения, быстрого управления, документирования и администрирования системы, которое используются совместно для управления проектами.

В моем случае эта система будет реализовывать лишь часть этих функций в связи с ложностью ее реализации, но в целом, такая система должна позволять проводить, в первую очередь, именно планирование, которое будет реализовано в виде досок задачами.

В качестве языка программирования для создания веб-приложения и интерфейса был использован фрейморк Spring Framework [2] языка Java.

В ходе работы, вся логика, которая предусмотрена тематикой приложения, должна быть реализована на средствах, поставляемых выбранной СУБД и её стандартом языка SQL [2]. В качестве основных инструментов, была выбрана СУБД MySQL 8 [3] среда разработки для неё MySQL Workbench 8.0 CE [4]. Разработка самого веб-приложения осуществлялась в IntelliJ IDEA 2018.3.4 [5].

1. Постановка задачи
   1. Обзор аналогов

Первым аналогом для обзора было выбрано веб-приложение «GanttPRO» (рис. 1.1).

Вид: онлайн диаграмма Ганта с возможностью переключения на доску задач по принципу Kanban.

Стоимость: от 5.9$ за пользователя в месяц при годовой оплате.

Бесплатный пробный период: 14 дней с полностью открытыми возможностями.

Платформа: веб-сервис.

Язык: русский, английский.

Ссылка: <https://ganttpro.com/ru/>

|  |
| --- |
| C:\Users\gamer\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Onlayn-diagramma-Ganta-GanttPRO-720x467.png |

Рисунок 1.1 ­– GanttPRO

Эта современная и профессиональная система для управления проектами представляет собой онлайн диаграмму Ганта. Работать с ней просто: в считанные минуты вы создаете задачи, которые размещаются по вертикальной оси, и указываете их даты, которые отображаются на горизонтальной оси. В результате получается визуализированный график, при беглом взгляде на который легко понять объем работ и даты по проекту.

При этом GanttPRO не просто диаграмма Ганта. Инструмент подходит для управления ресурсами и стоимостью проекта, а также для командной работы. А сфера применения ничем не ограничена: это может быть строительство, разработка ПО, производство, сфера услуг, маркетинговая кампания, организация мероприятий, обучение и т.д. Для этих целей сервис предлагает готовые шаблоны.

**Преимущества**:

* Красивый UX/UI дизайн, интуитивный интерфейс.
* Требует около 10 минут на полноценное изучение инструмента.
* Соотношение цена-качество.
* Интерфейс и служба поддержки на русском языке.
* Подходит для управления личными и командными проектами.
* Возможность переключения на доску задач.
* Настраиваемые колонки для персонализации проектов.
* Интеграция с JIRA Cloud.
* Готовые шаблоны для быстрого старта.

**Недостатки**:

* Нет возможности видеть все проекты на одном экране.
* Нет тайм-трекинга.

**Вывод**

Доступный инструмент для управления проектами с интуитивным интерфейсом. Дает четкую картину по всем задачам в личных и командных проектах.

Вторым аналогом для обзора было выбрано веб-приложение «Asana» (рис. 1.2).

**Вид**: платформа для организации работы и управления проектами.

**Стоимость**: от 6.25$ за пользователя в месяц при годовой оплате.

**Бесплатный пробный период**: есть.

**Платформа**: веб-сервис, iOS, Android.

**Язык**: английский, немецкий, испанский, французский, португальский, японский.

|  |
| --- |
| D:\Загрузки\Asana-720x405.png |

Рисунок 1.2 ­– Asana

**Преимущества**:

* Дизайн.
* Много синхронизаций.
* Применим во многих сферах.

**Недостатки**:

* Не для больших команд.
* Не поддерживается русский язык.

**Вывод**

Приятный дизайн, хорошие функциональные возможности. Этот сервис для управления проектами может подойти многим командам.

Третьим аналогом для обзора было выбрано веб-приложение «Basecamp» (рис. 1.3).

Вид: система для управления проектами.

Стоимость: 99$ в месяц независимо от количества членов в команде.

Бесплатный пробный период: 30 дней.

Платформа: веб-сервис, iOS, Android, Windows, MacOS.

Язык: английский.

|  |
| --- |
| D:\Загрузки\Basecamp-720x516.jpg |

Рисунок 1.3 ­– Basecamp

Basecamp представлен несколькими пространствами. **Списки дел** создаются для всей работы, которую нужно сделать. **Доски сообщений** помогут хранить все коммуникации по определенной теме в одном месте. **Графики** покажут все даты задач и события в проекте. **Хранилище** для документов, файлов и изображений – это удобный способ держать все необходимое в одном месте. **Групповой чат** в реальном времени поможет быстро задавать вопросы и получать ответы. **Вопросы**, адресованные всей участникам в одном месте (Check-in questions), заменят статусы и постоянные собрания.

В системе также возможно одновременно работать с клиентами.

**Преимущества:**

* Множество пространств.
* Отчеты.
* Возможности поиска: по ключевым словам, людям, везде или в отдельном пространстве, по изображениям и файлам.
* Публичные ссылки для третьих лиц вне вашего проекта.

**Недостатки:**

* Требует время на изучение.
* Не поддерживается русский язык.
* Недостаточно интеграций.
* Со временем проект накапливает множество деталей, в которых трудно разобраться.

**Вывод**

Популярный инструмент. Но подойдет не всем, особенно любителям четкой визуализации задач по примеру диаграммы Ганта [6].

* 1. Актуальность задачи

Цифровые технологии сильно изменили жизнь человека. Компьютер предоставляет большие возможности для работы с разного вида информацией.

Проект – это универсальная система администрирования приюта. Проект должен работать на разных устройствах, занимать мало места и быстро взаимодействовать с базой данных, чтобы быть часто используемым. Он призван упростить и организовать работу приюта.

* 1. Выбор инструментов разработки

Для решения задачи для хранения данных и создания процедур было выбрано MySQL.

MySQL это система управления реляционными базами данных с открытым исходным кодом (СУРБД) с моделью клиент-сервер. СУРБД — это программное обеспечение или служба, используемая для создания и управления базами данных на основе реляционной модели.

Для работы с БД будет использоваться MySQL Workbench — инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL (рис. 1.4).

|  |
| --- |
|  |

Рисунок 1.4­ – MySQL Workbench

Само же приложение будет создано на основе типизированноно объектно-ориентированного языка программир ования, разработанного компанией Sun Microsystems – Java. Данный язык программирования был выбран из-за его высокопроизводительности за счёт введения в ЯП Just-In-Time компилятора, что позволяет получить высокую производительность, а также из-за его распространённости.

IntelliJ IDEA – интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains (рис. 1.5).

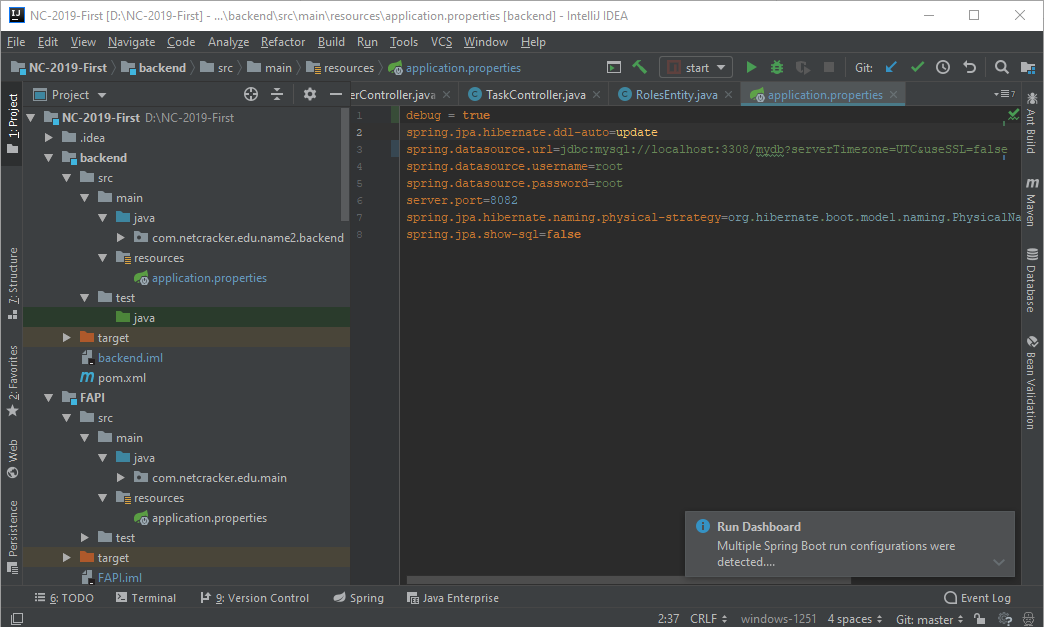


Рисунок 1.5 – IntelliJ IDEA

Программное обеспечение JetBrains IntelliJ IDEA – это ведущая среда быстрой разработки на языке Java. IntelliJ IDEA представляет собой высокотехнологичный комплекс тесно интегрированных инструментов программирования, включающий интеллектуальный редактор исходных текстов с развитыми средствами автоматизации, мощные инструменты рефакторинга кода, встроенную поддержку технологий J2EE, механизмы интеграции со средой тестирования Ant/JUnit и системами управления версиями, уникальный инструмент оптимизации и проверки кода Code Inspection, а также инновационный визуальный конструктор графических интерфейсов.

* 1. Постановка целей

Один из двух элементов научного познавательного процесса, который состоит из постановок и решений; конкретное решение, в свою очередь, тоже может быть представлено в виде процесса из постановок задач (подзадач) и решения их; постановка задачи осуществляется в понятиях и терминах какой-либо области науки, поэтому она опирается на весь предшествующий научный опыт, в том числе - на опыт, позволяющий выбрать область науки для решения задачи; некоторые авторы утверждают, что важность именно постановки задач недооценивается и что именно в постановке задач проявляется так называемое творческое мышление; не исключено, что глобальная задача выживания является тривиальной по постановке, а все прочие, в том числе научные – результатом расщепления глобальной задачи на подзадачи и их решения в процессе решения этой глобальной.

Следуя тому, что задача курсовой работы звучит как веб-приложение «Управление задачами и проектами», у меня должно быть веб-приложение, представленное как набор веб-страниц, запускаемые в браузере, что обеспечит кроссплатформенность данного приложения, так же необходима возможность перехода между данными страницами. Веб-приложение должно взаимодействовать с централизованной базой данных, в которых будут храниться данные, посредством архитектуры клиент-сервер. Пройдёмся по функциональным задачам проекта:

* сохранение рабочей информации в централизованной базе данных
* создавать пользователей с заданными ролями;
* создавать новые проекты;
* в рамках проектов создавать и распределять на исполнителей задачи;
* иметь жизненный цикл проекта и задачи;
* оценивать сложность, важность и дату выполнения задачи.

Следуя основной задаче проекта и его функциональным задачам, можно составить цели работы:

* создание централизованной базы данных;
* заполнение её данными;
* создание веб-приложения;
* разработка структуры клиент-сервер;
* связь приложения с базой данных;
* реализация алгоритма авторизации и регистрации пользователей;
* настройка параметров работы страниц;
* разграничение прав доступа;
* добавление функционала, в соответствии с целью веб-страницы;
* разработка алгоритма исключения ошибок и сбоев приложения;
* подбор простого и понятного интерфейса;
* создание привлекательного внешнего вида веб-приложения;
* тестирование, поиск и исправление ошибок;

После составленных целей работы и подробного анализа задачи можно приступать к разработке.

1. Разработка
   1. Разработка модели базы данных

Как отмечалось в предыдущих разделах, одним из ключевых моментов при проектировании и создании базы данных является грамотный анализ предметной области приложения. Как следствие – составление такой модели данных, которая будет правильно отражать то, как с этими с данными в общем, и этой моделью, в частности, подразумевается взаимодействовать.

Результатом корректно проведённого анализа, проектирования, и разработки, является модель, способная предоставить функционально все необходимые возможности для пользователя. В частности, для приложения «Система управления проектами», можно выделить набор таких операций, как:

* возможность регистрации, авторизации и редактирования пользователей;
* возможность добавления и редактирования задач;
* возможность добавления и редактирования проектов.

Основой инфраструктуры базы данных является грамотно спроектированная модель, которая отображает связь пользовательских таблиц. Правильное и корректное взаимодействие их друг с другом как раз и заключается в схеме базы данных со связями, верно отображающими их положение.

На рисунке 2.1 отображена модель базы данных, полученная в ходе анализа предметной области и необходимого функционала. Между сущностями, представленными таблицами, также настроены связи, отражающие их взаимосвязь.

В качестве примера, того, на чём основывались решения построения тех или иных связей, рассмотрю связи сущностей «Tasks» и «Projects», а также «Task» и «Users».

«Tasks» и «Projects» являются центральными понятиями в проектируемой системе, так как в совокупности они представляют собой, в некотором смысле, информационное ядро системы, отражающее её основное предназначение – управление проектами и создания под них определенных задач. Связь между этими сущностями определена как «один ко многим». Это объясняется тем, что один проект может (и должен) содержать некоторое множество задач.

Связь между сущностями «Task» и «Users» отражает факт участия в создании и работе над задачей – один пользователь может как создавать множество задач, так учувствовать в их решении. Соответственно, связь между ними должна быть отражена отношением «один ко многим».

|  |
| --- |
|  |

Рисунок 2.1 – Структурная схема базы данных.

Остальные примеры связей между сущностями являются довольно тривиальными в сравнении с вышеуказанными, и логика их проектирования не должна вызывать вопросов, исходя только из семантического отражения любой из сущностей.

На рисунке 2.1 проиллюстрированы следующие связи:

* связь «Roles-Users»: один ко многим;
* связь «Projects-Tasks»: один ко многим;
* связь «Users-Projects»: один ко многим;
* связь «Tasks-Attachment»: один ко многим;
* связь «Tasks-Comments»: один ко многим.
* связь «Status-Tasks»: один ко многим;
* связь «Priorities-Tasks»: один ко многим;
* связь «Users-Tasks»: один ко многим;
* связь «Users-Tasks»: один ко многим;
* связь «Users-Comments»: один ко многим.
  1. Разработка архитектуры проекта

Архитектура программного обеспечения (англ. software architecture) — совокупность важнейших решений об организации программной системы. Архитектура включает: выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов; соединение выбранных элементов структуры и поведения, во всё более крупные системы; архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

Веб-приложение было создано на базе языка программирования Java используя среду разработки IntelliJ IDEA 2019.2.3. Структура проекта имеет следующий вид (рисунок 2.1).

В качестве языка программирования для создания веб-приложения и интерфейса был использован фрейморк Spring Framework языка Java.

Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java - на любых платформах. Ключевой элемент Spring - поддержка инфраструктуры на уровне приложения: основное внимание уделяется "водопроводу" бизнес-приложений, поэтому разработчики могут сосредоточиться на бизнес-логике без лишних настроек в зависимости от среды исполнения.

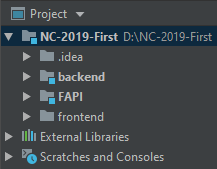


Рисунок 2.1 – общая структура проекта

Сам проект из себя приставляет трехуровневое приложение, с микросервиснаой архитектурой [7]:

* Frontend;
* FAPI;
* Backend.
  1. Структура проекта

Так же на рисунке 2.2 можно увидеть полную структуру проекта.

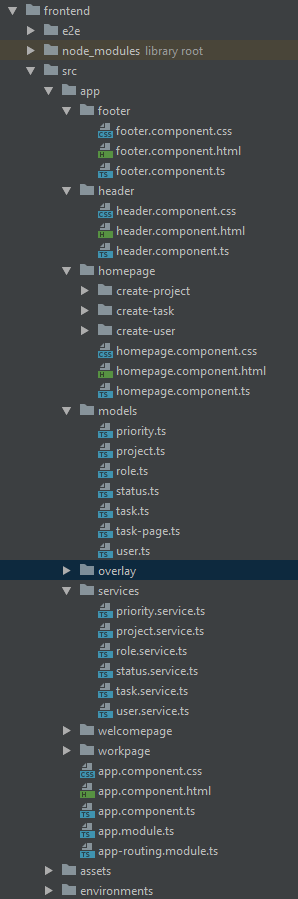
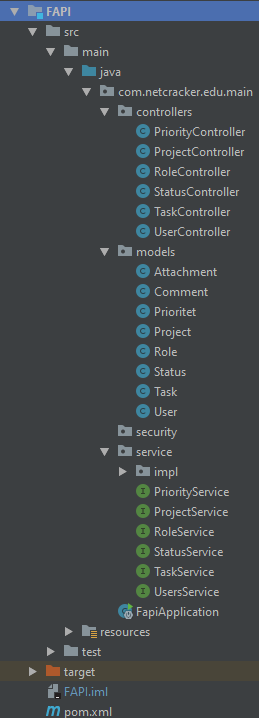
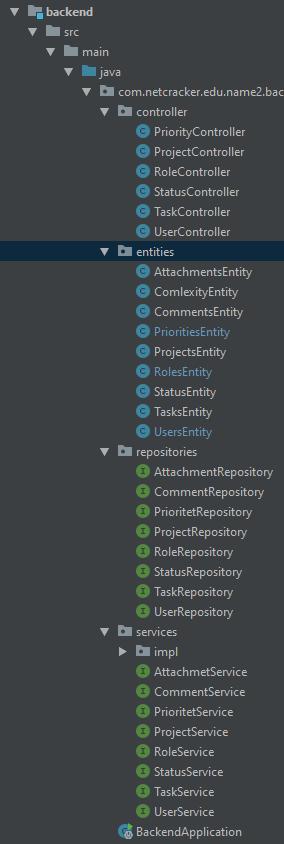


Рисунок 2.2 – полная структура проекта

* + 1. Backend

Состоит из:

* Controller.
* Entities
* Repositories.
* Services.

Каждый controller предоставляет серверу связь с БД посредством запросов, что видно в литсинге 2.1, где реализованы запросы к таблице Tasks. Каждый файл реализует связь со своей таблицей.

|  |
| --- |
| @RestController  @RequestMapping("/api/task")  public class TaskController {  @Autowired  private TaskService taskService;  @Autowired  private TaskRepository taskRepository;  @RequestMapping(value = "/idTask/{idTask}", method = RequestMethod.GET)  public ResponseEntity<TasksEntity> getTaskByIdTask(@PathVariable(name = "idTask") int idTask) {  TasksEntity task = taskService.getTaskByIdTask(idTask);  return ResponseEntity.ok(task);  }  @RequestMapping(value = "", method = RequestMethod.GET)  public List<TasksEntity> getAllTasks() {  List<TasksEntity> data= taskService.findAll();  return data;  }  @PatchMapping("")  public ResponseEntity<TasksEntity> updateTask(@RequestBody TasksEntity task) {  TasksEntity updatedTask = taskRepository.getTaskByIdTask(task.getIdTask());  updatedTask.setIdStatus(task.getIdStatus());  updatedTask.setIdPriority(task.getIdPriority());  updatedTask.setUpdatedDate(task.getUpdatedDate());  updatedTask.setDescription(task.getDescription());  updatedTask.setDueDate(task.getDueDate());  updatedTask.setResolvedDate(task.getResolvedDate());  updatedTask.setClosedDate(task.getClosedDate());  updatedTask.setEstimation(task.getEstimation());  updatedTask.setAssignee(task.getAssignee());  updatedTask.setReporter(task.getReporter());  TasksEntity taskResult = taskService.save(updatedTask);  if (taskResult == null)  return ResponseEntity.notFound().build();  return ResponseEntity.ok(taskResult);  }  @RequestMapping(method = RequestMethod.POST)  public TasksEntity saveTask(@RequestBody TasksEntity task) {  return taskService.save(task);  }  } |

Листинг 2.1 – TaskController

Каждая Entity представляет из себя представления таблицы из БД. Где объявляются поля таблицы, методы доступа и связи между таблицами, в листинге 2.2.

|  |
| --- |
| package com.netcracker.edu.name2.backend.entities;  import javax.persistence.\*;  @Entity  @Table(name = "users", schema = "mydb")  public class UsersEntity {  private int idUsers;  private String login;  private String password;  private String personalData;  //private int idRole;  private RolesEntity idRole;  @Id  @Column(name = "idUsers" )  public int getIdUsers() {  return idUsers;  }  public void setIdUsers(int idUsers) {  this.idUsers = idUsers;  }  @Basic  @Column(name = "login")  public String getLogin() {  return login;  }  public void setLogin(String login) {  this.login = login;  }  @Basic  @Column(name = "password")  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  @Basic  @Column(name = "personalData")  public String getPersonalData() {  return personalData;  }  public void setPersonalData(String personalData) {  this.personalData = personalData;  }    @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  UsersEntity that = (UsersEntity) o;  if (idUsers != that.idUsers) return false;  //if (idRole != that.idRole) return false;  if (login != null ? !login.equals(that.login) : that.login != null) return false;  if (password != null ? !password.equals(that.password) : that.password != null) return false;  if (personalData != null ? !personalData.equals(that.personalData) : that.personalData != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idUsers;  result = 31 \* result + (login != null ? login.hashCode() : 0);  result = 31 \* result + (password != null ? password.hashCode() : 0);  result = 31 \* result + (personalData != null ? personalData.hashCode() : 0);  //result = 31 \* result + idRole;  return result;  }  @ManyToOne  @JoinColumn(name = "idRole", referencedColumnName = "idRole", nullable = false)  public RolesEntity getIdRole() {  return idRole;  }  public void setIdRole(RolesEntity idRole) {  this.idRole = idRole;  }  } |

Листинг 2.2 –UsersEntity;

Каждый Services предоставляет иным микросервисам доступ к методам. В нашем приложении этим микросервисом является FAPI.

|  |
| --- |
| @Service  public class UserServiceImpl implements UserService {  @Autowired  private UserRepository userRepository;  @Override  public List<UsersEntity> findAll() {  return (List<UsersEntity>) userRepository.findAll();  }  @Override  public UsersEntity getUserByIdUsers(int idUser){  return userRepository.getUserByIdUsers(idUser);  }  @Override  public UsersEntity save(UsersEntity user) {  return userRepository.save(user);  }  @Override  public UsersEntity getUserByLogin(String login){  return userRepository.getUserByLogin(login);  }  } |

Листинг 2.3 – UserServiceImpl

* + 1. FrontendAPI

Задача FAPI:

* Слабая связанность компонентов системы, чтобы бэкенд и фронтенд можно было развивать параллельно.
* Высокая масштабируемость, чтобы новый API не мешал наращивать функциональность.
* Стабильность и согласованность.

Структура FAPI почти идентична, состоит из:

* Controller.
* Entities
* Services.
  + 1. Frontend

Frontend получает данные из FAPI и получая их выводит на веб-страницу. Имеет модульную структуру.

Состоит из:

* Компонентов.
* Моделей.
* Сервисов.

Модели – это представление таблиц базы данных, пример на листинге 2.4.

|  |
| --- |
| export class Task {  idTask:number;  description:string;  estimation:string;  personalData:string;  createDate:Date;  dueDate:Date;  updatedDate:Date;  resolvedDate:Date;  closedDate:Date;  idProject:Project ;  idPriority:Priority;  idStatus:Status;  assignee:User;  reporter:User;  } |

Листинг 2.4 – Task

Сервисы связываются с FAPI, пример на листинге 2.5.

|  |
| --- |
| @Injectable({  providedIn: 'root'  })  export class PriorityService {  constructor(private http: HttpClient) { }  getPriority(): Observable<Priority[]>{  return this.http.get<Priority[]>("/api/priority");  }  findById(id:string): any{  return this.http.get("/api/priority/priority/" + id);  }  findByName(name:string): any{  return this.http.get("/api/priority/name/" + name);  }  } |

Листинг 2.5 – PriorityService

Пример компоненты на листинге 2.6 ниже.

|  |
| --- |
| <div class="col-lg-4 containerCompact" >  <h3 class="centered">New Project</h3>  <label><b>Code</b></label> <input type="text" placeholder="" [(ngModel)]="project.idProject" required>  <label><b>Summary</b></label>  <input type="text" placeholder="" [(ngModel)]="project.summary" required>  <button (click)="onCreateProjectClick()" class="righted">Create</button>  </div> |

Листинг 2.6 – CreateProjectComponent

* 1. Разработка таблиц БД

Центральным объектом любой базы данных является связанная совокупность таблиц. Таблицы ограничивают и упорядочивают хранимую информацию, а также обеспечивают связанность за счет внешних ключей.

В первую очередь перед созданием самих таблиц необходимо выделить главные сущности, которые отражают хранимую информацию. В ходе разработки я выделил: «Projects», «Tasks» и «Users». Они представляют «Ядро» приложения «Система управления проектами».

На листинге 3.1 можно увидеть создание главной из связующих таблиц ­– Tasks.

|  |
| --- |
| -- Table `mydb`.`Tasks`  CREATE TABLE IF NOT EXISTS `mydb`.`Tasks` (  `idTask` INT NOT NULL AUTO\_INCREMENT,  `description` VARCHAR(45) NULL,  `estimation` VARCHAR(10) NULL,  `createDate` DATE NULL,  `dueDate` DATE NULL,  `updatedDate` DATE NULL,  `resolvedDate` DATE NULL,  `closedDate` DATE NULL,  `idProject` INT NOT NULL,  `idPriority` INT NOT NULL,  `idStatus` INT NOT NULL,  `idСomplexity` INT NOT NULL,  `assignee` INT NOT NULL,  `reporter` INT NOT NULL,  PRIMARY KEY (`idTask`),  CONSTRAINT `fk\_Tasks\_Projects1`  FOREIGN KEY (`idProject`)  REFERENCES `mydb`.`Projects` (`idProject`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Priorities1`  FOREIGN KEY (`idPriority`)  REFERENCES `mydb`.`Priorities` (`idPriority`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Statuses1`  FOREIGN KEY (`idStatus`)  REFERENCES `mydb`.`Status` (`idStatus`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Users1`  FOREIGN KEY (`assignee`)  REFERENCES `mydb`.`Users` (`idUsers`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Users2`  FOREIGN KEY (`reporter`)  REFERENCES `mydb`.`Users` (`idUsers`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

Листинг 2.4 – Создание таблицы Tasks;

Неотъемлемой частью почти любой базы данных является также информация о пользователях, использующих это приложение. В рамках моего приложения также предусмотрена возможность работы группы пользователей над одним проектом, включая разделение этой группы по ролям.

1. Руководство пользователя

Для проверки работоспособности, необходимо в первую очередь установить базу данных MySQL. После установки, необходимо произвести стандартную конфигурацию базы данных с указанием порта доступа, после чего указать пароль для доступа. Пароль необходим для подключения серверной части к базе данных и работы CRUD операций.

После настройки подключения базы данных, необходимо запустить сервер. Для запуска сервера необходимо программное обеспечение для работы IntelliJ IDEA установленное на компьютере.

По завершению запуска сервера, мы можем перейти на три основных веб страницы, предназначенные, для терминала, главного рабочего места, и главного администратора. Для перехода на каждую из вышеперечисленных страниц, необходимо ввести в адресную строку браузера url-адрес «localhost:4200» (в случае, если хотим открыть страницы на стороне сервера). Результат перехода представлен на рисунке 3.1.

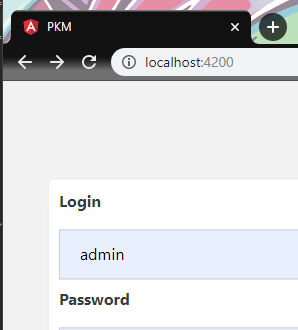


Рисунок 3.1 – строка URL

Первым нашим шагом в работе с программой, это авторизация, форму можно видеть на рисунке 3.2. Для этого нам необходимо ввести логин и пароль в соответствующую форму и нажать кнопку «Login», и в случае того, что данные окажутся верны нам перенесет на страницу с заданиями.

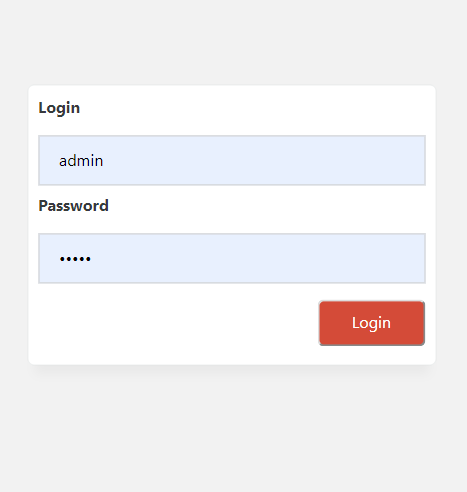


Рисунок 3.2 – окно авторизации

В окне управления заданиями мы имеем четыре возможности (рис. 3.3):

* создания проекта;
* создания задания;
* создания пользователя;
* управление выбранным заданием.

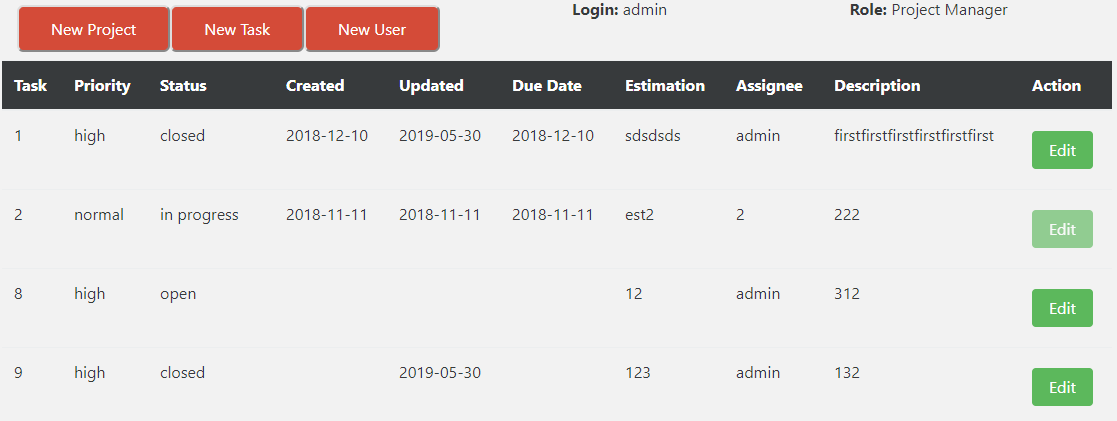


Рисунок 3.3 – окно с заданиями

Сперва мы рассмотрим возможность создания проектов, именно на них и базируются все задания, окно создания проекта мы можем видеть на рисунке 3.4. Состоит оно из двух полей: код проекта и его описание, так же в форме не видно, однако идет привязка проекта к создавшему его пользователю. После ввода всех знаний мы жмем кнопку «Create» и в случае, если проектов с таким кодом в базе данных нет, то проект успешно создаться.

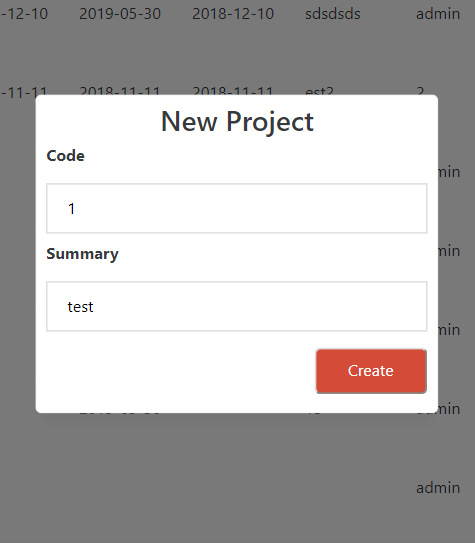


Рисунок 3.4 – окно создания проекта

Следующее немаловажное окно, это окно создания задания, оно состоит из следующих полей, как видно на рисунке 3.5:

* код проекта;
* дескриптор;
* приоритет;
* ориентировочная дата завершения;
* описание;
* исполнитель;

Так же, как и в окне создания проектов, здесь при создании задачи поле reporter присваивается значение пользователя, создавшего задачу. Помимо репортера так же заполняется поле CreateDate, которое отвечает за дату создания задачи.

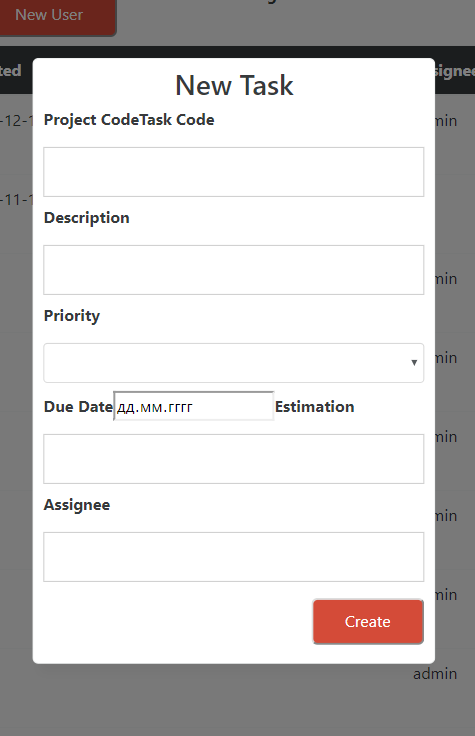


Рисунок 3.5 – окно создания задания

И последним окном создания является окно создания пользователей. Состоит из четырех полей, показанных ан рисунке 3.6, а именно:

* логин;
* пароль;
* персональные данные;
* роль.

На выбор предоставлены три роли:

* Project Manager;
* Developer;
* Tester.

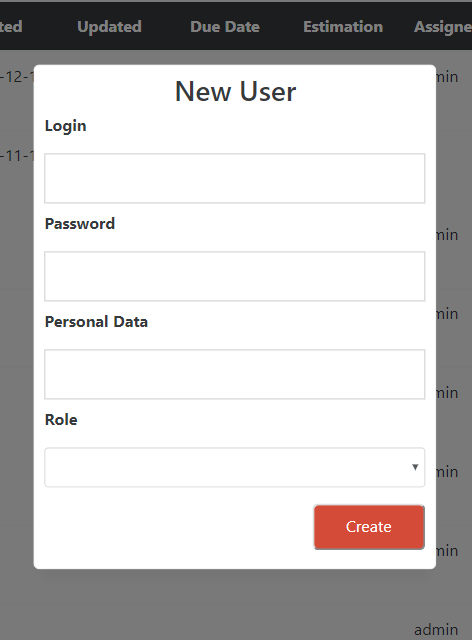


Рисунок 3.6 – окно регистрации пользователей

И последним доступным модулем является окно задачи, показанное на рисунке 3.7. Здесь мы можем менять статус выполнения нашей задачи

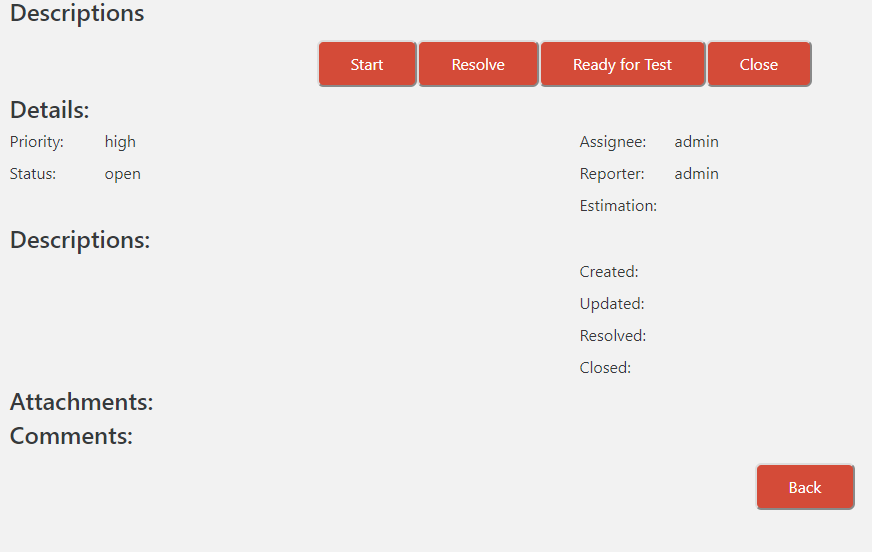


Рисунок 3.7 – окно управления заданием

Заключение

В ходе выполнения работы были проанализированы аналоги веб-приложения «Управление задачами и проектами», выявлены их положительные и отрицательные стороны. Что было учтено при проектировании дизайна приложения.

Само приложения представляет из себя три микросервиса.

К тому же созданы реляционная база данных для веб-приложения «Управления проектами и системами», имеющая 8 таблиц. Также было спроектировано и создано само приложение «Управление задачами и проектами».

Была создана авторизация пользователей.

В ходе выполнения данной работы подробно были закреплены знания для проектирования баз данных, с учётом большого количества хранимых параметров.

Для использования всего функционала созданной базы данных, был создан сервер, написанный на Spring со всеми необходимыми API, были созданы веб страницы для систематизации информации хранящейся в базе данных. В качестве СУБД использовался MySQL.

Благодаря проведенной работе, был создан проект, который может быть болванкой для создания системы управления проектами.

Основным плюсом данного проекта, является простота и возможность бесплатного распространения, так как технологии, использованные в данном проекте можно использовать OS Linux, которое является бесплатным.

Список используемых источников

1. Свободная энциклопедия Википедия [Электронный ресурс] /CRUD – Режим доступа: https://ru.wikipedia.org/wiki/CRUD ­– Дата доступа: 01.11.2019.
2. Spring Framework [Электронный ресурс] – Режим доступа: <https://spring-projects.ru/projects/spring-framework/> ­– Дата доступа: 02.11.2019.
3. MySQL [Электронный ресурс]  / MySQL 5.7 – Режим доступа: <https://dev.mysql.com/doc/refman/5.7/en/> ­– Дата доступа: 02.11.2019.
4. MySQL [Электронный ресурс]  / MySQL Workbench – Режим доступа: <https://dev.mysql.com/doc/workbench/en/> ­– Дата доступа: 11.11.2019.
5. IntelliJ IDEA [Электронный ресурс] / IntelliJ IDEA – Режим доступа: <https://www.jetbrains.com/ru-ru/idea/>­– Дата доступа: 11.11.2019.
6. 10 систем для управления проектами [Электронный ресурс] – Режим доступа: <https://blog.ganttpro.com/ru/sistemy-instrumenty-servisy-upravlenie-proektami> ­– Дата доступа: 12.11.2019.
7. Habr [Электронный ресурс] / Микросервисы – Режим доступа: <https://habr.com/ru/post/249183/> ­– Дата доступа: 11.11.2019.

Приложения А. Листинг создания БД и таблиц.

|  |  |
| --- | --- |
| CREATE TABLE IF NOT EXISTS `mydb`.`Projects` (  `idProject` INT NOT NULL AUTO\_INCREMENT,  `summary` VARCHAR(45) NULL,  `reporter` INT NOT NULL,  PRIMARY KEY (`idProject`),  CONSTRAINT `fk\_Projects\_Users1`  FOREIGN KEY (`reporter`)  REFERENCES `mydb`.`Users` (`idUsers`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB;  CREATE TABLE IF NOT EXISTS `mydb`.`Priorities` (  `idPriority` INT NOT NULL AUTO\_INCREMENT,  `name` VARCHAR(20) NULL,  PRIMARY KEY (`idPriority`))  ENGINE = InnoDB;  CREATE TABLE IF NOT EXISTS `mydb`.`Status` (  `idStatus` INT NOT NULL AUTO\_INCREMENT,  `name` VARCHAR(45) NULL,  PRIMARY KEY (`idStatus`))  ENGINE = InnoDB;  CREATE TABLE IF NOT EXISTS `mydb`.`Tasks` (  `idTask` INT NOT NULL AUTO\_INCREMENT,  `description` VARCHAR(45) NULL,  `estimation` VARCHAR(10) NULL,  `createDate` DATE NULL,  `dueDate` DATE NULL,  `updatedDate` DATE NULL,  `resolvedDate` DATE NULL,  `closedDate` DATE NULL,  `idProject` INT NOT NULL,  `idPriority` INT NOT NULL,  `idStatus` INT NOT NULL,  `idСomplexity` INT NOT NULL,  `assignee` INT NOT NULL,  `reporter` INT NOT NULL,  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Priorities1`  FOREIGN KEY (`idPriority`)  REFERENCES `mydb`.`Priorities` (`idPriority`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Statuses1`  FOREIGN KEY (`idStatus`)  REFERENCES `mydb`.`Status` (`idStatus`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Tasks\_Users1`  FOREIGN KEY (`assignee`)  REFERENCES `mydb`.`Users` (`idUsers`)  ON DELETE NO ACTION  ON UPDATE NO ACTION, | CREATE TABLE IF NOT EXISTS `mydb`.`Attachments` (  `idAttachment` INT NOT NULL AUTO\_INCREMENT,  `idTask` INT NOT NULL,  `path` VARCHAR(45) NULL,  PRIMARY KEY (`idAttachment`),  INDEX `fk\_Attachments\_Tasks1\_idx` (`idTask` ASC) VISIBLE,  CONSTRAINT `fk\_Attachments\_Tasks1`  FOREIGN KEY (`idTask`)  REFERENCES `mydb`.`Tasks` (`idTask`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB;  CREATE TABLE IF NOT EXISTS `mydb`.`Comments` (  `idComment` INT NOT NULL AUTO\_INCREMENT,  `idTask` INT NOT NULL,  `comment` VARCHAR(45) NULL,  `idUser` INT NOT NULL,  PRIMARY KEY (`idComment`),  INDEX `fk\_Comments\_Tasks\_idx` (`idTask` ASC) VISIBLE,  INDEX `fk\_Comments\_Users1\_idx` (`idUser` ASC) VISIBLE,  CONSTRAINT `fk\_Comments\_Tasks`  FOREIGN KEY (`idTask`)  REFERENCES `mydb`.`Tasks` (`idTask`)  ON DELETE NO ACTION  ON UPDATE NO ACTION,  CONSTRAINT `fk\_Comments\_Users1`  FOREIGN KEY (`idUser`)  REFERENCES `mydb`.`Users` (`idUsers`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB;  CREATE TABLE IF NOT EXISTS `mydb`.`Roles` (  `idRole` INT NOT NULL AUTO\_INCREMENT,  `name` VARCHAR(20) NULL,  PRIMARY KEY (`idRole`))  ENGINE = InnoDB;  CREATE TABLE IF NOT EXISTS `mydb`.`Users` (  `idUsers` INT NOT NULL AUTO\_INCREMENT,  `login` VARCHAR(20) NULL,  `password` VARCHAR(20) NULL,  `personalData` VARCHAR(45) NULL,  `idRole` INT NOT NULL,  PRIMARY KEY (`idUsers`),  FOREIGN KEY (`idRole`)  REFERENCES `mydb`.`Roles` (`idRole`)  ON DELETE NO ACTION  ON UPDATE NO ACTION)  ENGINE = InnoDB; |

Приложения Б. Листинг Микросервиса Backend

|  |  |
| --- | --- |
| public class PrioritiesEntity {  private int idPriority;  private String name;  @Id  @Column(name = "idPriority")  public int getIdPriority() {  return idPriority;  }  public void setIdPriority(int idPriority) {  this.idPriority = idPriority;  }  @Basic  @Column(name = "name")  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  PrioritiesEntity that = (PrioritiesEntity) o;  if (idPriority != that.idPriority) return false;  if (name != null ? !name.equals(that.name) : that.name != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idPriority;  result = 31 \* result + (name != null ? name.hashCode() : 0);  return result;  }  }  @Table(name = "projects", schema = "mydb")  public class ProjectsEntity {  private int idProject;  private String summary;  //private int reporter;  private UsersEntity reporter;  @Id  @Column(name = "idProject")  public int getIdProject() {  return idProject;  }  public void setIdProject(int idProject) {  this.idProject = idProject;  }  @Basic  @Column(name = "summary")  public String getSummary() {  return summary;  }  public void setSummary(String summary) {  this.summary = summary;  }  /\*@Basic  @Column(name = "reporter")  public int getReporter() {  return reporter;  }  public void setReporter(int reporter) {  this.reporter = reporter;  }\*/  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  ProjectsEntity that = (ProjectsEntity) o;  if (idProject != that.idProject) return false;  if (reporter != that.reporter) return false;  if (summary != null ? !summary.equals(that.summary) : that.summary != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idProject;  result = 31 \* result + (summary != null ? summary.hashCode() : 0);  //result = 31 \* result + reporter;  return result;  }  @ManyToOne  @JoinColumn(name = "reporter", referencedColumnName = "idUsers", nullable = false)  public UsersEntity getReporter() {  return reporter;  }  public void setReporter(UsersEntity reporter) {  this.reporter = reporter;  }  }  @Table(name = "roles", schema = "mydb")  public class RolesEntity {  private int idRole;  private String name;  @Id  @Column(name = "idRole")  public int getIdRole() {  return idRole;  }  public void setIdRole(int idRole) {  this.idRole = idRole;  }  @Basic  @Column(name = "name")  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  RolesEntity that = (RolesEntity) o;  if (idRole != that.idRole) return false;  if (name != null ? !name.equals(that.name) : that.name != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idRole;  result = 31 \* result + (name != null ? name.hashCode() : 0);  return result;  }  }  @Table(name = "status", schema = "mydb")  public class StatusEntity {  private int idStatus;  private String name;  @Id  @Column(name = "idStatus")  public int getIdStatus() {  return idStatus;  }  public void setIdStatus(int idStatus) {  this.idStatus = idStatus;  }  @Basic  @Column(name = "name")  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  StatusEntity that = (StatusEntity) o;  if (idStatus != that.idStatus) return false;  if (name != null ? !name.equals(that.name) : that.name != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idStatus;  result = 31 \* result + (name != null ? name.hashCode() : 0);  return result;  }  }  @Table(name = "tasks", schema = "mydb")  public class TasksEntity {  private int idTask;  private String description;  private String estimation;  private Date createDate;  private Date dueDate;  private Date updatedDate;  private Date resolvedDate;  private Date closedDate;  private ProjectsEntity idProject;  private PrioritiesEntity idPriority;  private StatusEntity idStatus;  private UsersEntity assignee;  private UsersEntity reporter;  @Id  @Column(name = "idTask")  public int getIdTask() {  return idTask;  }  public void setIdTask(int idTask) {  this.idTask = idTask;  }  @Basic  @Column(name = "description")  public String getDescription() {  return description;  }  public void setDescription(String description) {  this.description = description;  }  @Basic  @Column(name = "estimation")  public String getEstimation() {  return estimation;  }  public void setEstimation(String estimation) {  this.estimation = estimation;  }  @Basic  @Column(name = "createDate")  public Date getCreateDate() {  return createDate;  }  public void setCreateDate(Date createDate) {  this.createDate = createDate;  }  @Basic  @Column(name = "dueDate")  public Date getDueDate() {  return dueDate;  }  public void setDueDate(Date dueDate) {  this.dueDate = dueDate;  }  @Basic  @Column(name = "updatedDate")  public Date getUpdatedDate() {  return updatedDate;  }  public void setUpdatedDate(Date updatedDate) {  this.updatedDate = updatedDate;  }  @Basic  @Column(name = "resolvedDate")  public Date getResolvedDate() {  return resolvedDate;  }  public void setResolvedDate(Date resolvedDate) {  this.resolvedDate = resolvedDate;  }  @Basic  @Column(name = "closedDate")  public Date getClosedDate() {  return closedDate;  }  public void setClosedDate(Date closedDate) {  this.closedDate = closedDate;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  TasksEntity that = (TasksEntity) o;  if (idTask != that.idTask) return false;  if (idProject != that.idProject) return false;  if (idPriority != that.idPriority) return false;  if (idStatus != that.idStatus) return false;  if (assignee != that.assignee) return false;  if (reporter != that.reporter) return false;  if (description != null ? !description.equals(that.description) : that.description != null) return false;  if (estimation != null ? !estimation.equals(that.estimation) : that.estimation != null) return false;  if (createDate != null ? !createDate.equals(that.createDate) : that.createDate != null) return false;  if (dueDate != null ? !dueDate.equals(that.dueDate) : that.dueDate != null) return false;  if (updatedDate != null ? !updatedDate.equals(that.updatedDate) : that.updatedDate != null) return false;  if (resolvedDate != null ? !resolvedDate.equals(that.resolvedDate) : that.resolvedDate != null) return false;  if (closedDate != null ? !closedDate.equals(that.closedDate) : that.closedDate != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idTask;  result = 31 \* result + (description != null ? description.hashCode() : 0);  result = 31 \* result + (estimation != null ? estimation.hashCode() : 0);  result = 31 \* result + (createDate != null ? createDate.hashCode() : 0);  result = 31 \* result + (dueDate != null ? dueDate.hashCode() : 0);  result = 31 \* result + (updatedDate != null ? updatedDate.hashCode() : 0);  result = 31 \* result + (resolvedDate != null ? resolvedDate.hashCode() : 0);  result = 31 \* result + (closedDate != null ? closedDate.hashCode() : 0);  /\*result = 31 \* result + idProject;  result = 31 \* result + idPriority;  result = 31 \* result + idStatus;  result = 31 \* result + assignee;  result = 31 \* result + reporter;\*/  return result;  }  @ManyToOne  @JoinColumn(name = "idProject", referencedColumnName = "idProject", nullable = false)  public ProjectsEntity getIdProject() {  return idProject;  }  public void setIdProject(ProjectsEntity idProject) {  this.idProject = idProject;  }  @ManyToOne  @JoinColumn(name = "idPriority", referencedColumnName = "idPriority", nullable = false)  public PrioritiesEntity getIdPriority() {  return idPriority;  }  public void setIdPriority(PrioritiesEntity prioritiesByIdPriority) {  this.idPriority = prioritiesByIdPriority;  }  @ManyToOne  @JoinColumn(name = "idStatus", referencedColumnName = "idStatus", nullable = false)  public StatusEntity getIdStatus() {  return idStatus;  }  public void setIdStatus(StatusEntity statusByIdStatus) {  this.idStatus = statusByIdStatus;  }  @ManyToOne  @JoinColumn(name = "assignee", referencedColumnName = "idUsers", nullable = false)  public UsersEntity getAssignee() {  return assignee;  }  public void setAssignee(UsersEntity assignee) {  this.assignee = assignee;  }  @ManyToOne  @JoinColumn(name = "reporter", referencedColumnName = "idUsers", nullable = false)  public UsersEntity getReporter() {  return reporter;  }  public void setReporter(UsersEntity reporter) {  this.reporter = reporter;  }  } | public class ProjectController {  @Autowired  private ProjectService projectService;  @RequestMapping(value = "/project/{idProject}", method = RequestMethod.GET)  public ResponseEntity<ProjectsEntity> getProjectByIdProject(@PathVariable(name = "idProject") int idProject) {  ProjectsEntity project = projectService.getProjectByIdProject(idProject);  return ResponseEntity.ok(project);  }  @RequestMapping(value = "", method = RequestMethod.GET)  public List<ProjectsEntity> getAllProjects() {  return projectService.findAll();  }  @RequestMapping(method = RequestMethod.POST)  public ProjectsEntity saveProject(@RequestBody ProjectsEntity projectsEntity) {  return projectService.save(projectsEntity);  }  }  public class RoleController {  @Autowired  private RoleService roleService;  @RequestMapping(value = "", method = RequestMethod.GET)  public Iterable<RolesEntity> getAllStatus() {  return roleService.findAll();  }  @RequestMapping(value = "/role/{idRole}", method = RequestMethod.GET)  public ResponseEntity<RolesEntity> getRoleByIdRole(@PathVariable(name = "idRole") int idRole) {  RolesEntity statusEntity = roleService.getRoleByIdRole(idRole);  return ResponseEntity.ok(statusEntity);  }  @RequestMapping(value = "/name/{name}", method = RequestMethod.GET)  public ResponseEntity<RolesEntity> getRoleByName(@PathVariable(name = "name") String name) {  RolesEntity statusEntity = roleService.getRoleByName(name);  return ResponseEntity.ok(statusEntity);  }  }  public class TaskController {  @Autowired  private TaskService taskService;  @Autowired  private TaskRepository taskRepository;  @RequestMapping(value = "/idTask/{idTask}", method = RequestMethod.GET)  public ResponseEntity<TasksEntity> getTaskByIdTask(@PathVariable(name = "idTask") int idTask) {  TasksEntity task = taskService.getTaskByIdTask(idTask);  return ResponseEntity.ok(task);  }  @RequestMapping(value = "", method = RequestMethod.GET)  public List<TasksEntity> getAllTasks() {  List<TasksEntity> data= taskService.findAll();  return data;  }  @PatchMapping("")  public ResponseEntity<TasksEntity> updateTask(@RequestBody TasksEntity task) {  TasksEntity updatedTask = taskRepository.getTaskByIdTask(task.getIdTask());  updatedTask.setIdStatus(task.getIdStatus());  updatedTask.setIdPriority(task.getIdPriority());  updatedTask.setUpdatedDate(task.getUpdatedDate());  updatedTask.setDescription(task.getDescription());  updatedTask.setDueDate(task.getDueDate());  updatedTask.setResolvedDate(task.getResolvedDate());  updatedTask.setClosedDate(task.getClosedDate());  updatedTask.setEstimation(task.getEstimation());  updatedTask.setAssignee(task.getAssignee());  updatedTask.setReporter(task.getReporter());  TasksEntity taskResult = taskService.save(updatedTask);  if (taskResult == null)  return ResponseEntity.notFound().build();  return ResponseEntity.ok(taskResult);  }  @RequestMapping(method = RequestMethod.POST)  public TasksEntity saveTask(@RequestBody TasksEntity task) {  return taskService.save(task);  }  }  public class PriorityController {  @Autowired  private PrioritetService prioritetService;  @RequestMapping(value = "", method = RequestMethod.GET)  public Iterable<PrioritiesEntity> getAllStatus() {  return prioritetService.findAll();  }  @RequestMapping(value = "/priority/{idPriority}", method = RequestMethod.GET)  public ResponseEntity<PrioritiesEntity> getRoleByIdPriority(@PathVariable(name = "idPriority") int idPriority) {  PrioritiesEntity statusEntity = prioritetService.getRoleByIdPriority(idPriority);  return ResponseEntity.ok(statusEntity);  }  @RequestMapping(value = "/name/{name}", method = RequestMethod.GET)  public ResponseEntity<PrioritiesEntity> getRoleByName(@PathVariable(name = "name") String name) {  PrioritiesEntity statusEntity = prioritetService.getRoleByName(name);  return ResponseEntity.ok(statusEntity);  }  }  public class StatusController {  @Autowired  private StatusService statusService;  @RequestMapping(value = "", method = RequestMethod.GET)  public Iterable<StatusEntity> getAllStatus() {  return statusService.findAll();  }  @RequestMapping(value = "/status/{idStatus}", method = RequestMethod.GET)  public ResponseEntity<StatusEntity> getStatusByIdStatus(@PathVariable(name = "idStatus") int idStatus) {  StatusEntity statusEntity = statusService.getStatusByIdStatus(idStatus);  return ResponseEntity.ok(statusEntity);  }  @RequestMapping(value = "/name/{name}", method = RequestMethod.GET)  public ResponseEntity<StatusEntity> getStatusByName(@PathVariable(name = "name") String name) {  StatusEntity statusEntity = statusService.getStatusByName(name);  return ResponseEntity.ok(statusEntity);  }  }  public class UserController {  @Autowired  private UserService userService;  @RequestMapping(value = "/idUser/{idUser}", method = RequestMethod.GET)  public ResponseEntity<UsersEntity> getUserByIdUsers(@PathVariable(name = "idUser") int idUser) {  UsersEntity user = userService.getUserByIdUsers(idUser);  return ResponseEntity.ok(user);  }  @RequestMapping(value = "/login/{login}", method = RequestMethod.GET)  public ResponseEntity<UsersEntity> getUserByLogin(@PathVariable(name = "login") String login) {  UsersEntity user = userService.getUserByLogin(login);  return ResponseEntity.ok(user);  }  @RequestMapping(value = "", method = RequestMethod.GET)  public List<UsersEntity> getAllUsers() {  return userService.findAll();  }  @RequestMapping(method = RequestMethod.POST)  public UsersEntity saveNewUser(@RequestBody UsersEntity user) {  return userService.save(user);  }  }  @Table(name = "users", schema = "mydb")  public class UsersEntity {  private int idUsers;  private String login;  private String password;  private String personalData;  //private int idRole;  private RolesEntity idRole;  @Id  @Column(name = "idUsers" )  public int getIdUsers() {  return idUsers;  }  public void setIdUsers(int idUsers) {  this.idUsers = idUsers;  }  @Basic  @Column(name = "login")  public String getLogin() {  return login;  }  public void setLogin(String login) {  this.login = login;  }  @Basic  @Column(name = "password")  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  @Basic  @Column(name = "personalData")  public String getPersonalData() {  return personalData;  }  public void setPersonalData(String personalData) {  this.personalData = personalData;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  UsersEntity that = (UsersEntity) o;  if (idUsers != that.idUsers) return false;  //if (idRole != that.idRole) return false;  if (login != null ? !login.equals(that.login) : that.login != null) return false;  if (password != null ? !password.equals(that.password) : that.password != null) return false;  if (personalData != null ? !personalData.equals(that.personalData) : that.personalData != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idUsers;  result = 31 \* result + (login != null ? login.hashCode() : 0);  result = 31 \* result + (password != null ? password.hashCode() : 0);  result = 31 \* result + (personalData != null ? personalData.hashCode() : 0);  //result = 31 \* result + idRole;  return result;  }  @ManyToOne  @JoinColumn(name = "idRole", referencedColumnName = "idRole", nullable = false)  public RolesEntity getIdRole() {  return idRole;  }  public void setIdRole(RolesEntity idRole) {  this.idRole = idRole;  }  }  @Table(name = "comments", schema = "mydb")  @NamedStoredProcedureQueries({  @NamedStoredProcedureQuery(  name = "getCommentById",  procedureName = "getRoleById",  resultClasses = { RolesEntity.class },  parameters = {  @StoredProcedureParameter(  name = "log",  type = String.class,  mode = ParameterMode.IN)})  })  public class CommentsEntity {  private int idComment;  private int idTask;  private String comment;  private int idUser;  @Id  @Column(name = "idComment")  public int getIdComment() {  return idComment;  }  public void setIdComment(int idComment) {  this.idComment = idComment;  }  @Basic  @Column(name = "idTask")  public int getIdTask() {  return idTask;  }  public void setIdTask(int idTask) {  this.idTask = idTask;  }  @Basic  @Column(name = "comment")  public String getComment() {  return comment;  }  public void setComment(String comment) {  this.comment = comment;  }  @Basic  @Column(name = "idUser")  public int getIdUsers() {  return idUser;  }  public void setIdUsers(int idUsers) {  this.idUser = idUsers;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  CommentsEntity that = (CommentsEntity) o;  if (idComment != that.idComment) return false;  if (idTask != that.idTask) return false;  if (idUser != that.idUser) return false;  if (comment != null ? !comment.equals(that.comment) : that.comment != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idComment;  result = 31 \* result + idTask;  result = 31 \* result + (comment != null ? comment.hashCode() : 0);  result = 31 \* result + idUser;  return result;  }  } |

Приложения В. Листинг Микросервиса FAPI

|  |  |
| --- | --- |
| 8 public class Role {  private int idRole;  private String name;  public int getIdRole() {  return idRole;  }  public void setIdRole(int idRole) {  this.idRole = idRole;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Role that = (Role) o;  if (idRole != that.idRole) return false;  if (name != null ? !name.equals(that.name) : that.name != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idRole;  result = 31 \* result + (name != null ? name.hashCode() : 0);  return result;  }  }  public class Status {  private int idStatus;  private String name;  public int getIdStatus() {  return idStatus;  }  public void setIdStatus(int idStatus) {  this.idStatus = idStatus;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Status that = (Status) o;  if (idStatus != that.idStatus) return false;  if (name != null ? !name.equals(that.name) : that.name != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idStatus;  result = 31 \* result + (name != null ? name.hashCode() : 0);  return result;  }  }  public class Task {  private int idTask;  private String description;  private String estimation;  private Date createDate;  private Date dueDate;  private Date updatedDate;  private Date resolvedDate;  private Date closedDate;  private Project idProject;  //private int idProject;  private Prioritet idPriority;  private Status idStatus;  private User assignee;  private User reporter;  public Task(){  }  public int getIdTask() {  return idTask;  }  public void setIdTask(int idTask) {  this.idTask = idTask;  }  public String getDescription() {  return description;  }  public void setDescription(String description) {  this.description = description;  }  public String getEstimation() {  return estimation;  }  public void setEstimation(String estimation) {  this.estimation = estimation;  }  public Date getCreateDate() {  return createDate;  }  public void setCreateDate(Date createDate) {  this.createDate = createDate;  }  public Date getDueDate() {  return dueDate;  }  public void setDueDate(Date dueDate) {  this.dueDate = dueDate;  }  public Date getUpdatedDate() {  return updatedDate;  }  public void setUpdatedDate(Date updatedDate) {  this.updatedDate = updatedDate;  }  public Date getResolvedDate() {  return resolvedDate;  }  public void setResolvedDate(Date resolvedDate) {  this.resolvedDate = resolvedDate;  }  public Date getClosedDate() {  return closedDate;  }  public void setClosedDate(Date closedDate) {  this.closedDate = closedDate;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Task that = (Task) o;  if (idTask != that.idTask) return false;  if (idProject != that.idProject) return false;  if (idPriority != that.idPriority) return false;  if (idStatus != that.idStatus) return false;  if (assignee != that.assignee) return false;  if (reporter != that.reporter) return false;  if (description != null ? !description.equals(that.description) : that.description != null) return false;  if (estimation != null ? !estimation.equals(that.estimation) : that.estimation != null) return false;  if (createDate != null ? !createDate.equals(that.createDate) : that.createDate != null) return false;  if (dueDate != null ? !dueDate.equals(that.dueDate) : that.dueDate != null) return false;  if (updatedDate != null ? !updatedDate.equals(that.updatedDate) : that.updatedDate != null) return false;  if (resolvedDate != null ? !resolvedDate.equals(that.resolvedDate) : that.resolvedDate != null) return false;  if (closedDate != null ? !closedDate.equals(that.closedDate) : that.closedDate != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idTask;  result = 31 \* result + (description != null ? description.hashCode() : 0);  result = 31 \* result + (estimation != null ? estimation.hashCode() : 0);  result = 31 \* result + (createDate != null ? createDate.hashCode() : 0);  result = 31 \* result + (dueDate != null ? dueDate.hashCode() : 0);  result = 31 \* result + (updatedDate != null ? updatedDate.hashCode() : 0);  result = 31 \* result + (resolvedDate != null ? resolvedDate.hashCode() : 0);  result = 31 \* result + (closedDate != null ? closedDate.hashCode() : 0);  /\*result = 31 \* result + idProject;  result = 31 \* result + idPriority;  result = 31 \* result + idStatus;  result = 31 \* result + assignee;  result = 31 \* result + reporter;\*/  return result;  }  public Prioritet getIdPriority() {  return idPriority;  }  public void setIdPriority(Prioritet prioritiesByIdPriority) {  this.idPriority = prioritiesByIdPriority;  }  public Status getIdStatus() {  return idStatus;  }  public void setIdStatus(Status statusByIdStatus) {  this.idStatus = statusByIdStatus;  }  public Project getIdProject() {  return idProject;  }  public void setIdProject(Project idProject) {  this.idProject = idProject;  }  public User getAssignee() {  return assignee;  }  public void setAssignee(User assignee) {  this.assignee = assignee;  }  public User getReporter() {  return reporter;  }  public void setReporter(User reporter) {  this.reporter = reporter;  }  }  public class User {  private int idUsers;  private String login;  private String password;  private String personalData;  //private int idRole;  private Role idRole;  public User(){  }  public User (int idUsers, String login, String password, String personalData, Role role){  this.idUsers = idUsers;  this.login = login;  this.password = password;  this.personalData = personalData;  this.password = password;  this.idRole = role;  }  public int getIdUsers() {  return idUsers;  }  public void setIdUsers(int idUsers) {  this.idUsers = idUsers;  }  public String getLogin() {  return login;  }  public void setLogin(String login) {  this.login = login;  }  public String getPassword() {  return password;  }  public void setPassword(String password) {  this.password = password;  }  public String getPersonalData() {  return personalData;  }  public void setPersonalData(String personalData) {  this.personalData = personalData;  }  /\* public int getIdRole() {  return idRole;  }  public void setIdRole(int idRole) {  this.idRole = idRole;  }\*/  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  User that = (User) o;  if (idUsers != that.idUsers) return false;  // if (idRole != that.idRole) return false;  if (login != null ? !login.equals(that.login) : that.login != null) return false;  if (password != null ? !password.equals(that.password) : that.password != null) return false;  if (personalData != null ? !personalData.equals(that.personalData) : that.personalData != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idUsers;  result = 31 \* result + (login != null ? login.hashCode() : 0);  result = 31 \* result + (password != null ? password.hashCode() : 0);  result = 31 \* result + (personalData != null ? personalData.hashCode() : 0);  //result = 31 \* result + idRole;  return result;  }  public Role getIdRole() {  return idRole;  }  public void setIdRole(Role idRole) {  this.idRole = idRole;  }  }  @Service  public class PriorityServiceImpl implements PriorityService {  @Value("http://localhost:8082/")  private String backendUrl;  @Override  public List<Prioritet> findAll(){  RestTemplate restTemplate = new RestTemplate();  Prioritet[] priorityResponse = restTemplate.getForObject(backendUrl + "/api/priority", Prioritet[].class);  return priorityResponse == null ? Collections.emptyList() : Arrays.asList(priorityResponse);  }  @Override  public Prioritet getRoleByIdPriority(int idPriority) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/priority/priority/" + idPriority, Prioritet.class);  }  @Override  public Prioritet getRoleByName(String name) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/priority/name/" + name, Prioritet.class);  }  }  @Service  public class ProjectServiceImpl implements ProjectService {  @Value("http://localhost:8082/")  private String backendUrl;  @Override  public List<Project> findAll(){  RestTemplate restTemplate = new RestTemplate();  Project[] projectResponse = restTemplate.getForObject(backendUrl + "/api/projects", Project[].class);  return projectResponse == null ? Collections.emptyList() : Arrays.asList(projectResponse);  }  @Override  public Project getProjectByIdProject(int idProject) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/projects/project/" + idProject, Project.class);  }  @Override  public Project save(Project project) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.postForEntity(backendUrl + "/api/projects", project, Project.class).getBody();  }  }  @Service  public class StatusServiceImpl implements StatusService {  @Value("http://localhost:8082/")  private String backendUrl;  @Override  public List<Status> findAll(){  RestTemplate restTemplate = new RestTemplate();  Status[] statusResponse = restTemplate.getForObject(backendUrl + "/api/status", Status[].class);  return statusResponse == null ? Collections.emptyList() : Arrays.asList(statusResponse);  }  @Override  public Status getStatusByIdStatus(int idStatus) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/status/status/" + idStatus, Status.class);  }  @Override  public Status getStatusByName(String name) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/status/name/" + name, Status.class);  } | 8 public class PriorityController {  @Autowired  private PriorityService priorityService;  @RequestMapping(value = "", method = RequestMethod.GET)  public Iterable<Prioritet> getAllStatus() {  return priorityService.findAll();  }  @RequestMapping(value = "/priority/{idPriority}", method = RequestMethod.GET)  public ResponseEntity<Prioritet> getRoleByIdPriority(@PathVariable(name = "idPriority") int idPriority) {  Prioritet statusEntity = priorityService.getRoleByIdPriority(idPriority);  return ResponseEntity.ok(statusEntity);  }  @RequestMapping(value = "/name/{name}", method = RequestMethod.GET)  public ResponseEntity<Prioritet> getRoleByName(@PathVariable(name = "name") String name) {  Prioritet statusEntity = priorityService.getRoleByName(name);  return ResponseEntity.ok(statusEntity);  }  }  public class RoleController {  @Autowired  private RoleService roleService;  @RequestMapping(value = "", method = RequestMethod.GET)  public Iterable<Role> getAllStatus() {  return roleService.findAll();  }  @RequestMapping(value = "/role/{idRole}", method = RequestMethod.GET)  public ResponseEntity<Role> getRoleByIdRole(@PathVariable(name = "idRole") int idRole) {  Role statusEntity = roleService.getRoleByIdRole(idRole);  return ResponseEntity.ok(statusEntity);  }  @RequestMapping(value = "/name/{name}", method = RequestMethod.GET)  public ResponseEntity<Role> getRoleByName(@PathVariable(name = "name") String name) {  Role statusEntity = roleService.getRoleByName(name);  return ResponseEntity.ok(statusEntity);  }  }  public class ProjectController {  @Autowired  private ProjectService projectService;    @RequestMapping(value = "/project/{idProject}", method = RequestMethod.GET)  public Project getProjectByIdProject(@PathVariable(name = "idProject") int idProject) {  return projectService.getProjectByIdProject(idProject);  }  @RequestMapping(value = "", method = RequestMethod.GET)  public List<Project> getAllProjects() {  return projectService.findAll();  }  @RequestMapping(method = RequestMethod.POST)  public Project saveProject(@RequestBody Project projectsEntity) {  return projectService.save(projectsEntity);  }  }  public class StatusController {  @Autowired  private StatusService statusService;  @RequestMapping(value = "", method = RequestMethod.GET)  public Iterable<Status> getAllStatus() {  return statusService.findAll();  }  @RequestMapping(value = "/status/{idStatus}", method = RequestMethod.GET)  public ResponseEntity<Status> getStatusByIdStatus(@PathVariable(name = "idStatus") int idStatus) {  Status statusEntity = statusService.getStatusByIdStatus(idStatus);  return ResponseEntity.ok(statusEntity);  }  @RequestMapping(value = "/name/{name}", method = RequestMethod.GET)  public ResponseEntity<Status> getStatusByName(@PathVariable(name = "name") String name) {  Status statusEntity = statusService.getStatusByName(name);  return ResponseEntity.ok(statusEntity);  }  }  public class TaskController {  @Autowired  private TaskService taskService;  @GetMapping(value="")  public List<Task> getAllTask(){  return taskService.findAll();  }  @GetMapping("/idTask/{idTask}")  public Task getTaskByIdTask(@PathVariable int idTask) {  return taskService.getTaskByIdTask(idTask);  }  @PatchMapping("")  public Task updateTask(@RequestBody Task taskModel) {  {  return taskService.saveTask(taskModel);  }  }  @RequestMapping(method = RequestMethod.POST, produces = "application/json")  public Task saveTask(@RequestBody Task task){  return taskService.saveTask(task);  }  }  public class UserController {  @Autowired  private UsersService usersService;  @GetMapping(value="")  public List<User> getAllUsers(){  return usersService.findAll();  }  @GetMapping("/login/{login}")  public User getUserByLogin(@PathVariable String login) {  return usersService.getUserByLogin(login);  }  @RequestMapping(method = RequestMethod.POST, produces = "application/json")  public User saveUser(@RequestBody User user){  return usersService.saveUser(user);  }  }  public class Attachment {  private int idAttachment;  private int idTask;  private String path;  public int getIdAttachment() {  return idAttachment;  }  public void setIdAttachment(int idAttachment) {  this.idAttachment = idAttachment;  }  public int getIdTask() {  return idTask;  }  public void setIdTask(int idTask) {  this.idTask = idTask;  }  public String getPath() {  return path;  }  public void setPath(String path) {  this.path = path;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Attachment that = (Attachment) o;  if (idAttachment != that.idAttachment) return false;  if (idTask != that.idTask) return false;  if (path != null ? !path.equals(that.path) : that.path != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idAttachment;  result = 31 \* result + idTask;  result = 31 \* result + (path != null ? path.hashCode() : 0);  return result;  }  }  public class Comment {  private int idComment;  private int idTask;  private String comment;  private int idUser;  public int getIdComment() {  return idComment;  }  public void setIdComment(int idComment) {  this.idComment = idComment;  }  public int getIdTask() {  return idTask;  }  public void setIdTask(int idTask) {  this.idTask = idTask;  }  public String getComment() {  return comment;  }  public void setComment(String comment) {  this.comment = comment;  }  public int getIdUsers() {  return idUser;  }  public void setIdUsers(int idUsers) {  this.idUser = idUsers;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Comment that = (Comment) o;  if (idComment != that.idComment) return false;  if (idTask != that.idTask) return false;  if (idUser != that.idUser) return false;  if (comment != null ? !comment.equals(that.comment) : that.comment != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idComment;  result = 31 \* result + idTask;  result = 31 \* result + (comment != null ? comment.hashCode() : 0);  result = 31 \* result + idUser;  return result;  }  }  public class Prioritet {  private int idPriority;  private String name;  public int getIdPriority() {  return idPriority;  }  public void setIdPriority(int idPriority) {  this.idPriority = idPriority;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Prioritet that = (Prioritet) o;  if (idPriority != that.idPriority) return false;  if (name != null ? !name.equals(that.name) : that.name != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idPriority;  result = 31 \* result + (name != null ? name.hashCode() : 0);  return result;  }  }  public class Project {  private int idProject;  private String summary;  private User reporter;  public int getIdProject() {  return idProject;  }  public void setIdProject(int idProject) {  this.idProject = idProject;  }  public String getSummary() {  return summary;  }  public void setSummary(String summary) {  this.summary = summary;  }  public User getReporter() {  return reporter;  }  public void setReporter(User reporter) {  this.reporter = reporter;  }  @Override  public boolean equals(Object o) {  if (this == o) return true;  if (o == null || getClass() != o.getClass()) return false;  Project that = (Project) o;  if (idProject != that.idProject) return false;  if (reporter != that.reporter) return false;  if (summary != null ? !summary.equals(that.summary) : that.summary != null) return false;  return true;  }  @Override  public int hashCode() {  int result = idProject;  result = 31 \* result + (summary != null ? summary.hashCode() : 0);  // result = 31 \* result + reporter;  return result;  }  }  @Service  public class TaskServiceImlp implements TaskService {  @Value("http://localhost:8082/")  private String backendServerUrl;  @Override  public Task getTaskByIdTask(int idTask) {  RestTemplate restTemplate = new RestTemplate();  Task task = restTemplate.getForObject(backendServerUrl + "/api/task/idTask/" + idTask, Task.class);  return task;  }  @Override  public List<Task> findAll() {  RestTemplate restTemplate = new RestTemplate();  Task[] taskResponse = restTemplate.getForObject(backendServerUrl + "/api/task", Task[].class);  return taskResponse == null ? Collections.emptyList() : Arrays.asList(taskResponse);  }  @Override  public Task saveTask(Task task) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.postForEntity(backendServerUrl + "/api/task", task, Task.class).getBody();  }  }  @Service  public class UsersServiceImpl implements UsersService {  @Value("http://localhost:8082/")  private String backendUrl;  @Override  public User getUserByLogin(String login) {  RestTemplate restTemplate = new RestTemplate();  User user = restTemplate.getForObject(backendUrl + "/api/user/login/" + login, User.class);  return user;  }  @Override  public List<User> findAll() {  RestTemplate restTemplate = new RestTemplate();  User[] usersResponse = restTemplate.getForObject(backendUrl + "/api/user", User[].class);  return usersResponse == null ? Collections.emptyList() : Arrays.asList(usersResponse);  }  @Override  public User saveUser(User user) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.postForEntity(backendUrl + "/api/user", user, User.class).getBody();  }  @Service  public class RoleServiceImpl implements RoleService {  @Value("http://localhost:8082/")  private String backendUrl;  @Override  public List<Role> findAll(){  RestTemplate restTemplate = new RestTemplate();  Role[] roleResponse = restTemplate.getForObject(backendUrl + "/api/role", Role[].class);  return roleResponse == null ? Collections.emptyList() : Arrays.asList(roleResponse);  }  @Override  public Role getRoleByIdRole(int idRole) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/role/role/" + idRole, Role.class);  }  @Override  public Role getRoleByName(String name) {  RestTemplate restTemplate = new RestTemplate();  return restTemplate.getForObject(backendUrl + "/api/role/name/" + name, Role.class);  }  }  } |