**Санкт-Петербургский политехнический университет Петра Великого**

**Институт компьютерных наук и технологий**

**Кафедра компьютерных систем и программных технологий**

**Курсовая работа**

**Реализация приложения «Бистро Шаверма» с помощью «Spring Framework»**

по дисциплине «ПО распределённых вычислительных систем»

Работу выполнил Карандашов С.М.  
студент группы 25341/3

Работу принял Стручков И.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2019г.

Санкт-Петербург

2019

**Оглавление**

Оглавление

[Анализ задания 3](#__RefHeading___Toc1893_2200388220)

[1. Формулировка индивидуального задания 3](#__RefHeading___Toc1895_2200388220)

[2. Требования, которым должно удовлетворять приложение 3](#__RefHeading___Toc1897_2200388220)

[3. Основные варианты использования 4](#__RefHeading___Toc1899_2200388220)

[4. Описание модели предметной области 5](#__RefHeading___Toc1901_2200388220)

[Вовлеченные Сущности 5](#__RefHeading___Toc1903_2200388220)

[Реализация задания с помощью «Spring Framework» 6](#__RefHeading___Toc1905_2200388220)

[1. Объектно-ориентированное проектирование с учётом особенностей технологии 6](#__RefHeading___Toc1907_2200388220)

[Диаграмма последовательности 7](#__RefHeading___Toc1909_2200388220)

[2. Приложение-клиент 7](#__RefHeading___Toc1911_2200388220)

[3. Полный текст программы 8](#__RefHeading___Toc1913_2200388220)

[4. Тестирование 9](#__RefHeading___Toc1915_2200388220)

[5. Инструкция системному администратору по развёртыванию приложения 9](#__RefHeading___Toc1917_2200388220)

[6. Инструкция пользователю по запуску приложения 9](#__RefHeading___Toc1919_2200388220)

[Вывод 10](#__RefHeading___Toc1921_2200388220)

[Литература 11](#__RefHeading___Toc1923_2200388220)

[Приложение 12](#__RefHeading___Toc1925_2200388220)

**Анализ задания**

1. **Формулировка индивидуального задания**

Реализовать приложение «Бистро Шаверма» с помощью Spring Framework

1. **Требования, которым должно удовлетворять приложение**

Приложение должно строиться, используя следующие принципы и инструменты:

* Spring Framework
* Rest
* Паттерны

1. **Основные варианты использования**

**1. Создание заказа**

1. Клиент входит в систему как клиент
2. Клиент генерирует заказ
3. Клиент подтверждает заполненную информацию (оформляет заказ)

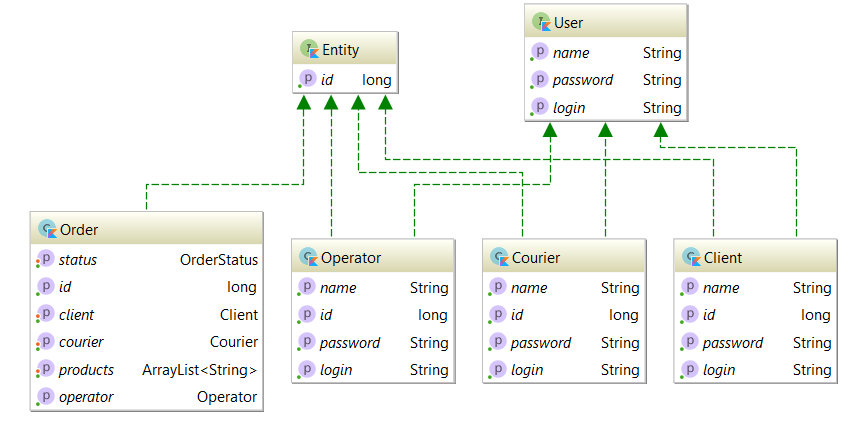
**2. Просмотр заказа оператором**

1. Оператор выполняет вход в систему
2. Оператор просматривает список заказов, которые ожидают его подтверждения
3. Оператор выбирает заказ для подтверждения
4. Оператор подтверждает заказ.
5. Оператор меняет статус на доставку
6. Оператор подтверждает выполнение заказа

Альтернатива 1 В пункте 4 клиент отменяет заказ. В таком случае заказ удаляется, переход к пункту 2.

**3. Доставка**

1. Курьер выполняет вход в систему
2. Курьер просматривает список заказов, которые готовы и ожидают доставки
3. Курьер выбирает заказы, которые хочет доставить
4. Курьер смотрит информацию
5. Курьер подтверждает доставку.
6. **Описание модели предметной области**

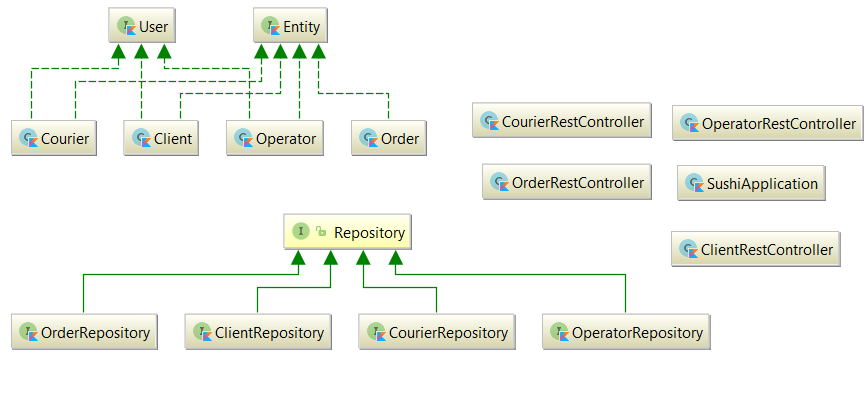


**Вовлеченные Сущности**

В результате разработки вариантов использования были выделены следующие вовлеченные сущности: Клиент, оператор, курьер, сам заказ.

**Реализация задания с помощью «Spring Framework»**

1. **Объектно-ориентированное проектирование с учётом особенностей технологии**



@SpringBootApplication – сканирует все контроллеры, сервисы и прочие составляющие спринга, чтобы все это вместе запустить

Application – класс запуска программы

@RestController – принимает запросы пользователя и возвращает данные в JSON формате

OrderRestController – контроллер для работы с сущностью заказа

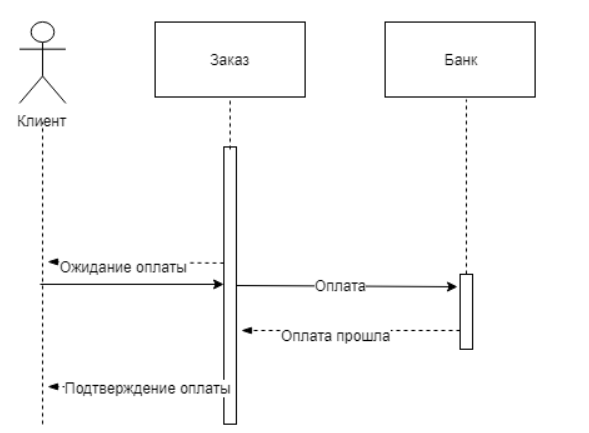
@Entity – дает знать спрингу, что это часть кода – сущность, которую мы сохраняем в БД

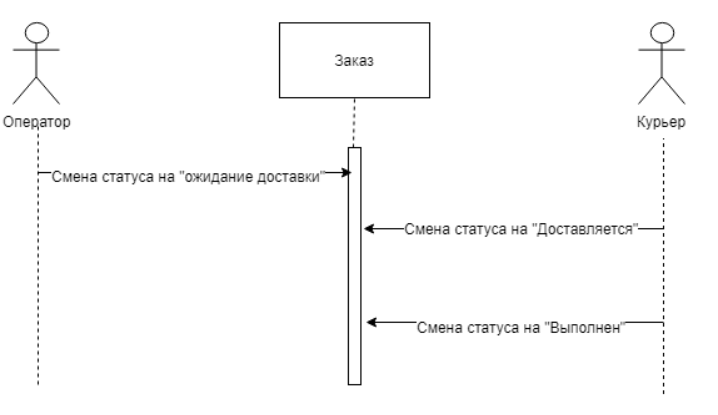
Order – класс заказа

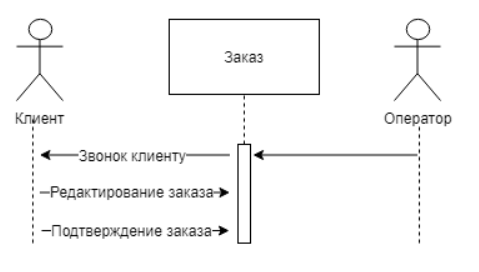
User – класс пользователя

CrudRepository – обеспечивает сложные функциональные возможности CRUD для класса сущностей

**Диаграмма последовательности**







1. **Полный текст программы**

См. Приложение.

1. **Тестирование**

|  |  |  |
| --- | --- | --- |
| **Вариант тестирования** | **Ожидаемый результат** | **Фактический результат** |
| Регистрация нового пользователя | Добавлен новый пользователь | Совпадает с ожидаемым |
| Нельзя использовать уже занятый логин | Совпадает с ожидаемым |
| Все поля формы регистрации должны пройти валидацию | Совпадает с ожидаемым |
| Авторизация | Уведомить, если логин или пароль не прошли валидацию | Совпадает с ожидаемым |
| Роль Оператора | Оператор может создать заказ | Совпадает с ожидаемым |
| Оператор может подтвердить заказ | Совпадает с ожидаемым |
| Роль Курьера | Курьер может взять заказы, готовые к отправке | Совпадает с ожидаемым |
| Роль Клиента | Клиент может создать заказ | Совпадает с ожидаемым |

1. **Инструкция системному администратору по развёртыванию приложения**
2. Скачать архив eda
3. Запустить файл run.bat
4. **Инструкция пользователю по запуску приложения**
5. Зарегистрироваться в системе
6. Войти в систему
7. Оформить заказ
8. Дождаться выполнения и доставки заказа

**Вывод**

В результате работы на курсовым проектом было спроектировано и разработано приложение «Бистро Шаверма». В процессе проектирования были закреплены на практике знания о Spring Framework.

Для реализации проекта использовался Spring Boot.

Степень выполнения поставленного задания составляет 60%. Текущая реализация требует улучшения и некоторых доработок по следующим направлениям:

* Надежность данных при авторизации
* Расширения функциональности для пользователей

В текущей реализации продемонстрированы такие фундаментальные навыки как построение клиент-серверной системы, организация взаимодействия компонентов посредством REST API.

**Литература**

1. Фаулер М. UML. Основы, 3-е издание. Символ-Плюс, 2006.

2. Гранд М. Шаблоны проектирования в Java. BHV-СПб, 2004.

3. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования, 3-е издание. Вильямс, 2007.

4. Гайд по разработке приложений на Spring - https://spring.io/guides

**Приложение**

|  |
| --- |
| package com.example.eda  import org.springframework.boot.autoconfigure.SpringBootApplication import org.springframework.boot.runApplication  @SpringBootApplication class EdaApplication  fun main(args: Array<String>) {  *runApplication*<EdaApplication>(\*args) } |

|  |
| --- |
| package com.example.eda.controller  import com.example.eda.model.api.LoginAndPassword import com.example.eda.model.storage.Client import com.example.eda.repository.ClientRepository import com.example.eda.util.extension.ElementAlreadyExists import com.example.eda.util.extension.ElementNotFound import org.springframework.web.bind.annotation.\*  @RestController @RequestMapping("/api/clients") class ClientRestController(private val clients: ClientRepository) {   @GetMapping("/all")  fun all(): Iterable<Client> = clients.findAll().*toList*()   @PostMapping("/create")  fun create(@RequestBody client: Client): Any = when {  clients.findAll().*any* **{ it**.login == client.login **}** -> *ElementAlreadyExists*()  else -> clients.save(client)  }   @PostMapping("/sign")  fun signIn(@RequestBody loginAndPassword: LoginAndPassword): Any {  val (login, password) = loginAndPassword  return clients.findAll().*find* **{ it**.login == login && **it**.password == password **}** ?: *ElementNotFound*()  }   @GetMapping("/{id}")  fun get(@PathVariable(value = "id") id: Long): Any = try {  clients.findById(id).get()  } catch (e: Exception) {  *ElementNotFound*()  }   @DeleteMapping("/{id}")  fun delete(@PathVariable(value = "id") id: Long) = try {  clients.deleteById(id)  } catch (e: Exception) {  *ElementNotFound*()  } } |

|  |
| --- |
| package com.example.eda.controller  import com.example.eda.model.api.LoginAndPassword import com.example.eda.model.storage.Courier import com.example.eda.repository.CourierRepository import com.example.eda.util.extension.ElementAlreadyExists import com.example.eda.util.extension.ElementNotFound import org.springframework.web.bind.annotation.\*  @RestController @RequestMapping("/api/couriers") class CourierRestController(private val couriers: CourierRepository) {   @GetMapping("/all")  fun all() = couriers.findAll()   @PostMapping("/create")  fun create(@RequestBody courier: Courier): Any = when {  couriers.findAll().*any* **{ it**.login == courier.login **}** -> *ElementAlreadyExists*()  else -> couriers.save(courier)  }   @PostMapping("/sign")  fun signIn(@RequestBody loginAndPassword: LoginAndPassword): Any {  val (login, password) = loginAndPassword  return couriers.findAll().*find* **{ it**.login == login && **it**.password == password **}** ?: *ElementNotFound*()  }   @GetMapping("/{id}")  fun get(@PathVariable(value = "id") id: Long): Any = try {  couriers.findById(id).get()  } catch (e: Exception) {  *ElementNotFound*()  }   @DeleteMapping("/{id}")  fun delete(@PathVariable(value = "id") id: Long) = try {  couriers.deleteById(id)  } catch (e: Exception) {  *ElementNotFound*()  } } |

|  |
| --- |
| package com.example.eda.controller  import com.example.eda.model.api.LoginAndPassword import com.example.eda.model.storage.Operator import com.example.eda.repository.OperatorRepository import com.example.eda.util.extension.ElementAlreadyExists import com.example.eda.util.extension.ElementNotFound import org.springframework.web.bind.annotation.\*  @RestController @RequestMapping("/api/operators") class OperatorRestController(private val operators: OperatorRepository) {   @GetMapping("/all")  fun all() = operators.findAll()   @PostMapping("/create")  fun create(@RequestBody operator: Operator): Any = when {  operators.findAll().*any* **{ it**.login == operator.login **}** -> *ElementAlreadyExists*()  else -> operators.save(operator)  }   @PostMapping("/sign")  fun signIn(@RequestBody loginAndPassword: LoginAndPassword): Any {  val (login, password) = loginAndPassword  return operators.findAll().*find* **{ it**.login == login && **it**.password == password **}** ?: *ElementNotFound*()  }   @GetMapping("/{id}")  fun get(@PathVariable(value = "id") id: Long): Any = try {  operators.findById(id).get()  } catch (e: Exception) {  ElementNotFound()  }   @DeleteMapping("/{id}")  fun delete(@PathVariable(value = "id") id: Long) = try {  operators.deleteById(id)  } catch (e: Exception) {  ElementNotFound()  } } |

|  |
| --- |
| package com.example.eda.controller  import com.example.eda.model.mapper.toApi import com.example.eda.model.mapper.toDomain import com.example.eda.repository.\* import com.example.eda.util.extension.ElementAlreadyExists import com.example.eda.util.extension.ElementNotFound import org.springframework.web.bind.annotation.\* import com.example.eda.model.api.Order as ApiRequest  @RestController @RequestMapping("/api/orders") class OrderRestController(  private val orders: OrderRepository,  private val couriers: CourierRepository,  private val clients: ClientRepository,  private val operators: OperatorRepository ) {   @GetMapping("/all")  fun all() = orders.findAll().map **{** it.toApi() **}** @PostMapping("/create")  fun create(@RequestBody request: ApiRequest): Any = when {  orders.existsById(request.id) -> ElementAlreadyExists()  else -> orders.save(request.toDomain(couriers, clients, operators)).toApi()  }   @GetMapping("/{id}")  fun get(@PathVariable(value = "id") id: Long): Any = try {  orders.findById(id).get().toApi()  } catch (e: Exception) {  ElementNotFound()  }   @DeleteMapping("/{id}")  fun delete(@PathVariable(value = "id") id: Long) = try {  orders.deleteById(id)  } catch (e: Exception) {  ElementNotFound()  }   @PostMapping("/change")  fun update(@RequestBody request: ApiRequest): Any = when {  orders.existsById(request.id) -> {  val entity = orders.findById(request.id).get()  entity.courier = request.courierId?.let **{** couriers.findById(it).get() **}** entity.status = request.status  entity.products = request.products  orders.save(entity)  }  else -> ElementNotFound()  } } |

|  |
| --- |
| package com.example.eda.repository  import com.example.eda.model.storage.Client import org.springframework.data.repository.CrudRepository import org.springframework.stereotype.Repository  @Repository interface ClientRepository : CrudRepository<Client, Long> |

|  |
| --- |
| package com.example.eda.model.mapper  import com.example.eda.model.storage.Order import com.example.eda.repository.ClientRepository import com.example.eda.repository.OperatorRepository import com.example.eda.repository.CourierRepository import com.example.eda.model.api.Order as ApiOrder  fun Order.toApi() = ApiOrder(  id = id,  status = status,  products = products,  operatorId = operator?.id,  clientId = client.id,  courierId = courier?.id )  fun ApiOrder.toDomain(couriers: CourierRepository, clients: ClientRepository, operators: OperatorRepository) = Order(  id = id,  status = status,  products = products,  operator = operatorId?.*let* **{** operators.findById(**it**).get() **}**,  client = clientId.*let* **{** clients.findById(**it**).get() **}**,  courier = courierId?.*let* **{** couriers.findById(**it**).get() **}** ) |

package com.example.eda.model.storage  
  
import javax.persistence.\*  
import com.example.eda.model.storage.Entity as DbEntity  
  
@Entity  
@Table(name = "orderrequest")  
data class Order(  
 @Id @GeneratedValue(strategy = GenerationType.IDENTITY)  
 override val id: Long = 0,  
 var status: OrderStatus = OrderStatus.WAIT\_CONFIRM,  
  
 @Basic  
 var products: ArrayList<String> = ArrayList(),  
  
 @ManyToOne @JoinColumn(name = "operator\_id")  
 val operator: Operator? = null,  
  
 @ManyToOne @JoinColumn(name = "courier\_id")  
 var courier: Courier? = null,  
  
 @ManyToOne @JoinColumn(name = "client\_id")  
 var client: Client  
) : DbEntity  
  
  
enum class OrderStatus {  
 WAIT\_CONFIRM, CONFIRMED, WAIT\_DELIVERY, IN\_DELIVERY, COMPLETE, CANCELED  
}