Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по курсу «Бизнес-логика программных систем»

Вариант 12

Выполнили:

Батманов Д.Е.

Сафонова О.Д.

Преподаватели:

Цопа Е.А.

Кривоносов Е.Д.

Санкт-Петербург

2025

Оглавление

Задание

Описать бизнес-процесс в соответствии с нотацией BPMN 2.0, после чего реализовать его в виде приложения на базе Spring Boot.

Вариант №12:
OZON маркетплейс — миллионы товаров по выгодным ценам — https://www.ozon.ru.
Бизнес-процесс: создание и редактирование отзывов на товары и продавцов.

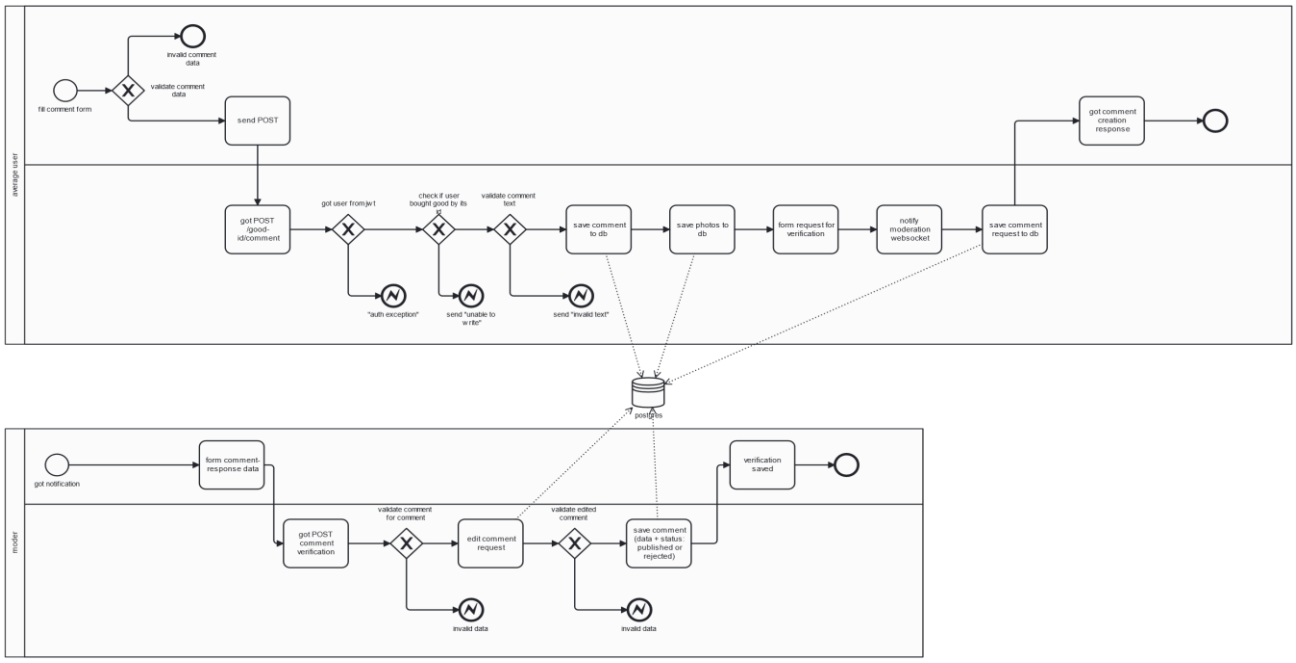
Требования к реализации

- Специфицировать модель реализуемого бизнес-процесса в соответствии с требованиями BPMN 2.0
- Разработать приложение на базе Spring Boot
- Приложение должно использовать СУБД PostgreSQL
- Для всех публичных интерфейсов должны быть разработаны REST API
- Разработать набор curl-скриптов для тестирования публичных интерфейсов разработанного программного модуля
- Развернуть разработанное приложение на сервере helios

Выполнение

BPMN

Спецификация бизнес-процесса создания и редактирования отзывов на товары описана в виде нотации и модели бизнес-процесса, в виде диаграммы:



Программная реализация

Для сущности уровня базы данных был создан следующий класс:

```
package web.ozon.entity;

import jakarta.annotation.Nonnull;
import jakarta.persistence.*;
import org.hibernate.annotations.ColumnDefault;
import org.hibernate.annotations.Where;
import lombok.*;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "comments")
@Where(clause = "is_deleted = false")
public class CommentEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Nonnull
    @ManyToOne
    @JoinColumn(name = "product_id")
    private ProductEntity product;

    @Nonnull
    @ManyToOne
    @JoinColumn(name = "author_id")
    private UserEntity author;

    @Nonnull
    @Column(name = "content")
    private String content;

    @Nonnull
    @Column(name = "is_anonymous")
    @ColumnDefault("false")
    private Boolean isAnonymous;

    @Nonnull
    @Column(name = "is_deleted")
    @ColumnDefault("false")
    private Boolean isDeleted;

    @Nonnull
    @Column(name = "is_checked")
    @ColumnDefault("false")
    private Boolean isChecked;
}
```

Контроллер для API взаимодействия с комментариями использует следующие end-points:

```
GET: /comments/{productId}/{from}/{to}
POST: /comments
PUT: /comments/{id}
DELETE: /comments/{id}
```

Для сообщения с пользовательской частью предоставлен интерфейс класса CommentEntity – CommentDTO. Бизнес-логика описана в классе CommentService. Кроме этого, настроен доступ к end-point'ам по ролям с помощью Spring Security.

Ниже приведена часть класса CommentsController, метод getComments:

```
   @PermitAll
   @GetMapping("/{productId}/{from}/{to}")
   public ResponseEntity<List<CommentDTO>> getComments(@PathVariable Long
productId,
           @PathVariable Integer from,
           @PathVariable Integer to) {
       return new ResponseEntity<>(commentService.getAllByProductId(productId, from, to),
HttpStatus.OK);
   }
```

В запросах POST и PUT в теле запроса ожидается объект класса CommentDTO:

```
   @PreAuthorize("hasAnyAuthority('USER')")
   @PostMapping
   public ResponseEntity<CommentDTO> postComment(@RequestBody CommentDTO
commentDTO) {
       CommentDTO result = commentService.save(commentDTO);
       return new ResponseEntity<>(result, result != null ? HttpStatus.CREATED :
HttpStatus.BAD_REQUEST);
   }

   @PreAuthorize("hasAnyAuthority('USER')")
   @PutMapping("/{id}")
   public ResponseEntity<CommentDTO> updateComment(@PathVariable Long id,
@RequestBody CommentDTO commentDTO) {
       if (!id.equals(commentDTO.getId())) {
           return ResponseEntity.badRequest().build();
       }
       CommentDTO result = commentService.update(commentDTO);
       return result != null ? ResponseEntity.ok(result) : ResponseEntity.badRequest().build();
   }
```

Так как бизнес-логика маркетплейса подразумевает ручную проверку отзывов, была создана сущность уровня базы данных CommentRequestEntity:

```
package web.ozon.entity;

import org.hibernate.annotations.ColumnDefault;
import org.hibernate.annotations.Where;

import jakarta.annotation.Nonnull;
import jakarta.persistence.*;
import jakarta.validation.constraints.Null;
import lombok.*;

@Setter
@Getter
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "comment_requests")
@Where(clause = "is_deleted = false")
public class CommentRequestEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Long id;

    @Nonnull
    @ManyToOne
    @JoinColumn(name = "comment_id")
    private CommentEntity comment;

    @Null
    @Column(name = "is_checked")
    private Boolean isChecked;

    @Null
    @ManyToOne
    @JoinColumn(name = "checker_id")
    private UserEntity checker;

    @Nonnull
    @Column(name = "is_deleted")
    @ColumnDefault("false")
    private Boolean isDeleted;
}
```

Сообщение с пользовательской частью запросов на комментарии происходит с помощью CommentRequestController, каждый метод которого помечается аннотацией *@PreAuthorize("hasAnyAuthority('ADMIN')")* в связи с установленной бизнес-логикой.

Контроллер для API взаимодействия с заявками комментариев использует следующие end-points:

```
GET: /comment-requests/{from}/{to}
GET: /comment-requests/unchecked/{from}/{to}
PUT: /comment-requests/{id}
DELETE: /comment-requests/{id}
```

POST-запрос отсутствует в связи с автоматическим созданием заявки при создании комментария:

```
@Transactional
   public CommentDTO save(CommentDTO commentDTO) {
      if (!isCommentOk(commentDTO))
         return null;
      CommentEntity commentEntity = commentConverter.fromDTO(commentDTO);
      commentRepository.save(commentEntity);
      CommentDTO result = commentConverter.fromEntity(commentEntity);
      commentRequestService.createRequest(result);
      return result;
   }
```

Данный метод находится в классе CommentService (бизнес-логика обработки комментариев) использует аннотацию @Transactional в связи с необходимостью создавать заявку только на удачно добавленный в базу данных комментарий.

Работа с базой данных реализуется с помощью JPA и Hibernate и настраивается в файле application.properties.

Сборка приложения происходит с помощью maven, поэтому подключение зависимостей описывается в pom.xml:

```
    <dependency>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

   <dependency>
       <groupId>org.hibernate</groupId>
       <artifactId>hibernate-core</artifactId>
       <version>6.1.4.Final</version>
    </dependency>

    <dependency>
       <groupId>org.postgresql</groupId>
       <artifactId>postgresql</artifactId>
       <version>42.6.0</version>
    </dependency>
```

Реализация скриптов для тестирования

В связи с использованием Spring Security и JWT, с помощью которого происходило сообщение авторизированного пользователя с приложением, необходимо хранить токен авторизации. Например, получение токена для тестового юзера с логином 1 и паролем 1:

```
LOGIN_RESPONSE=$(curl -s -X GET "http://localhost:18018/auth/log/1/1")
TOKEN=$(echo "$LOGIN_RESPONSE" | jq -r '.token')
```

Ниже приведен фрагмент скрипта для тестирования REST API взаимодействия пользователя с комментариями:

```
# 3. Создание комментария
COMMENT_RESPONSE=$(curl -s -X POST "http://localhost:18018/comments" \
  -H "Cookie: token=$TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "product": {
      "id": 2,
      "owner": { "id": 2, "login": "log" }
    },
    "author": { "id": 3, "login": "1" },
    "content": "hi",
    "isAnonymous": false,
    "isDeleted": false
  }')
COMMENT_ID=$(echo "$COMMENT_RESPONSE" | jq -r '.id')
```

Объект комментария представлен в формате JSON, передаётся с помощью флага -d, а токен передаётся посредством Cookie. Тестовый объект записан непосредственно в параметрах curl. Curl использует POST запрос и url: "http://localhost:18018/comments". Порт изменён со стандартного на 18018 во избежание конфликтов при запуске на сервере helios.

Кроме curl-скриптов для манипулирования комментариями, написаны скрипты для взаимодействия с заявками комментариев:

```
# 9. Обновление статуса запроса
curl -X PUT "http://localhost:18018/comment-requests/6" \
  -H "Cookie: token=$TOKEN" \
  -d '{
    "comment": {
      "id": 6,
      "product": { "id": 2, "owner": { "id": 2 }, "content": "hi" },
      "author": { "id": 3 },
      "content": "hi",
      "isAnonymous": false
    },
    "isChecked": true,
    "checker": { "id": 4 },
    "isDeleted": true
  }'
```

Деплой на сервере helios

Ход деплоя:

1. Подготовили application.properties для работы на удаленном сервере (изменили подключение к БД и указали порт сервера)

   spring.datasource.url=jdbc:postgresql://pg:5432/studs
   spring.datasource.username=s******
   spring.datasource.password= “~/pgpass”
   spring.datasource.driver-class-name=org.postgresql.Driver

   spring.jpa.hibernate.ddl-auto=update
   spring.jpa.show-sql=false
   server.port=18018

2. Собрали исполняемый файл при помощи mvn clean package –DskipTests
3. Полученный артефакт перенесли на helios
4. Переподключились к helios следующим образом: ssh -p 2222 s******@helios.cs.ifmo.ru -L 8080:helios.cs.ifmo.ru:18018
5. Запустили .jar файл
6. Перешли при помощи psql -h pg -d studs в БД и заполнили БД тестовыми данными
7. Теперь можно взаимодействовать с API по адресу localhost:8080

Исходный код

Исходный код приведён по ссылке: https://github.com/snOlga/web3.0

Где

CodeAxeAttacks – Батманов Д.Е.

snOlga – Сафонова О.Д.

Вывод

В процессе выполнения лабораторной работы была сформирована (описана с помощью спецификации BPMN) и реализована бизнес-логика создания и редактирования комментариев маркетплейса. В данной работе реализованы аспекты взаимодействия с бизнес-логикой посредством REST API, взаимодействия (хранения, чтения, добавления и *мягкого* удаления) с СУБД PostgreSQL. Для тестирования бизнес-логики был реализован bash-скрипт с использованием программы curl. Приложение может быть развернуто и запущено на сервере helios, работая под портом 18018.

В процессе работы была изучена спецификация BPMN для описания бизнес-процесса. На основе составленной и утверждённой преподавателем спецификации реализовывалась бизнес-логика приложения.

Были изучены некоторые способы тестирования end-point’ов: с помощью swagger-ui, с помощью postman и insomnia, с помощью программы curl. Для программы curl был найден способ передачи требуемых в реализации API данных (тела запроса, Cookie).