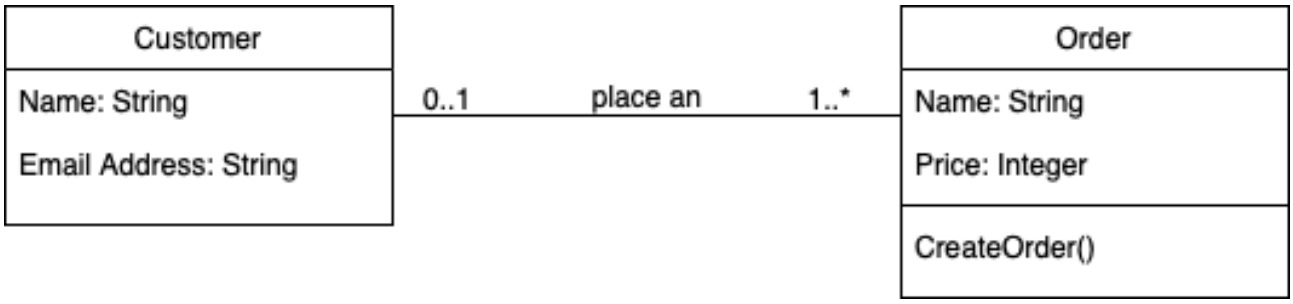
ДЗ: Сервис оформления заказов для покупателей.

Стек технологий:

- Spring Boot 2.3.4
- Gradle
- Java 11
- Spring Web
- Spring Data JDBC
- H2 Database (in memory)
- Lombok
- JUnit 5 (подключается вручную в gradle)
- Mockito (подключается вручную в gradle)
- Spring Boot Test
- Postman (manual testing)
- https://start.spring.io (для генерации проекта)

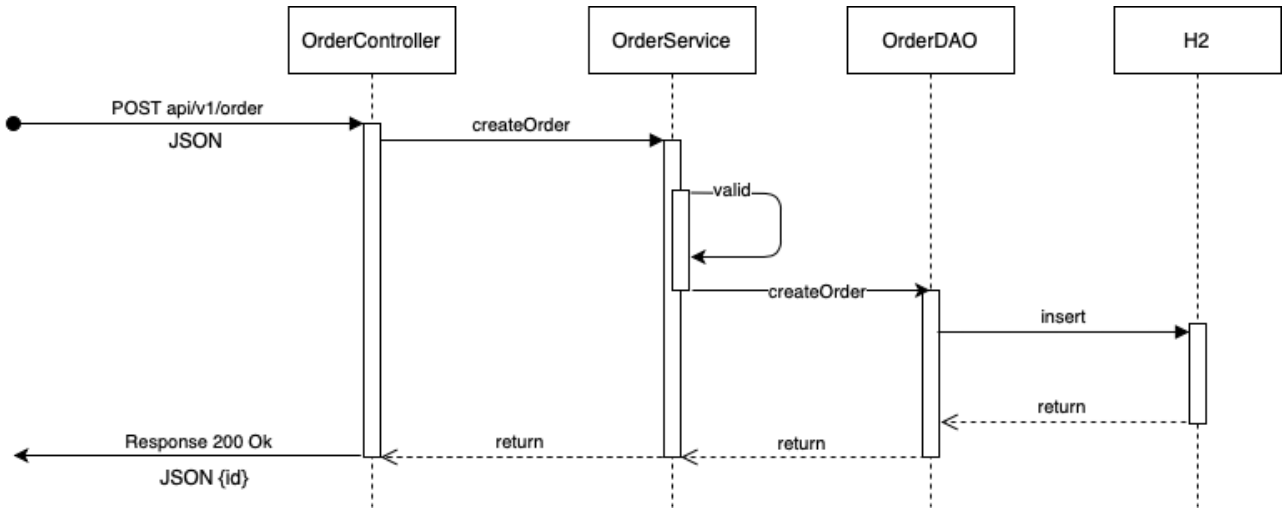
Цель:
- Создать REST сервис с помощью которого можно будет оформлять заказы для покупателей.
- Покупатель одним запросом может оформить только один заказ.
- Необходимо реализовать только добавление нового заказа.
- Должны быть написаны Unit-тесты для уровня DAO, Service, Controller.
- Должны быть написаны Integration-тесты для Controller.
- Для того чтобы вручную проверить, что сервис работает, воспользуйтесь Postman утилитой.
- Залейте полученный результат в ваш GitHub репозиторий в папку lesson4

Логическая модель данных:



Примечание: Для инициализации БД не забудьте воспользоваться, например, ApplicationRunner классом у Spring, либо spring.datasource.initialization-mode=always, либо любым другим удобным для вас способом инициализации БД в Spring Boot.

Диаграмма последовательности:



Примеры тестов:

Unit-тесты (JUnit, Mockito):

```
@Test
@DisplayName("Создание нового пользователя")
public void testSaveOk() {
    UserEditDto userDto = new UserEditDto();
    userDto.setLogin("this login is available");
    userDto.setSubdivision(List.of(new SelectsItem<>(UUID.randomUUID(),  name: "name")));
    when(userRepository.findAll(ArgumentMatchers.<~>any())).thenReturn(Collections.emptyList());
    User converted = new User();
    converted.setId(UUID.randomUUID());

    when(userRepository.save(any(User.class))).thenReturn(converted);

    ResponseEntity answer = userService.save(userDto);
    assertEquals(answer.getStatusCodeValue(),  actual: 200);
    assertEquals(answer.getBody(), converted.getId());

    verify(userRepository, times( wantedNumberOfInvocations: 1)).save(any(User.class));
    verifyNoMoreInteractions(userRepository);
    verifyNoMoreInteractions(appSystemRepository);
    verifyNoMoreInteractions(appFunctionRepository);
}
```

```java
@ExtendWith(MockitoExtension.class)
class RoleControllerTest {
    @Mock
    RoleService roleService;

    @InjectMocks
    RoleController roleController;

    /*
     * READ тесты ------------------------------------------------------------------
     */

    @Test
    @DisplayName("Поиск - корректные вызовы")
    void findAllCallsRoleServiceProperly() {
        RoleFilter roleFilter = new RoleFilter();
        Pageable pageable = Pageable.unpaged();

        roleController.findAll(roleFilter, pageable);

        verify(this.roleService, times( wantedNumberOfInvocations: 1))
                .findAll(any(), any());
    }
```

Integration-тесты (Spring Boot Test):

```java
@Test
public void findObjectsforMagistralTest() throws Exception {
    mvc.perform(get( urlTemplate: "/accounting_objects/find_objects_for_magistral").contentType(APPLICATION_JSON)
            .param( name: "page",  ...values: "0")
            .param( name: "size",  ...values: "25")
            .param( name: "accountingObjectId",  ...values: "f3243718-497d-4ac1-8322-d9bb502e141e")
            .param( name: "departmentId",  ...values: "5e5cf9fc-fdae-11e6-bc64-92361f002671"))
            .andExpect(status().isOk())
            .andExpect(jsonPath( expression: "content.[*].type.alias", everyItem(is(HEAT_SUPPLY_NETWORK))))
            .andExpect(jsonPath( expression: "content.[*].departmentId", everyItem(is( value: "5e5cf9fc-fdae-11e6-bc64-92361f002671"))))
            .andExpect(jsonPath( expression: "content.[*].network.viewOfNetwork.code", everyItem(is(MAIN))))
            .andExpect(jsonPath( expression: "content.[*].published", everyItem(is( value: true))))
            .andDo(result -> System.out.println("result -> " + result.getResponse().getContentAsString()));
}
```