Документация, содержащая информацию, необходимую для эксплуатации экземпляра программного обеспечения, предоставленного для проведения экспертной проверки

Программного обеспечения «Amplicode»

Термины и определения

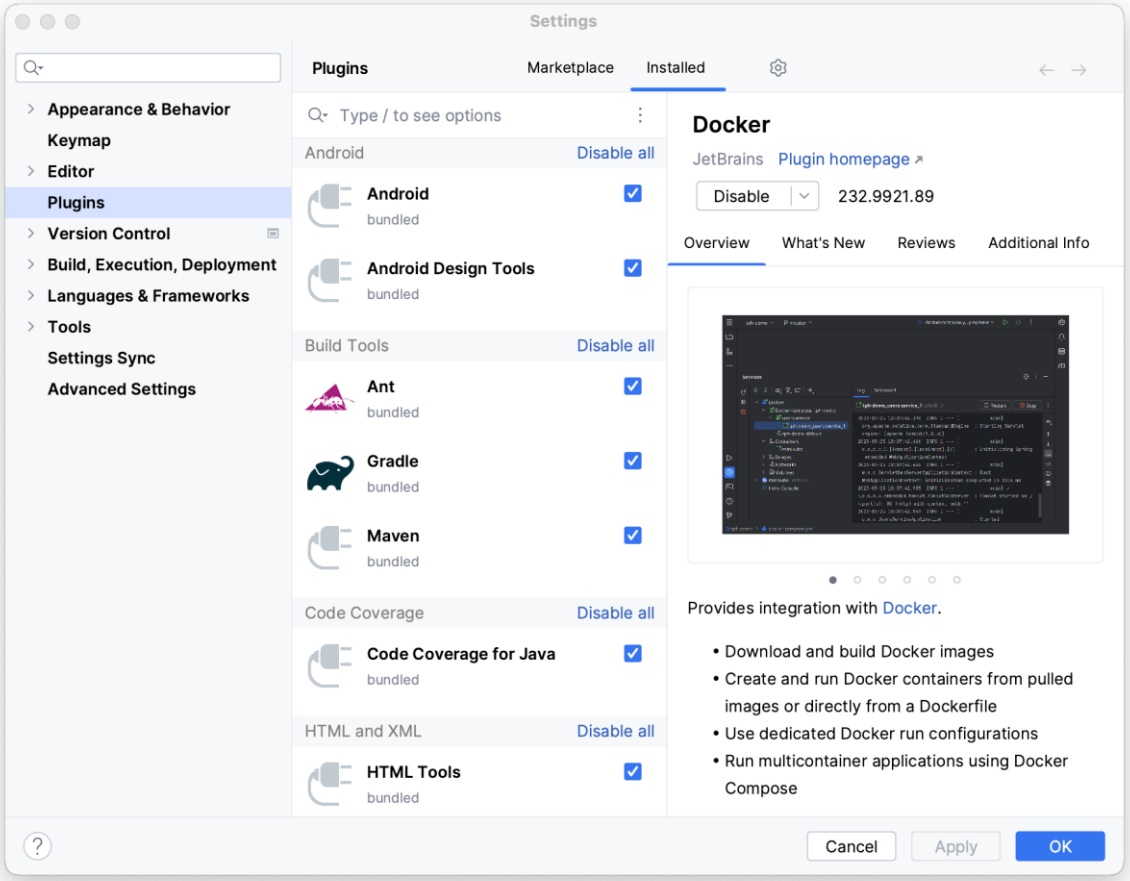
Термин	Определение
ПКМ	Правая кнопка мыши
Git branch	Команда для управления ветками в репозитории Git
Gutter icon	Расположены в редакторе слева. Они вызывают некоторые базовые действия и другие функции, специфичные для фреймворка и технологии.
Docker	Открытая платформа для разработки приложений, созданная для поддержки DevOps и разработчиков
PostgreSQL	Свободная объектно-реляционная система управления базами данных (СУБД)
Spring Security	Java/Java EE фреймворк, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для промышленных приложений, созданных с помощью Spring Framework
DTO	Удобный способ структурировать данные для передачи по сети или между различными компонентами приложения. Основная цель DTO - облегчить обмен информацией между различными частями приложения, упрощая процесс коммуникации и повышая его эффективность.
REST контроллер	Используется для создания RESTfull веб-сервисов. Он взаимодействует с какими-либо данными и возвращает объект, который представляется в HTTP ответе в виде JSON или XML.
Эндпоинт	Конечная точка веб-сервиса, к которой клиентское приложение обращается для выполнения определённых операций или получения данных

1. Первоначальная настройка

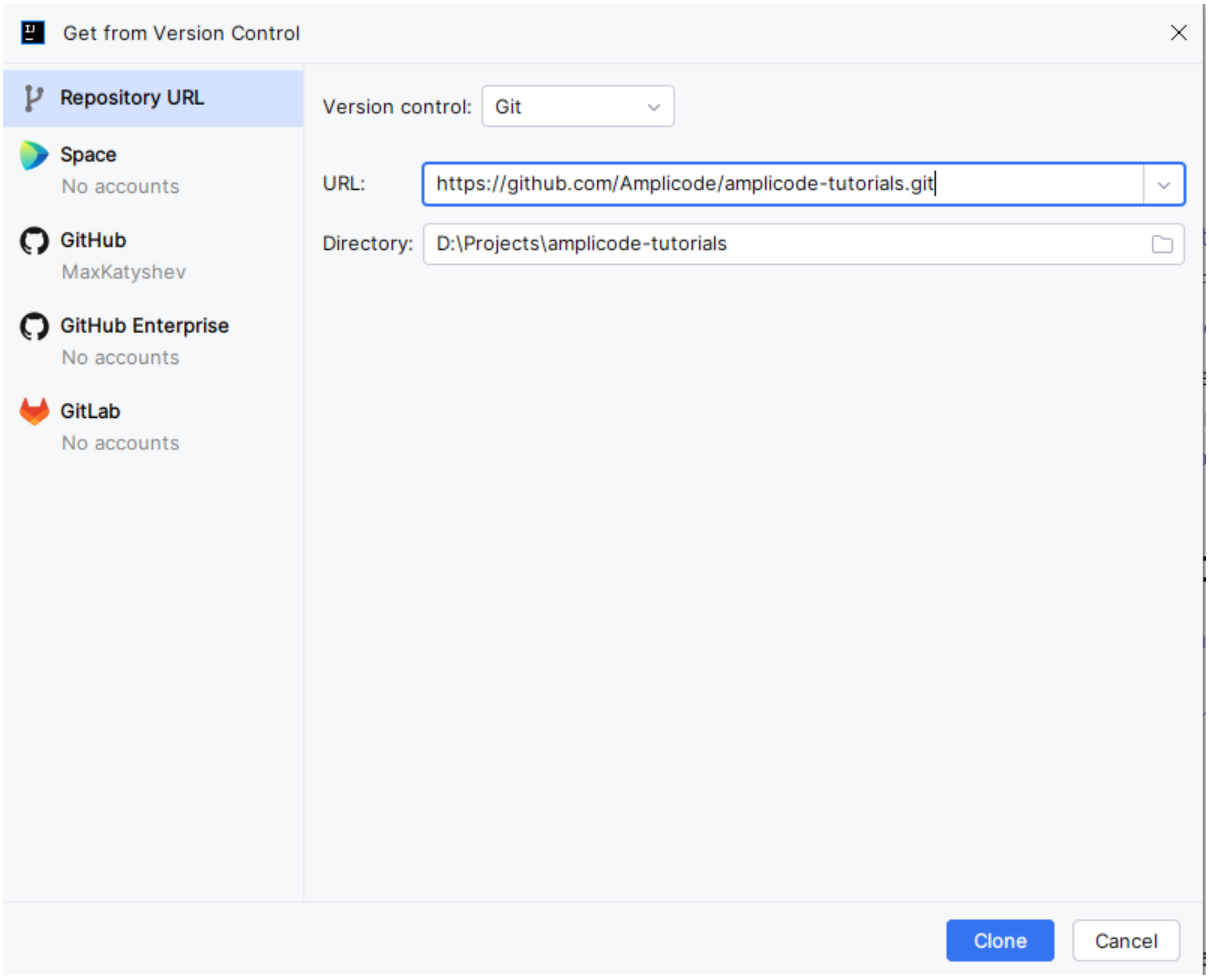
Установите экземпляр ПО как описано в документе “Инструкция по установке экземпляра программного обеспечения, предоставленного для проведения экспертной проверки ПО «Amplicode»”.

2. Скачивание проекта Spring Petclinic

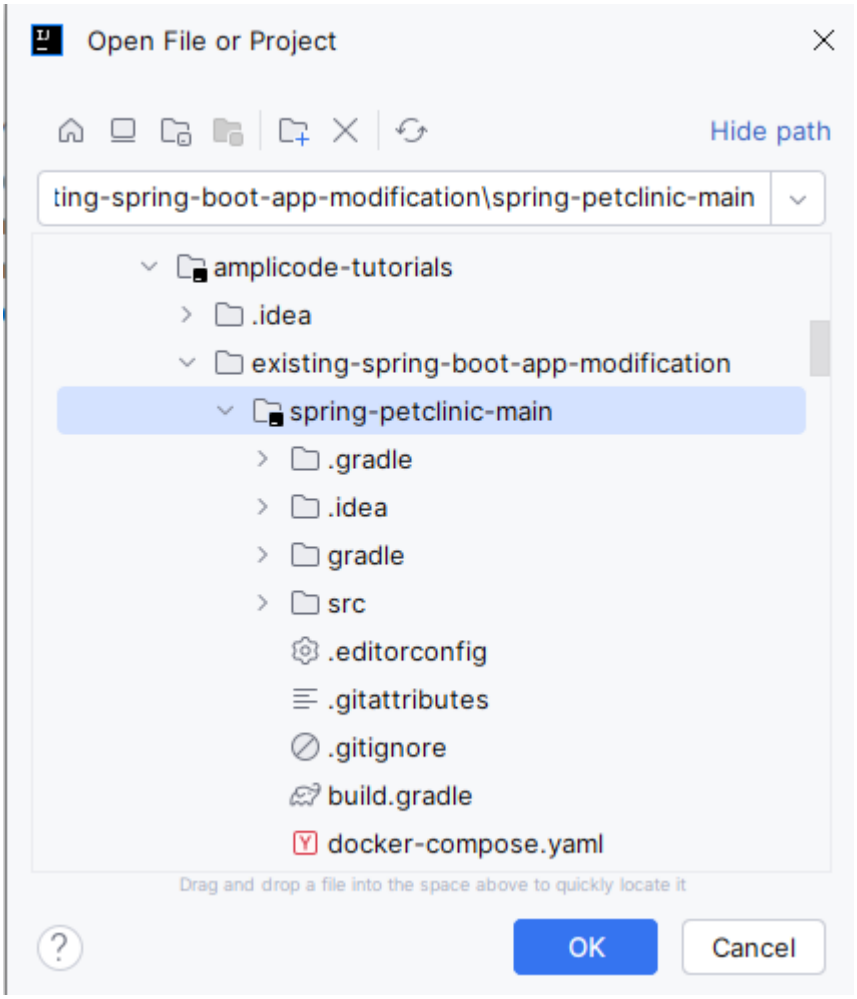
1. Откройте настройки (File → Settings)
2. Перейдите в секцию Plugins



3. Находясь на складке Marketplace, впишите в поле ввода Git и установите плагины Git и GitHub
4. Перезапустите IntelliJ IDEA
5. Откройте File → New → Project from Version Control... и в поле URL вставьте следующую ссылку на репозиторий https://github.com/Amplicode/amplicode-tutorials.git



6. Нажмите Clone
7. Далее откройте проект Spring petclinic, находящийся в этом репозитории и нажмите OK



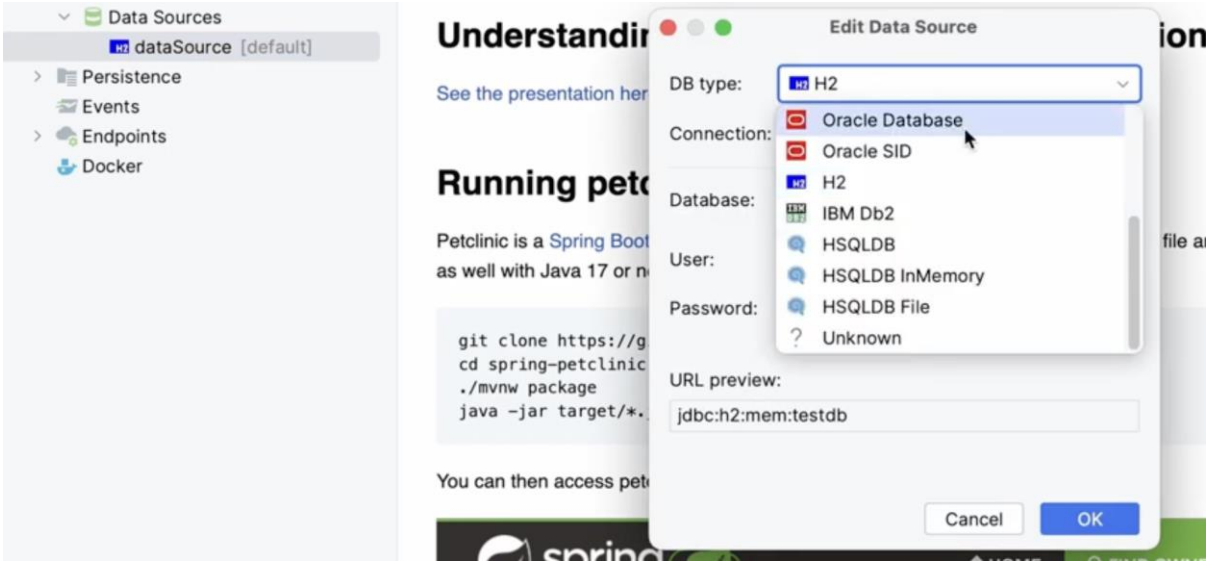
8. Выберете ветку (branch) main в верхнем меню → нажмите Checkout Tag or Revision → в поле введите 6b745c79082e96940b1f1ea3eeb004a21b0278c6 и нажмите OK
9. Далее, чтобы начать писать проект с новой ветки выберете в меню Git → New branch → введите название ветки amplicode-demo → Create
   Теперь можно начинать писать проект.

3. Работа плагина в существующем проекте

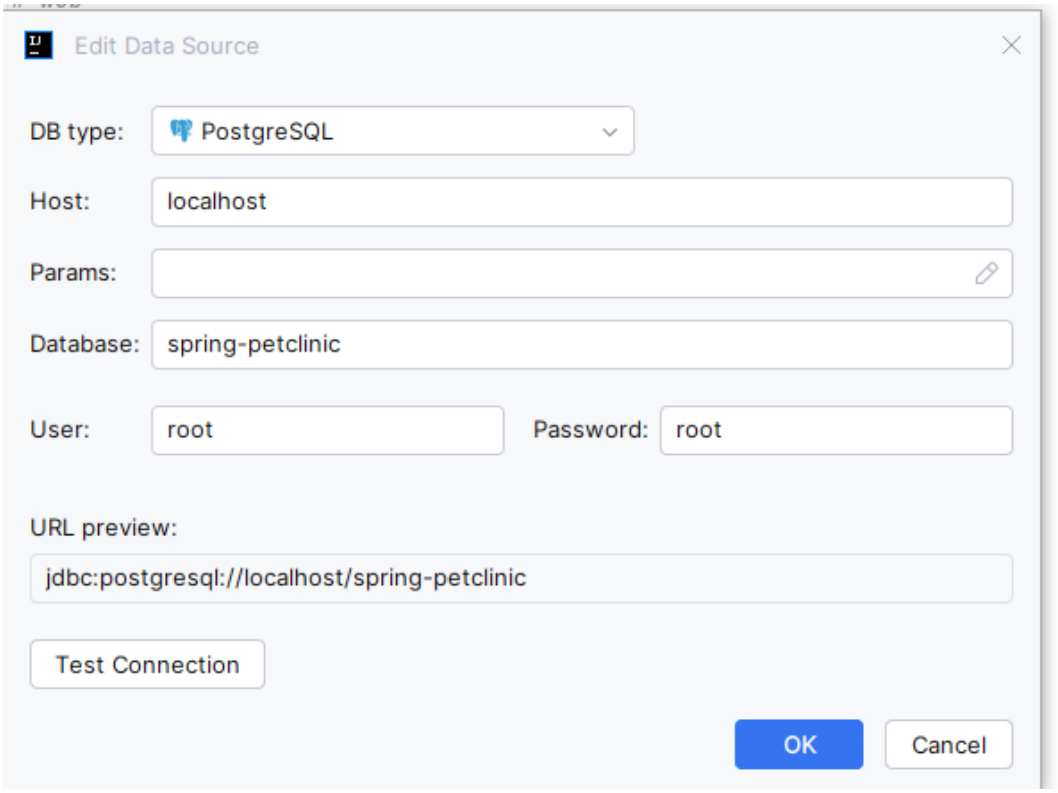
После успешного клонирования проекта появляется Welcome Screen и все возможности Amplicode становятся доступны. Amplicode предоставляет возможность проанализировать приложение в контексте используемых фреймворков и библиотек с помощью панели Amplicode Explorer.

3.1 Редактирование встроенной базы данных

1. Amplicode Explorer → PetClinicApplication → Configurations → Data Sources → выберите двойным кликом dataSource [default]



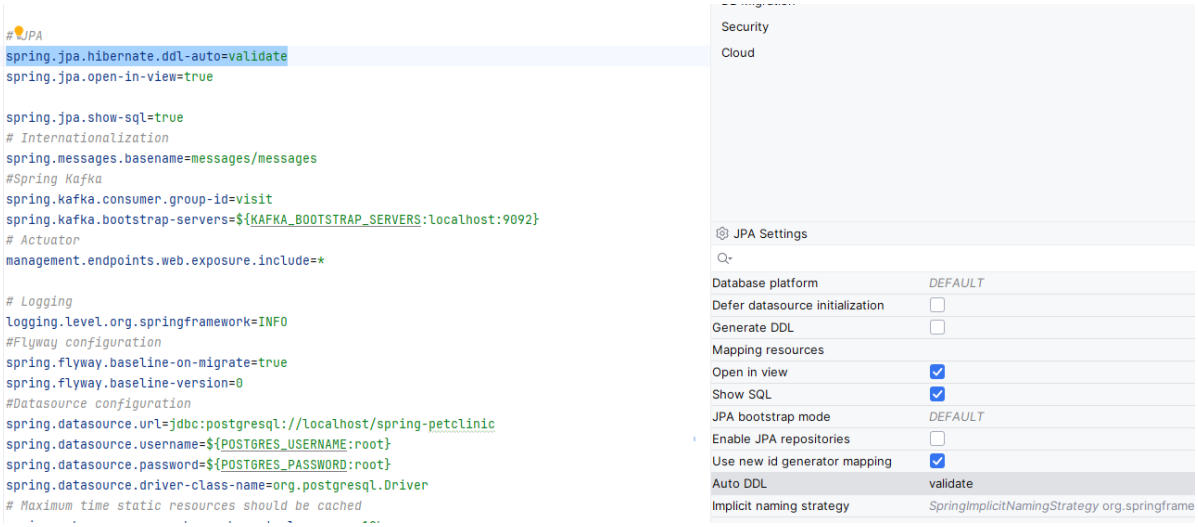
2. В открывшемся окне необходимо выполнить следующие действия: изменить тип базы данных на PostgreSQL; указать необходимые параметры, такие как хост, порт, название базы данных, имя пользователя и пароль, как показано ниже → нажмите ОК



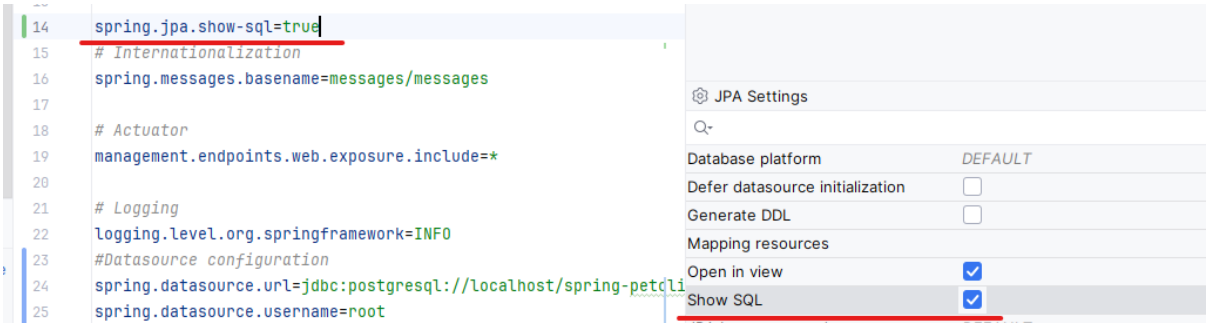
После этого Amplicode не только обновил значения свойств в файле application.properties, но также добавил необходимую зависимость для взаимодействия с базой данных PostgreSQL из нашего приложения.

```
21    logging.level.org.springframework=INFO
22    #Datasource configuration
23    spring.datasource.url=jdbc:postgresql://localhost/spring-petclinic
24    spring.datasource.username=root
25    spring.datasource.password=root
26    spring.datasource.driver-class-name=org.postgresql.Driver
34      developmentOnly 'org.springframework.boot:spring-boot-de
35      testImplementation 'org.springframework.boot:spring-boot
36        runtimeOnly 'org.postgresql:postgresql'
37    }
```

3. Перейти в файл application.properties и поставить курсор на свойство spring.jpa.hibernate.ddl-auto=none
   Откройте Amplicode Designer выберите значение validate для Auto DDL. Благодаря использованию этого свойства со значением validate, можно убедиться в том, что JPA модель полностью соответствует схеме, заданной в базе данных.



4. Выберите свойство Show SQL в значении true для удобства отслеживания того, какие именно запросы отправляются в базу данных.



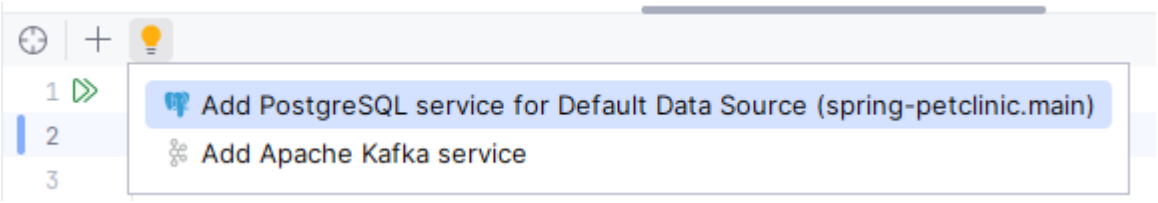
3.2 Настройка сервисов в файле Docker Compose

С помощью узла Docker в панели Amplicode Explorer вы можете создать файл Docker Compose.

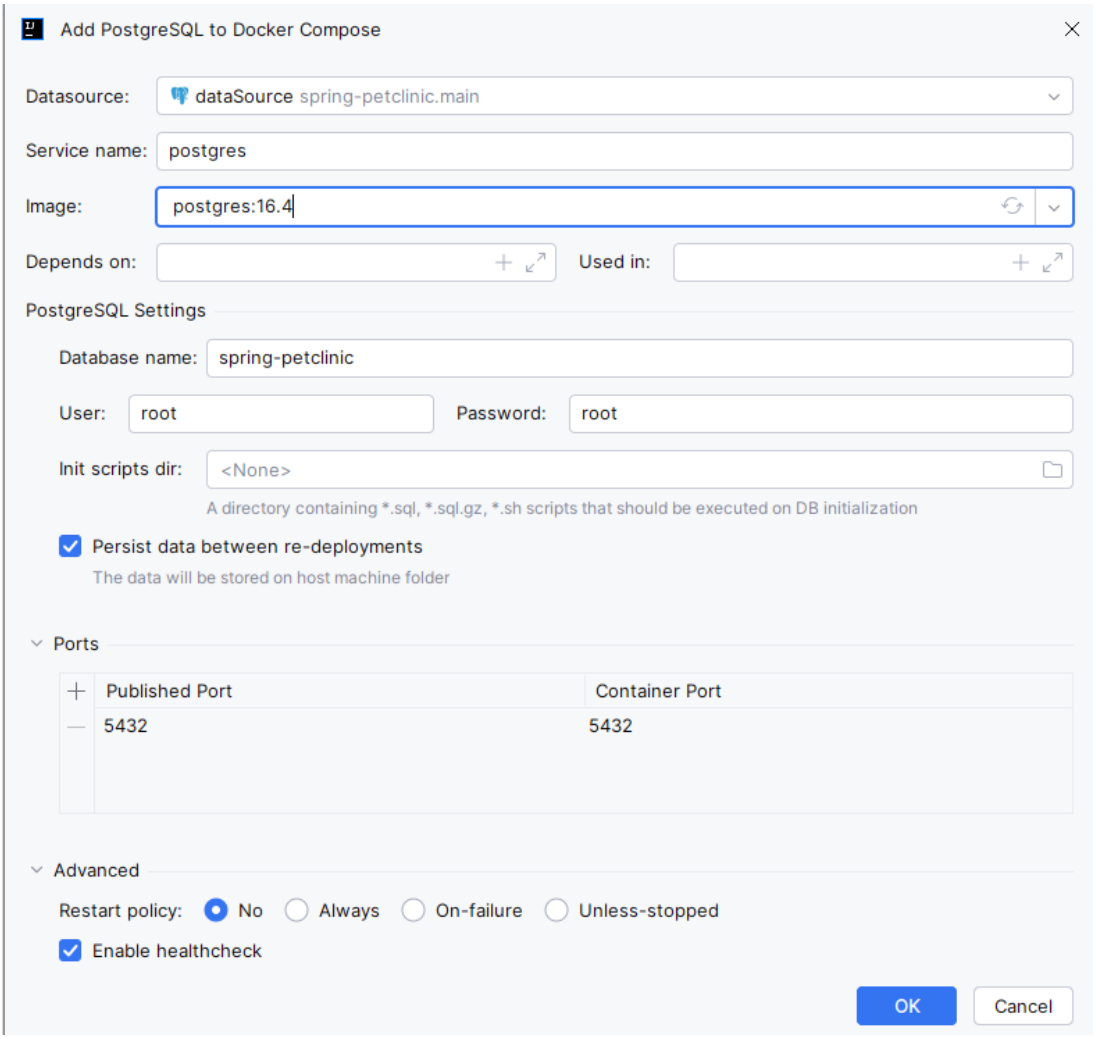
1. Нажмите правую кнопку мыши на узле Docker → New → Docker Compose file →

оставьте наименование файла по умолчанию → OK

2. В открывшемся файле нажмите на иконку желтой лампочки под названием Suggestions в панели Editor Toolbar.



3. Здесь уже присутствует подсказка по добавлению необходимого сервиса. Нажмите Add PostgreSQL service for Default Data Source. Amplicode воспользовался уже существующим подключением к базе данных и подставил все необходимые значения в обязательные поля.

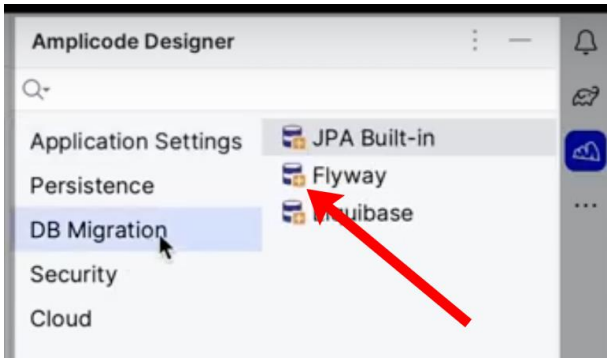


4. Любой из параметров можно менять в случае необходимости, но на текущем этапе этого делать не нужно. Достаточно просто нажать OK. Сервис готов.
5. Запустите базу данных, а затем и приложение, чтобы убедиться, что все работает, как и прежде. Для запуска базы данных нажмите на значок двойной стрелочки в файле docker-compose.yaml.

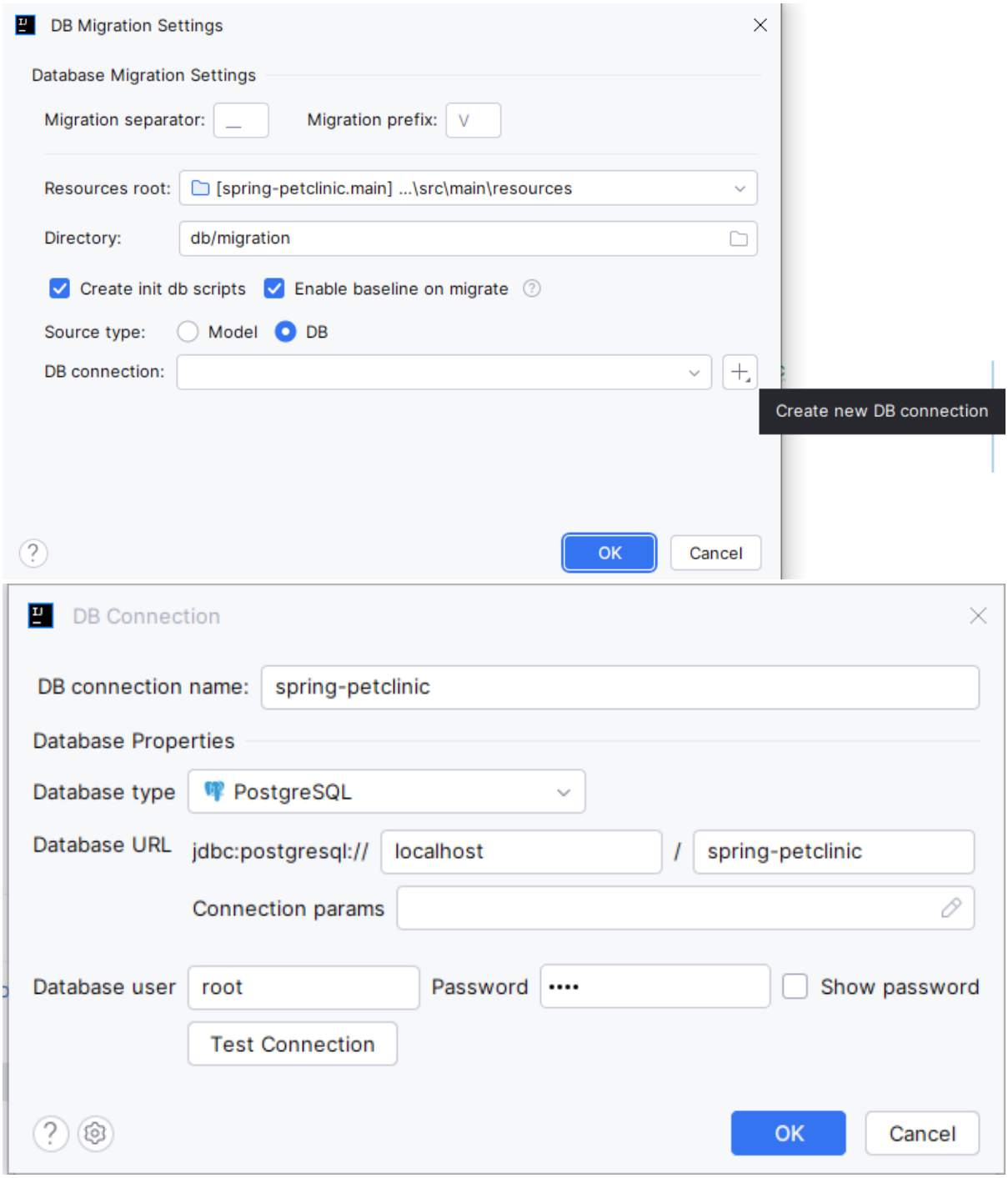
```yaml
services:
  postgres:
    image: postgres:16.4
    restart: "no"
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: root
      POSTGRES_PASSWORD: root
      POSTGRES_DB: spring-petclinic
    healthcheck:
      test: pg_isready -U $$POSTGRES_USER -d $$POSTGRES_DB
      interval: 10s
      timeout: 5s
      start_period: 10s
      retries: 5
volumes:
  postgres_data:
```

3.3 Подключение Flyway. Генерация скрипта инициализации БД

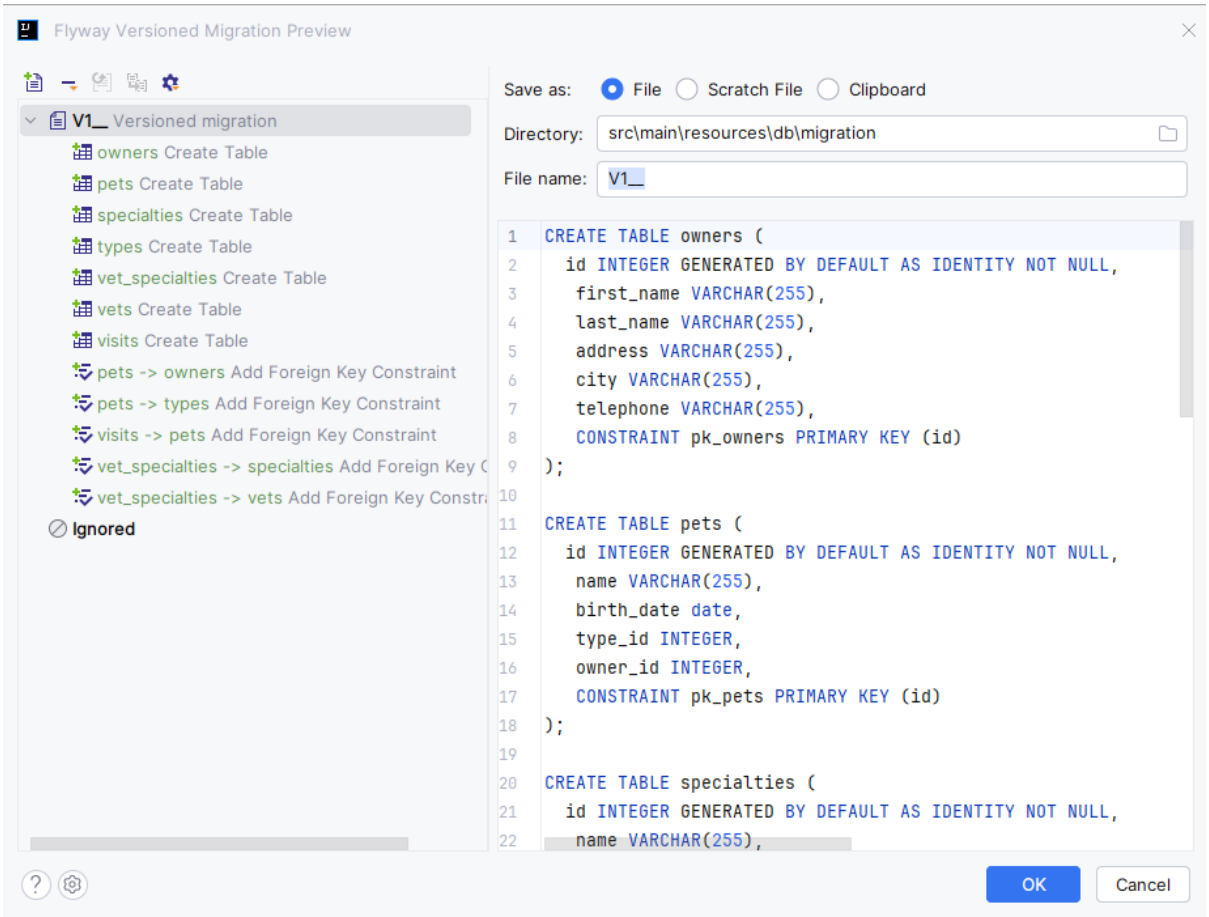
1. Оставаясь в файле application.properties, выберите Flyway в секции DB Migration в Amplicode Designer



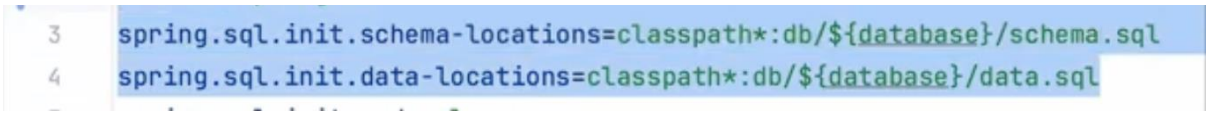
2. Оставьте значение префикса-сепаратора и расположение директории, которая будет содержать сгенерированные скрипты, по умолчанию → для того чтобы Amplicode смог прочитать структуру базы данных и сгенерировать скрипт инициализации, нажмите на + в поле DB Connection → Create New → создайте новое подключение, нажав OK в диалоговом окне “DB Connection”.



3. Нажмите OK в диалоговом окне “DB Migration Settings”
4. Amplicode Explorer → Create (+) → DB Versioning → Flyway Versioned Migration → в поле Select DB Connection выберите созданное подключение к Postgres и нажмите OK. Amplicode сгенерирует скрипты инициализации базы данных. Прежде чем сохранить их, следует убедиться, что они соответствуют нашим ожиданиям.



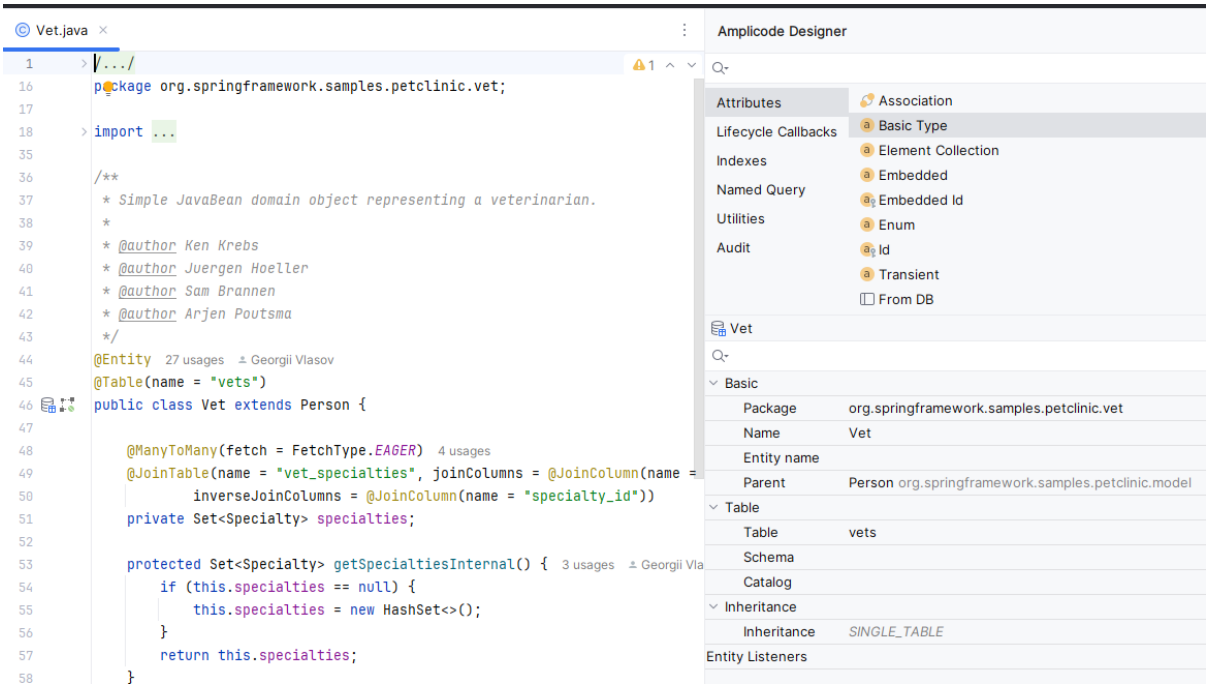
5. Нажмите OK. Скрипт инициализации базы данных готов. Ранее существовавшие скрипты инициализации базы данных при наполнении ее данными больше не понадобятся, поэтому необходимо удалить те свойства, которые на них ссылаются.



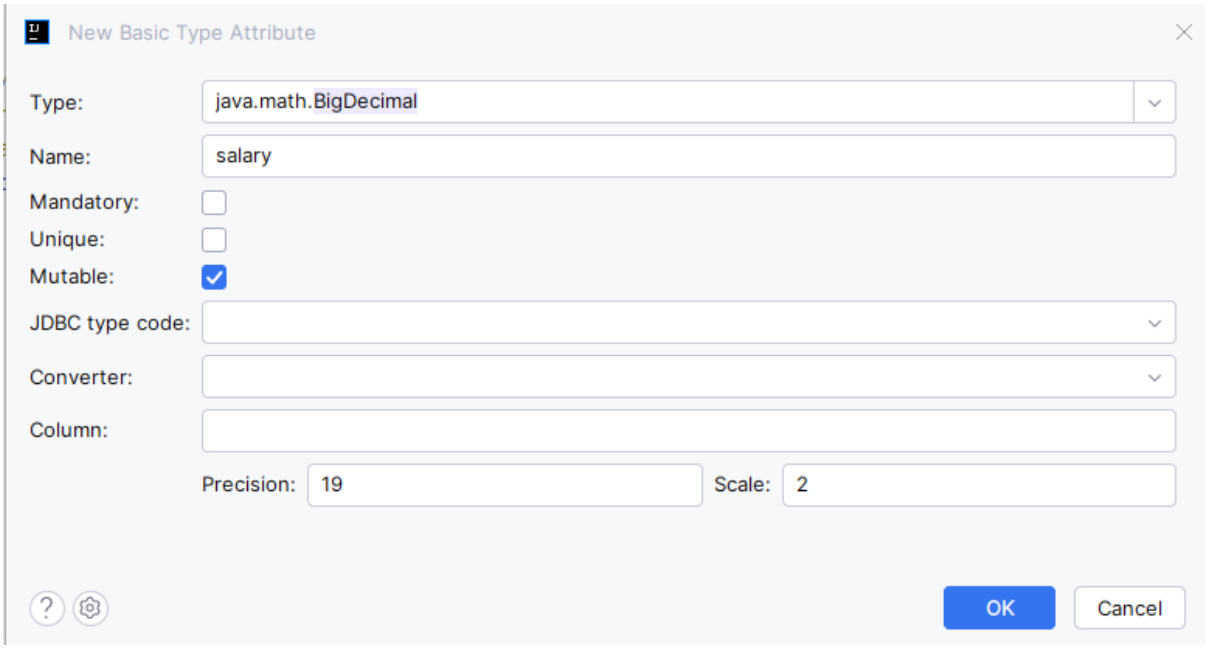
3.4 Изменение модели: добавление базового атрибута

Следующей задачей является реализация учета заработных плат сотрудников. Для этого нужно добавить информацию о зарплате в сущность Vet.

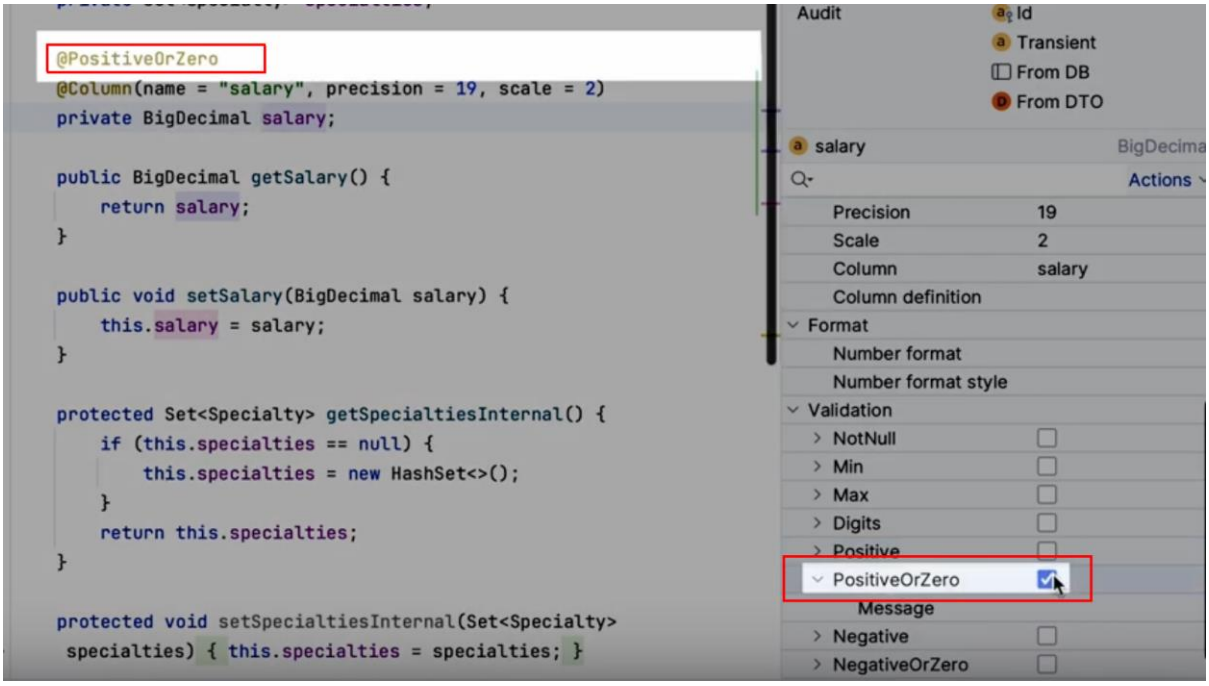
1. Перейдите в сущность Vet
2. Amplicode Designer → Attributes → двойным кликом нажмите на Basic type



3. Выберите тип данных BigDecimal и введите название salary



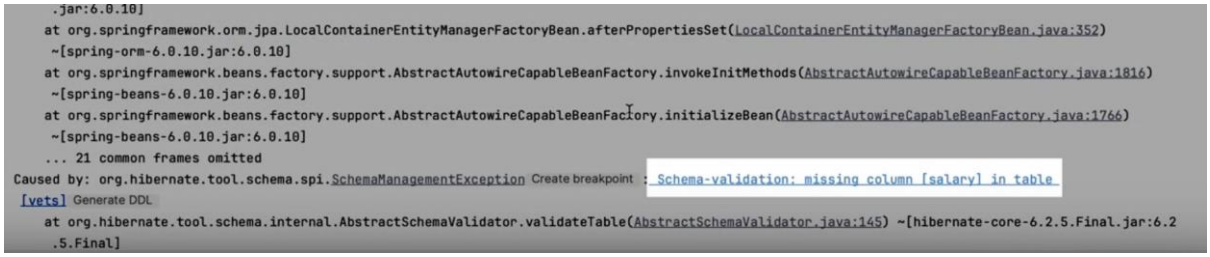
4. Нажмите OK. Атрибут готов.
5. Добавьте валидацию для атрибута @PositiveOrZero с помощью Amplicode Designer



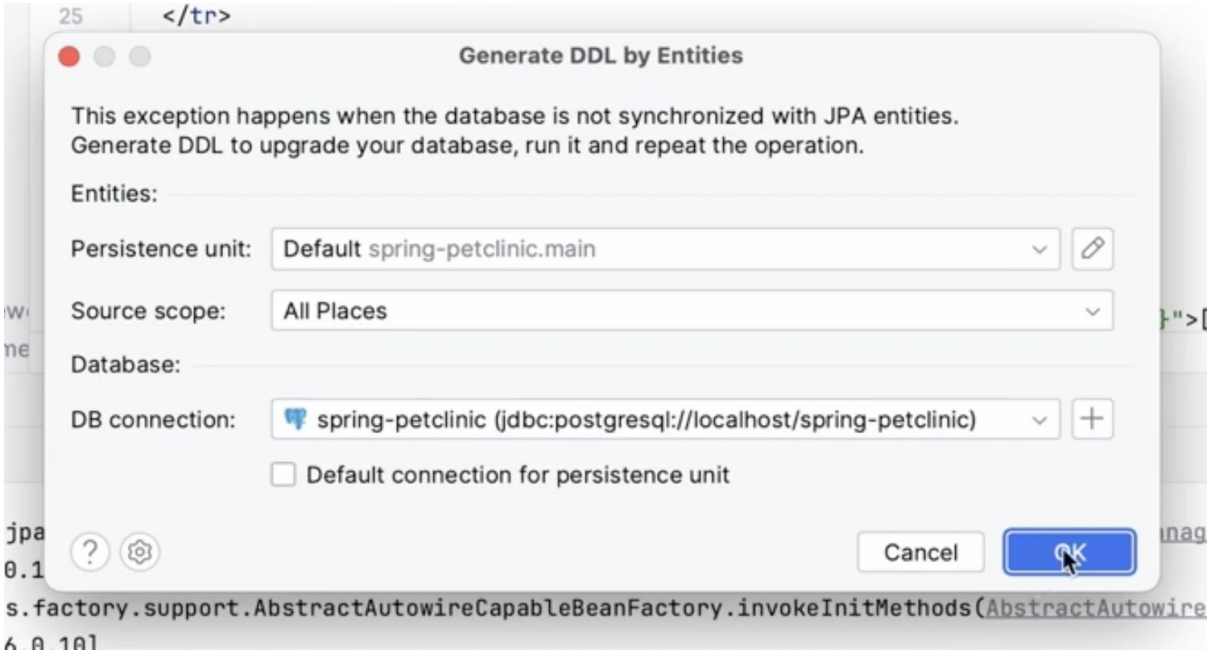
6. Внесите соответствующие изменения в пользовательский интерфейс, добавив новое поле в файл vetList.html

```
<table id="vets" class="table table-striped">
  <thead>
  <tr>
    <th>Name</th>
    <th>Specialties</th>
    <th>Salary</th>
  </tr>
  </thead>
  <tbody>
  <tr th:each="vet : ${listVets}">
    <td th:text="${vet.firstName + ' ' + vet.lastName}"></td>
    <td><span th:each="specialty : ${vet.specialties}"
              th:text="${specialty.name + ' '}"/> <span
      th:if="${vet.nrOfSpecialties == 0}">none</span></td>
    <td th:text="${vet.salary}"></td>
  </tr>
  </tbody>
</table>
```

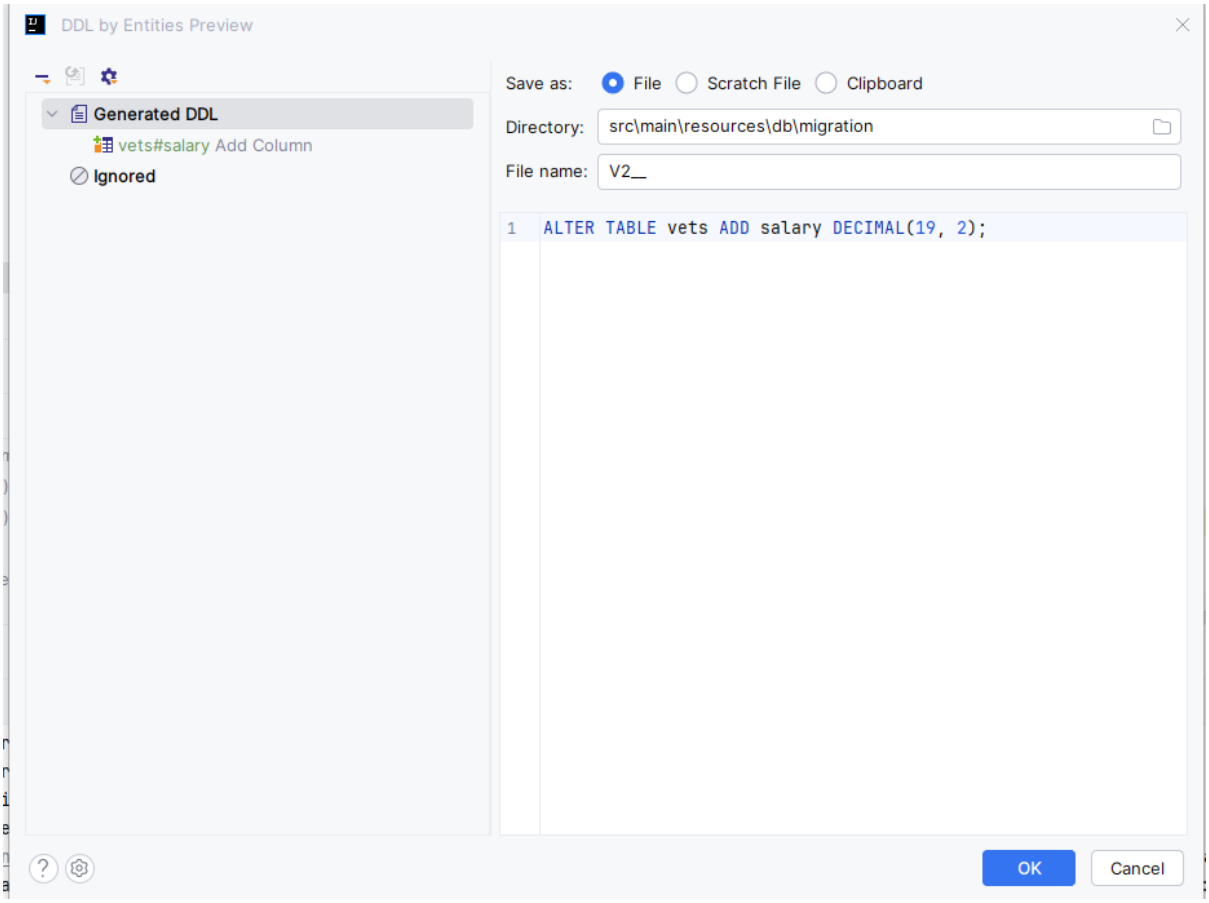
7. Запустите приложение. Появляется ошибка в логах приложения. Это корректное поведение из-за того, что соответствующий скрипт для добавления новой колонки в базу данных отсутствует. Amplicode позволяет вызвать действие по синхронизации модели с базой данных прямо из stacktrace. Для этого нажмите на помеченную на рисунке ниже ссылку:



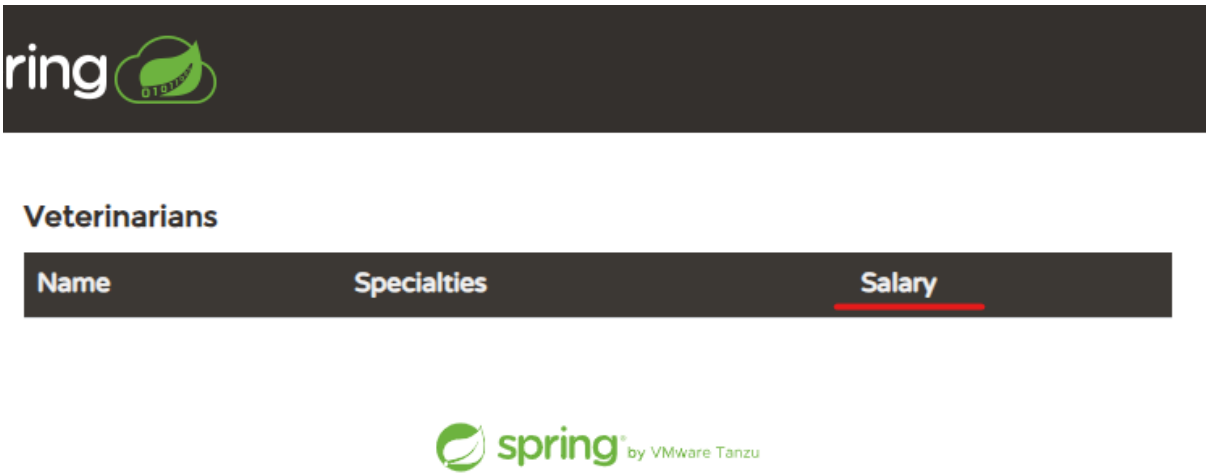
8. В появившемся диалоговом окне “Generate DDL by Entities” нажмите ОК



9. Нажмите ОК в диалоговом окен “DDL by Entities Preview”



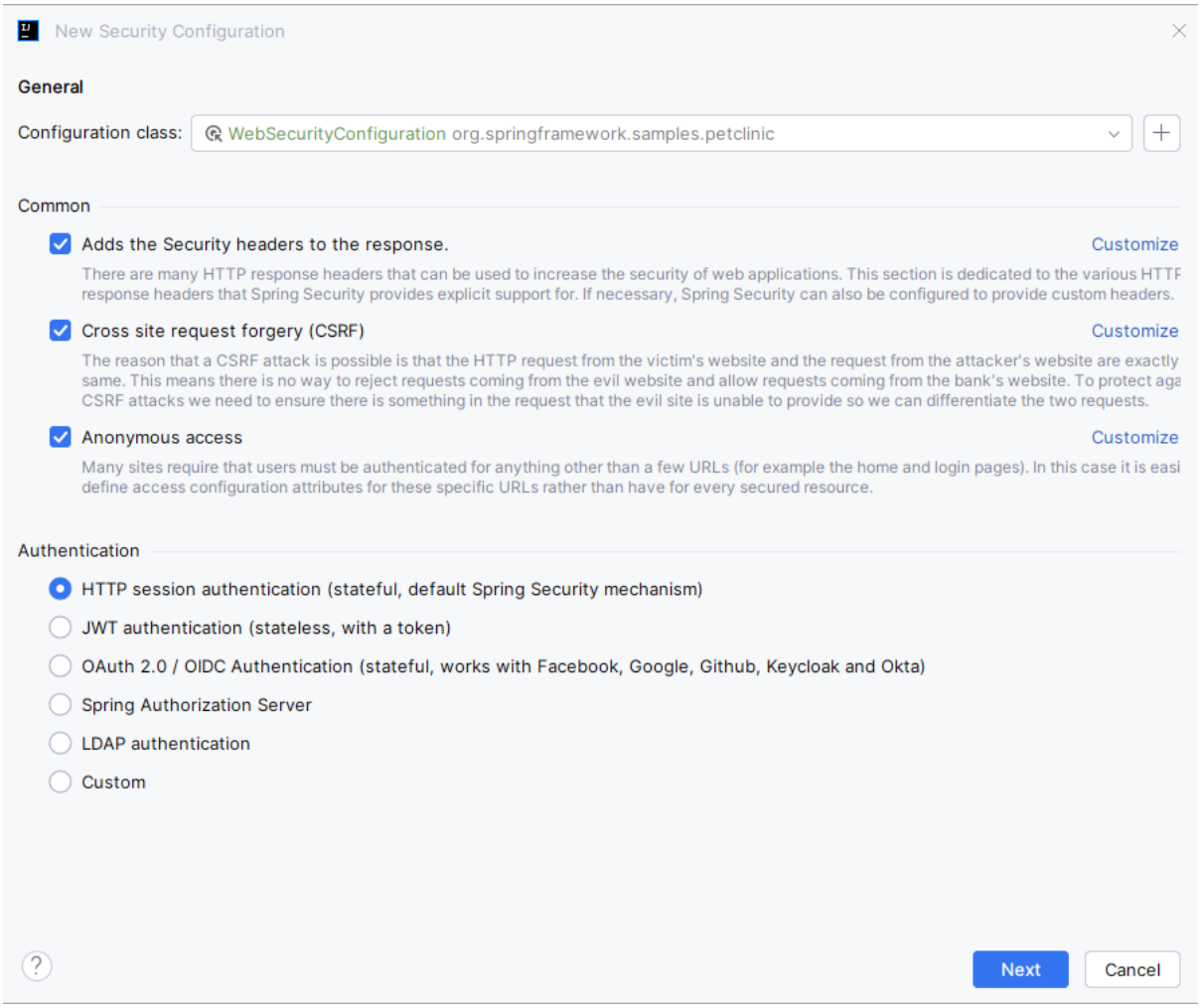
10. Снова запустите приложение → в браузере перейдите по ссылке http://localhost:8080/vets.html
Можно убедиться, что поле Salary теперь доступно в приложении



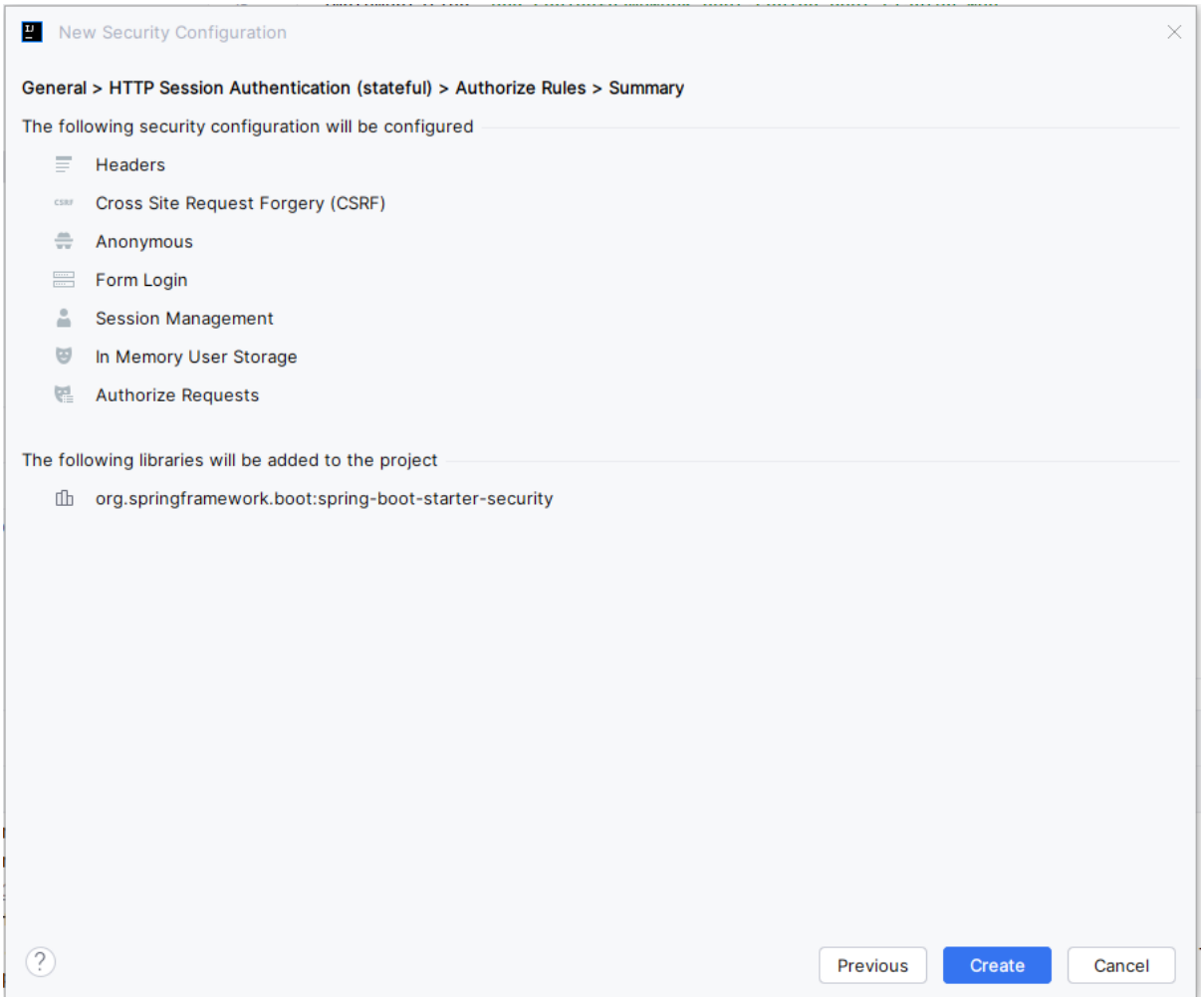
3.5 Подключение и настройка Spring Security

Давайте проанализируем, что мы уже сделали. Решив предыдущую задачу, мы столкнулись с небольшой проблемой безопасности. Теперь абсолютно все, даже неаутентифицированные пользователи, могут просматривать информацию о зарплатах сотрудников. Давайте сделаем так, чтобы эту информацию могли видеть только аутентифицированные пользователи. Для этого добавим Spring Security и настроим доступ к эндпоинту.

1. В панели Amplicode Explorer нажмите Create (+) → Spring → Configurations → Spring Security Configuration. В диалоговом окне есть возможность настроить один из множества наиболее популярных способов аутентификации и дополнительные параметры, специфичные для каждого из типов.



2. Оставьте базовый механизм Spring Security HTTP session authentication и нажмите Next
3. В следующем окне оставьте все параметры по умолчанию → Next → Next
4. На последнем шаге можно увидеть, какие новые зависимости будут добавлены к проекту, затем нажмите Create

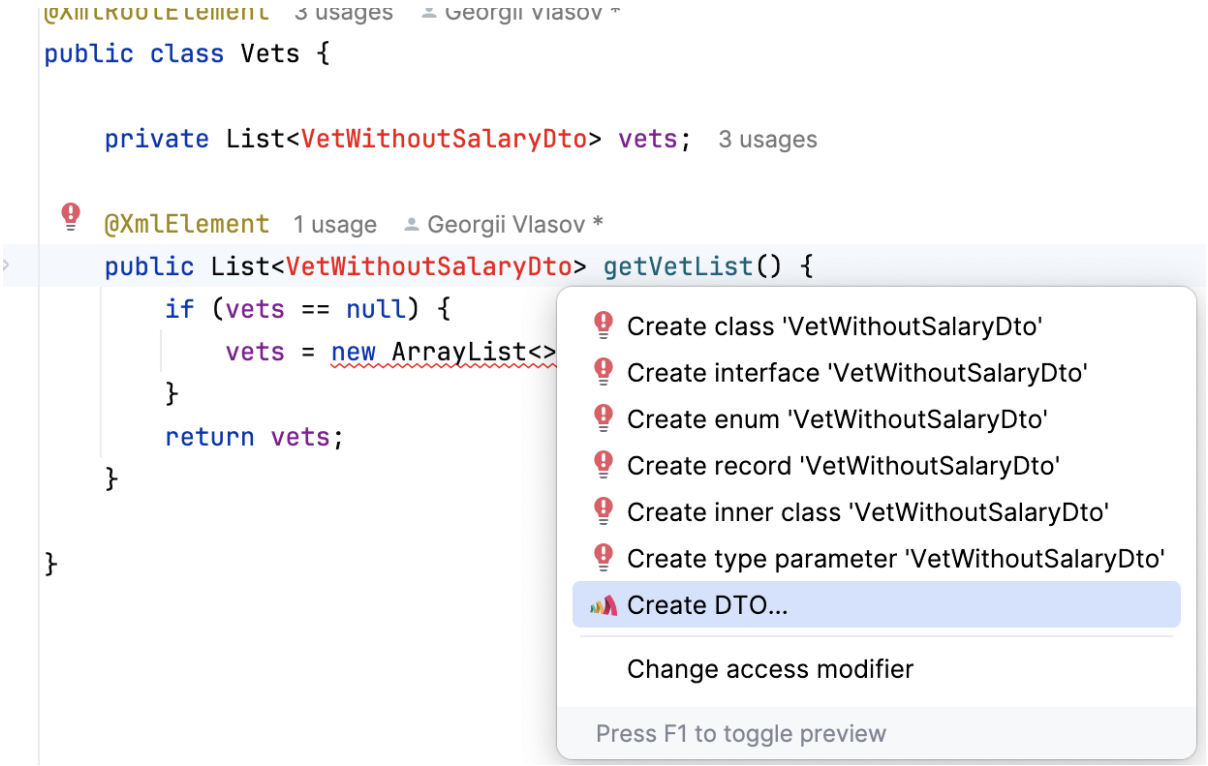


Конфигурационный класс для Spring Security готов. Перезапустите приложение, чтобы убедиться в том, что теперь информацию о ветеринарах смогут видеть только аутентифицированные пользователи.

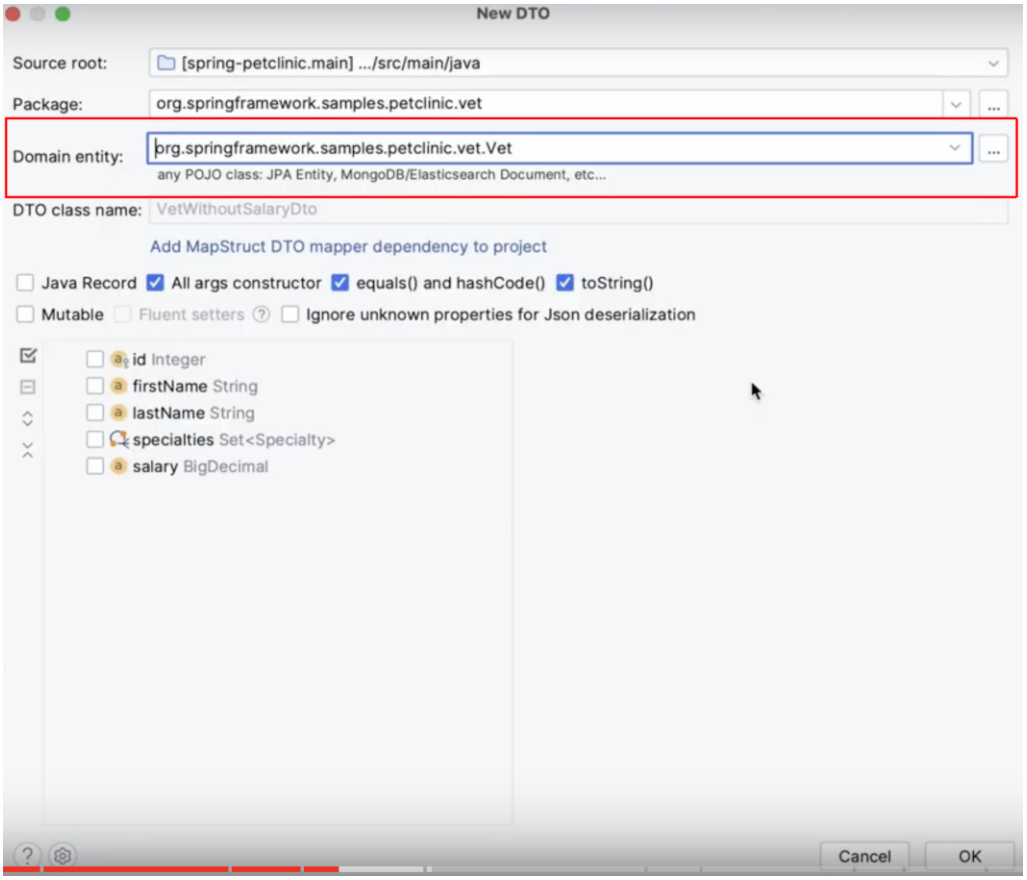
3.6 Модификация существующего REST эндпоинта

Помимо MVC эндпоинтов в приложении также существует один REST эндпоинт (класс Vets). В качестве возвращаемого значения здесь используется список сущностей. Как следствие, каждое изменение в модели, такое как добавление зарплаты или любого другого поля, будет отражаться и на клиентах. Наиболее распространенным решением этой проблемы является использование DTO в качестве возвращаемого типа. Необходимо изменить возвращаемый тип на не существующий пока что VetWithoutSalaryDto.

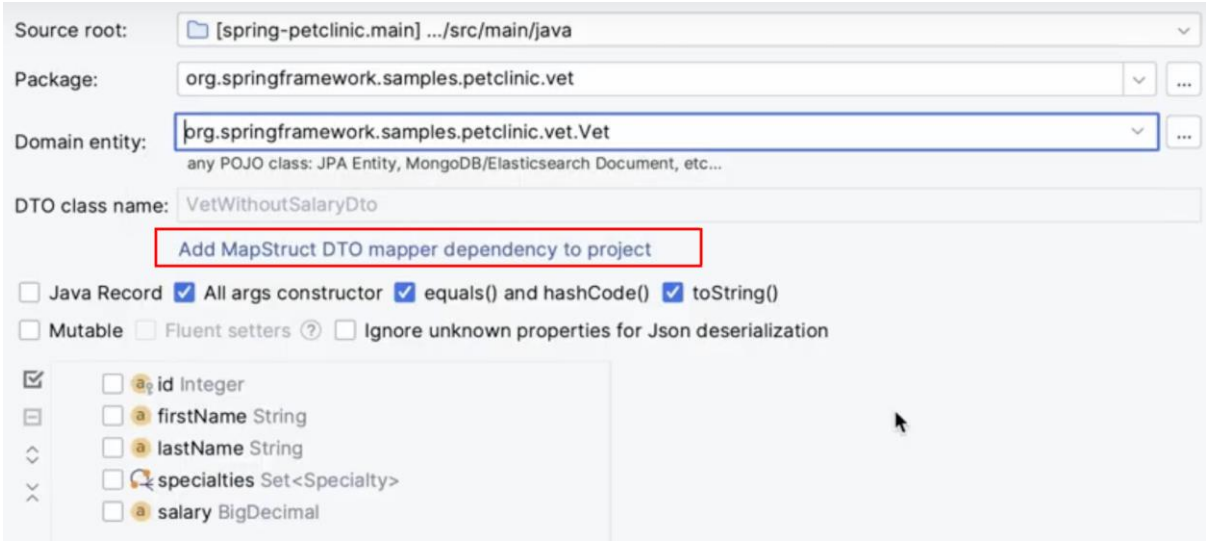
1. Откройте класс Vets
2. Вместо сущности Vet напишите VetWithoutSalaryDto → вызовите диалог создания DTO прямо отсюда (нажмите Alt+Enter для Windows/Linux или ⌥+Enter для macOS)



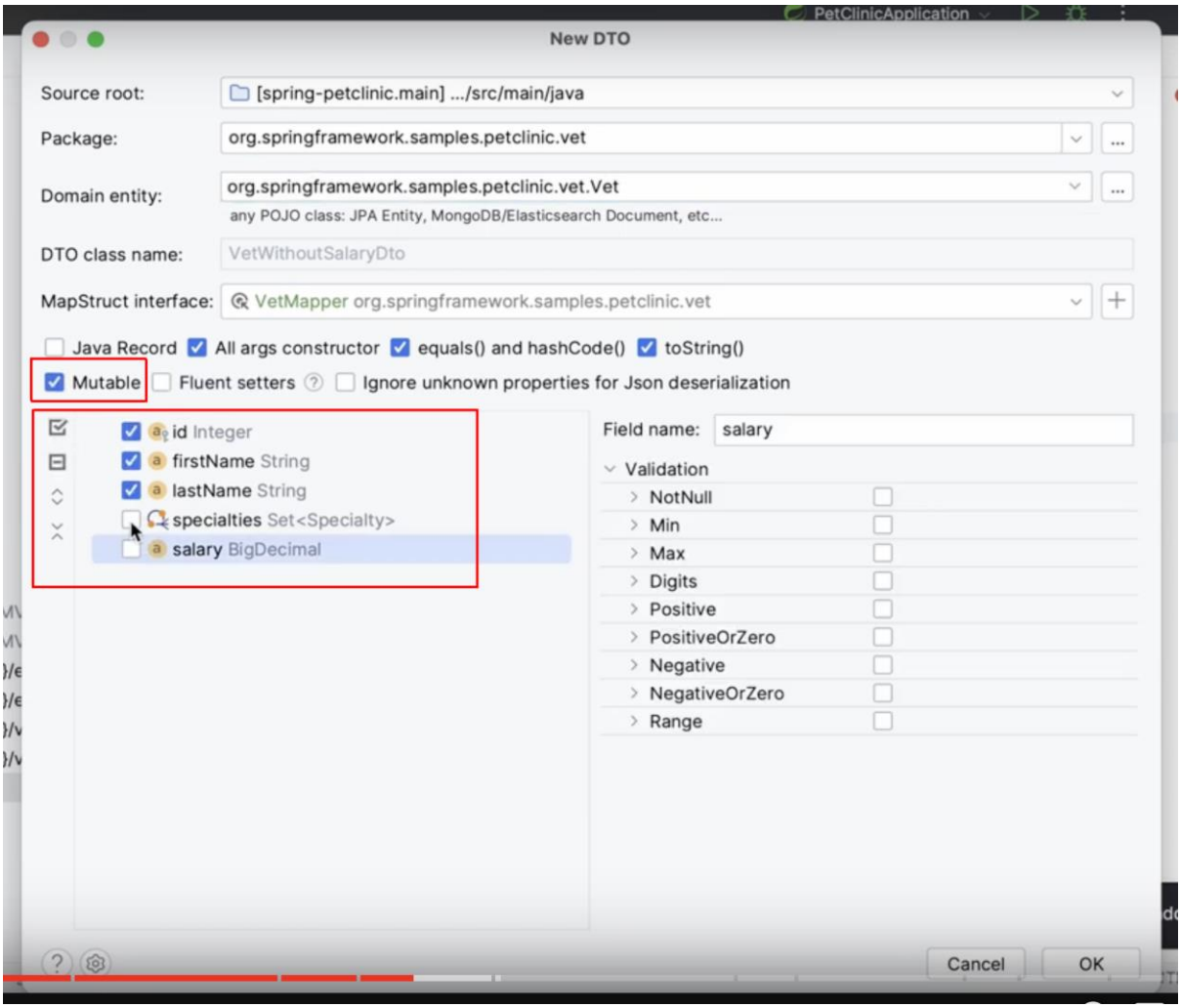
3. Нажмите Create DTO
4. Выберите сущность Vet в поле Domain entity



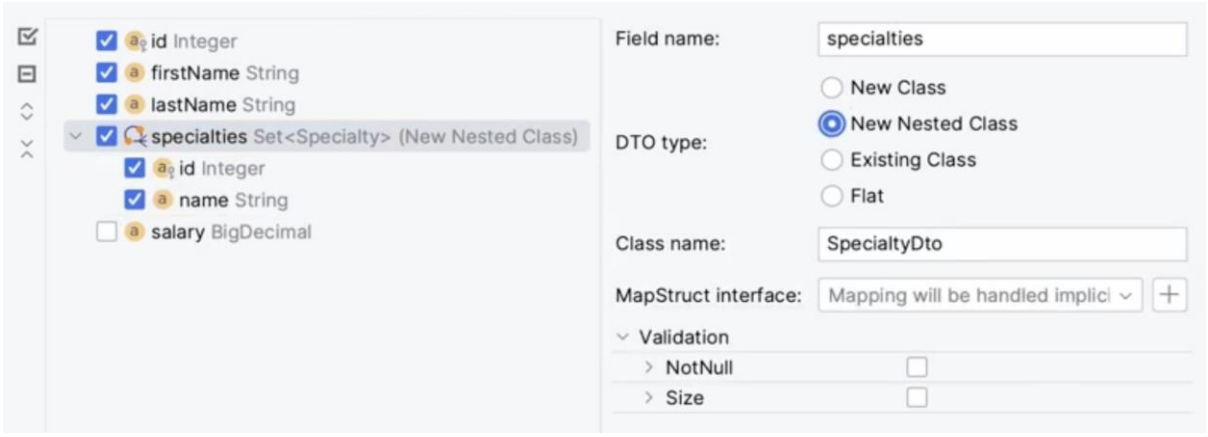
5. Нажмите Add MapStruct DTO mapper dependency to project → создайте абстрактный интерфейс в поле MapStruct interface



6. Сделайте DTO мутабельным и выберите все поля, кроме зарплаты



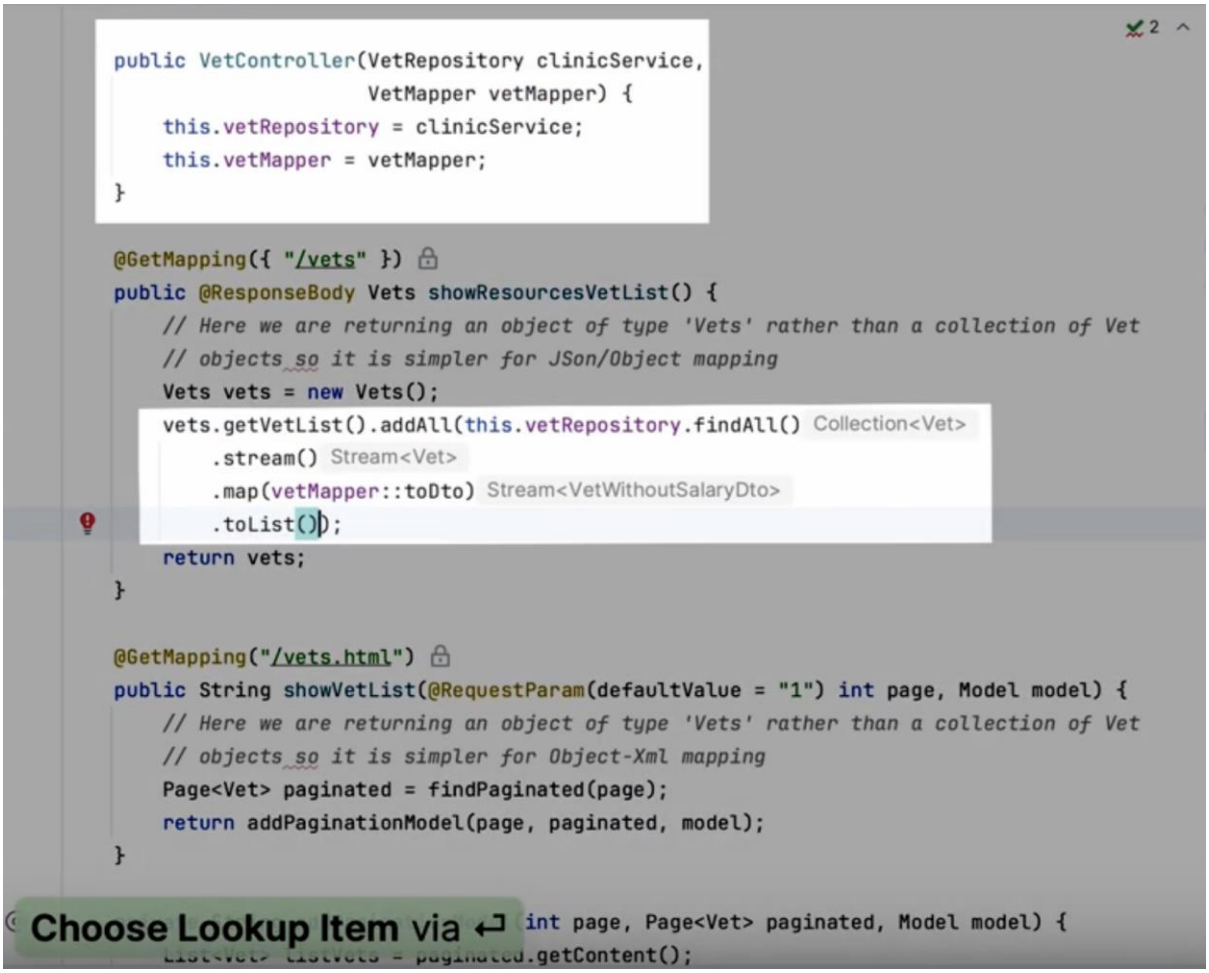
7. Для ассоциативного поля специальностей Amplicode также предлагает создать DTO. Следовательно, создайте DTO для специальностей прямо в текущем объекте, выбрав тип New Nested Class.



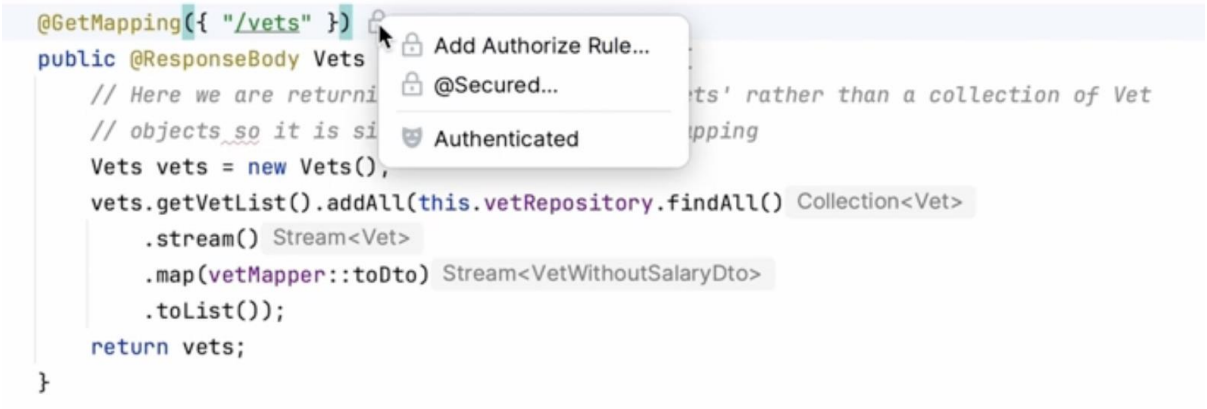
8. Нажмите OK
9. Преобразуйте список возвращаемых сущностей в список DTO в методе showResourcesVetList класса VetController, добавив .map после findAll()

10. Выберите mapToVetWithoutSalaryDto

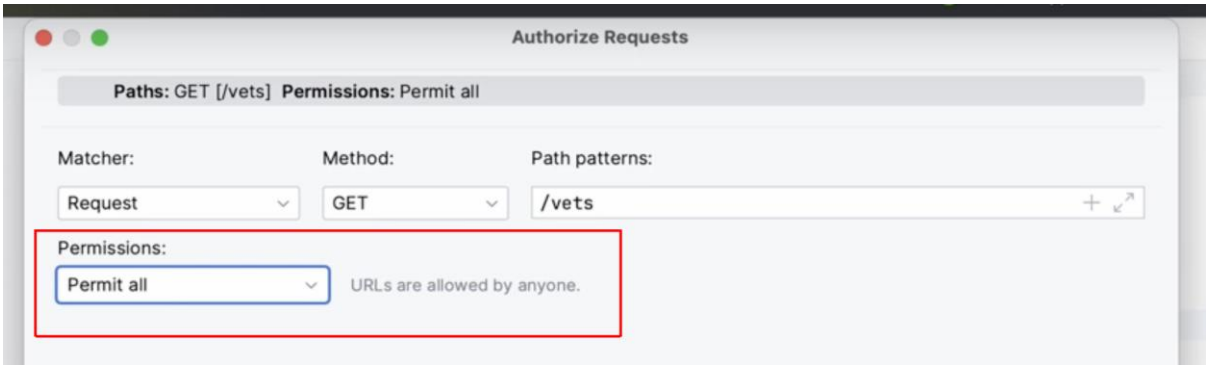
    Помимо этого, также в контроллере мы получим корректно заинжектированный Bean маппера в наш контроллер.



11. Еще одним пунктом будет предоставление доступа к эндпоинту всем без исключения. Amplicode позволяет сделать это прямо из исходного кода контроллера. Для этого кликните на иконку замочка рядом с аннотацией @GetMapping метода showResourcesVetList



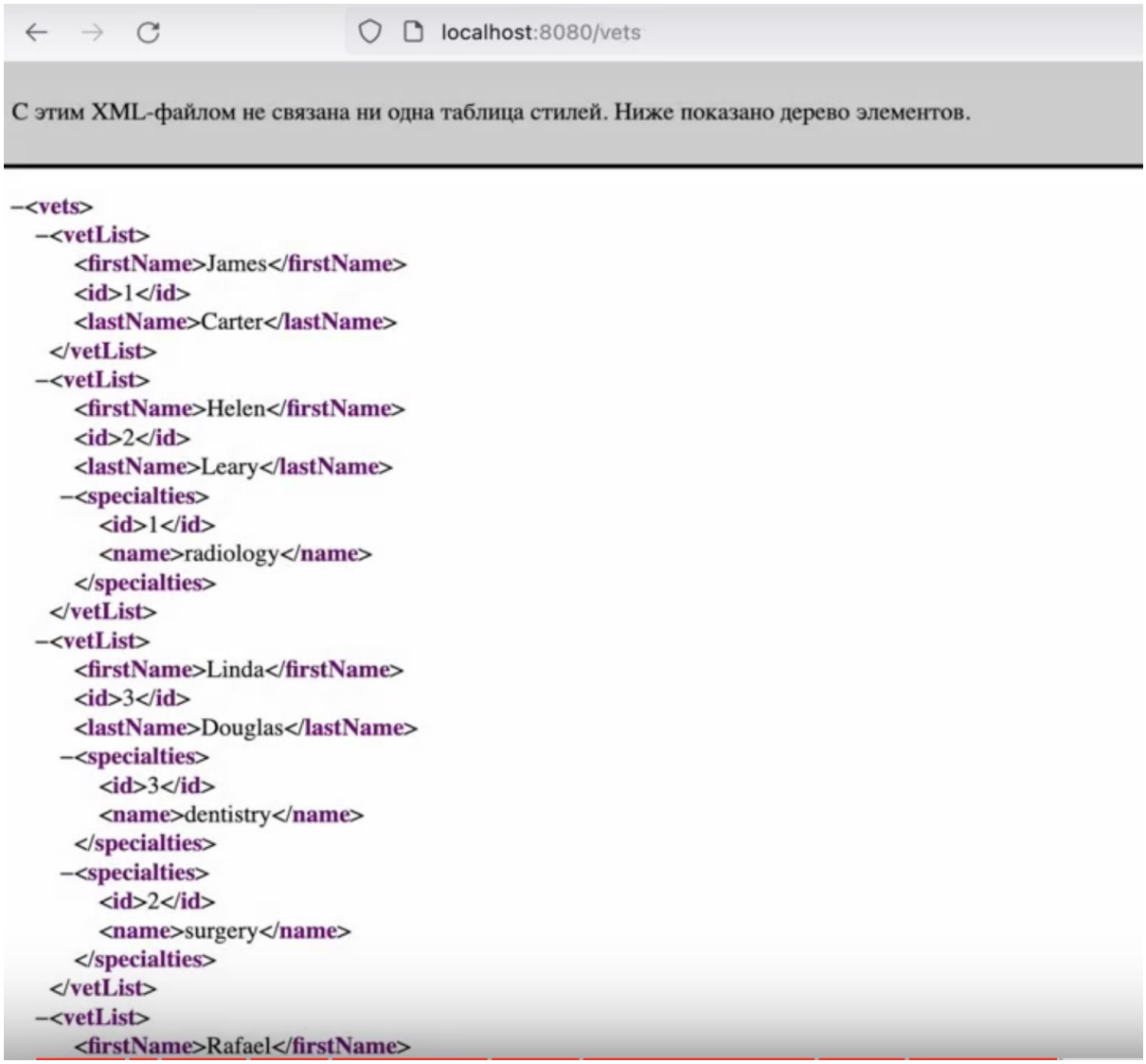
12. Нажмите на Add Authorize Rule и добавьте правило авторизации Permit All в поле Permissions



13. Нажмите OK
    Amplicode внесет изменения в существующие параметры доступа к эндпоинту в классе WebSecurityConfiguration:

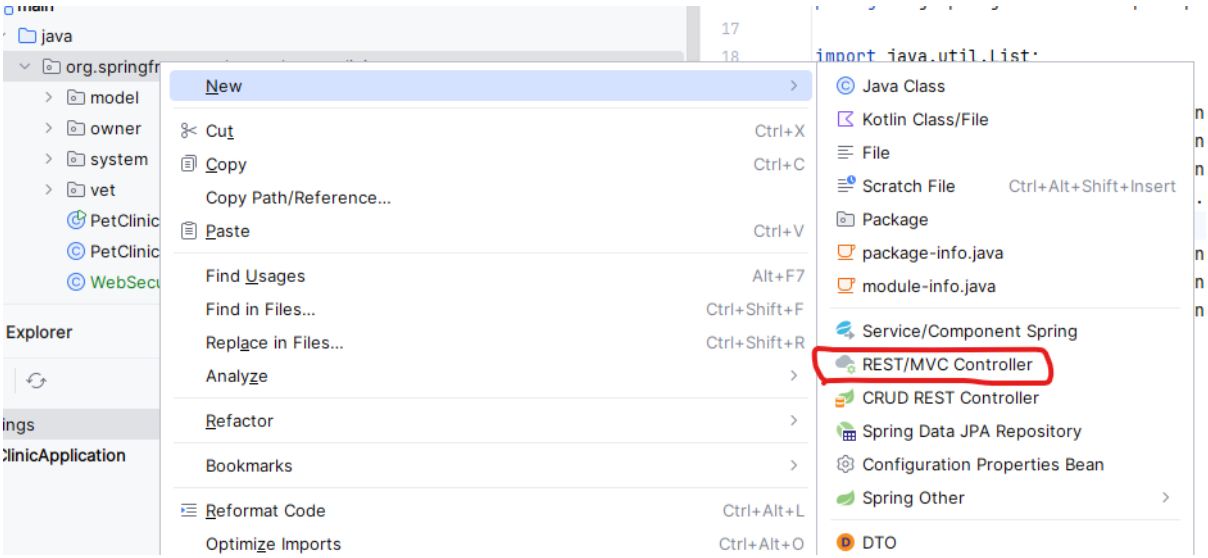
```
http.authorizeHttpRequests(authorizeHttpRequests -> authorizeHttpRequests
        .requestMatchers(HttpMethod.GET, "/vets").permitAll()
        .anyRequest().authenticated());
```

14. Перезапустите приложение и убедитесь в том, что все работает корректно. Теперь все приложение доступно только аутентифицированным пользователям, а единственный REST эндпоинт - всем без исключения.

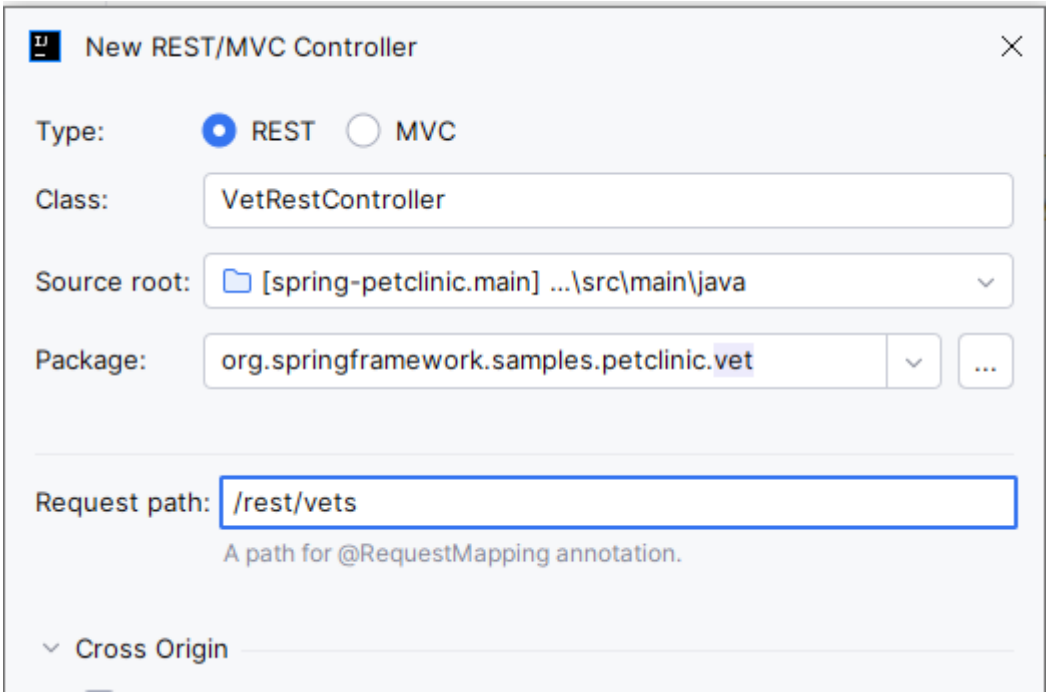

localhost:8080/vets
С этим XML-файлом не связана ни одна таблица стилей. Ниже показано дерево элементов.
−<vets>
−<vetList>
<firstName>James</firstName>
<id>1</id>
<lastName>Carter</lastName>
</vetList>
−<vetList>
<firstName>Helen</firstName>
<id>2</id>
<lastName>Leary</lastName>
−<specialties>
<id>1</id>
<name>radiology</name>
</specialties>
</vetList>
−<vetList>
<firstName>Linda</firstName>
<id>3</id>
<lastName>Douglas</lastName>
−<specialties>
<id>3</id>
<name>dentistry</name>
</specialties>
−<specialties>
<id>2</id>
<name>surgery</name>
</specialties>
</vetList>
−<vetList>
<firstName>Rafael</firstName>

3.7 Создание REST-контроллера с нуля

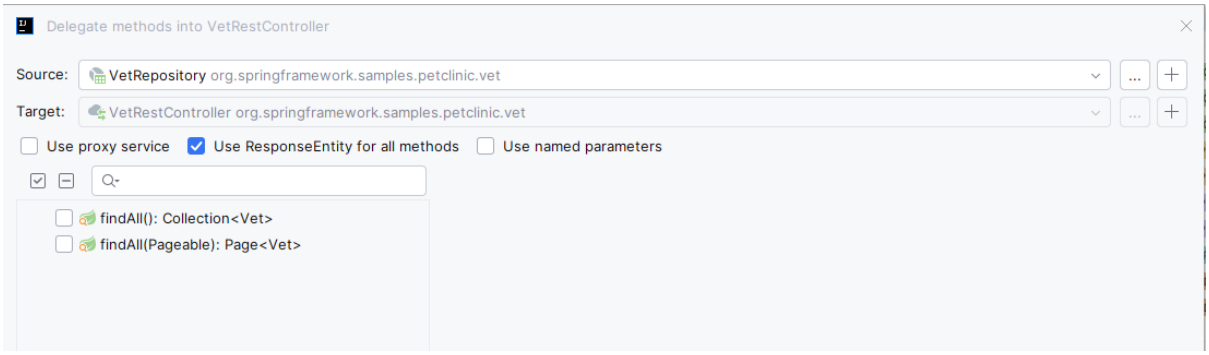
1. Прямо в дереве проекта вызовите действие REST/MVC Controller от Amplicode нажатием правой кнопки мыши



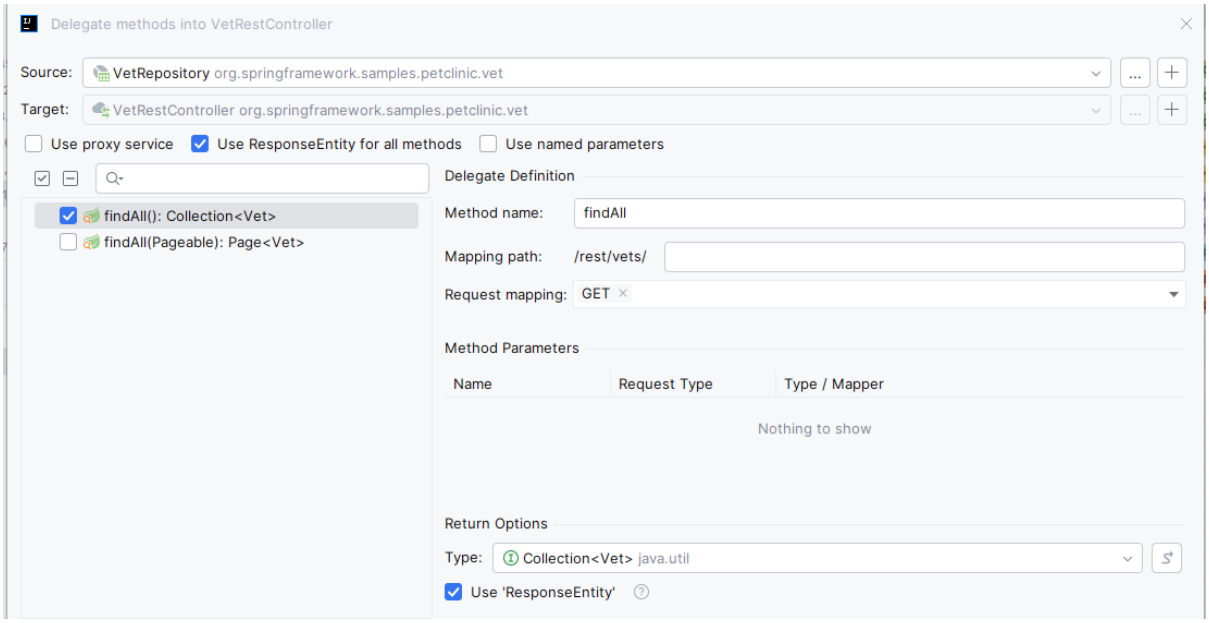
2. Оставьте тип REST → В открывшемся окне укажите название контроллера, пакет, в котором он будет создан, и путь, по которому он будет доступен, как показано ниже



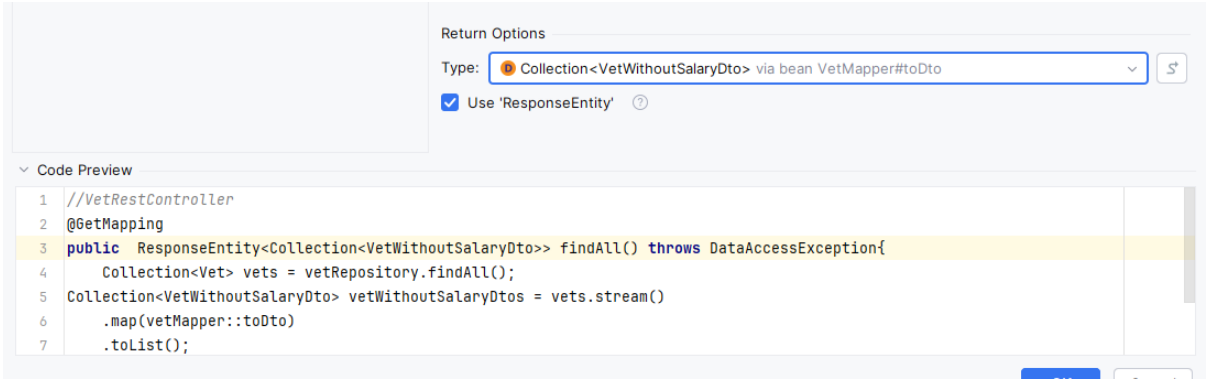
3. Нажмите OK
4. В Amplicode Designer выберите Request Handling → Delegate From. Amplicode автоматически выбрал интерфейс VetRepository в качестве источника для делегирования. Это как раз то, что нужно.



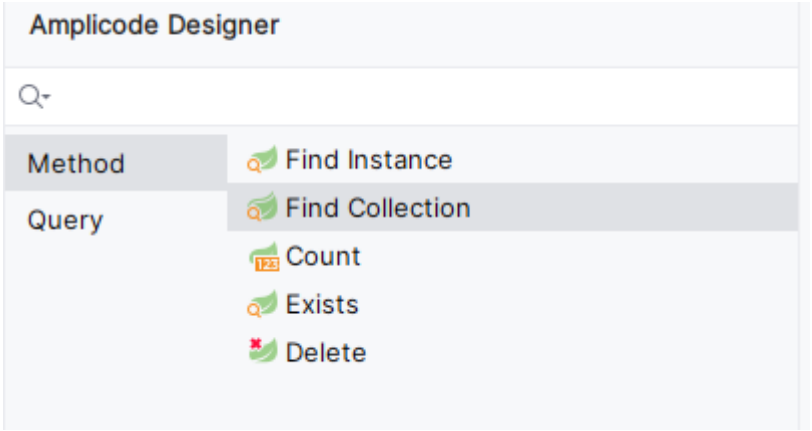
5. Выберите метод findAll. Его название, путь и тип запроса выбраны автоматически именно такие, которые и нужны.



6. Измените возвращаемый тип, используя выпадающий список в поле Type опции Return Options, так как у нас уже есть класс VetWithoutSalaryDto. Также, в Code Preview можно увидеть, как изменится код, если применить DTO

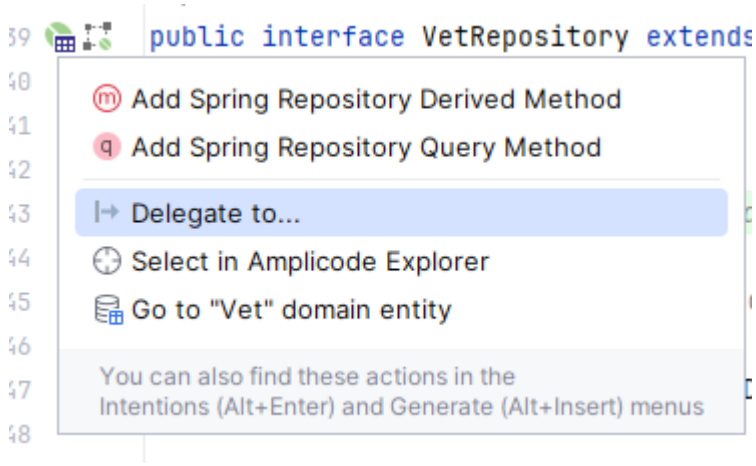


7. Нажмите OK.
8. Теперь необходимо написать второй метод, который будет отвечать за получение ветеринаров только определенной специальности. Для этого перейдите к классу VetRepository и создайте метод поиска коллекции сущностей: Amplicode Designer → Method → двойным щелчком мыши выберите Find Collection



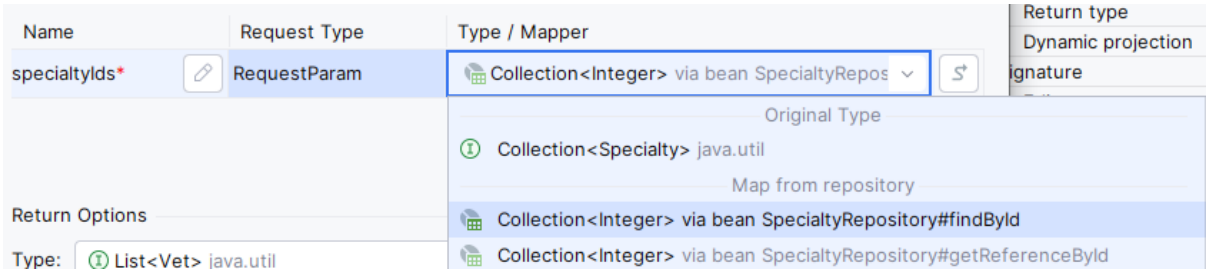
9. Выберите атрибут specialities и оператор условия in

10. Нажмите OK
11. Метод готов, и теперь следует делегировать его прямо в контроллер, воспользовавшись опцией Delegate to из Line Marker.

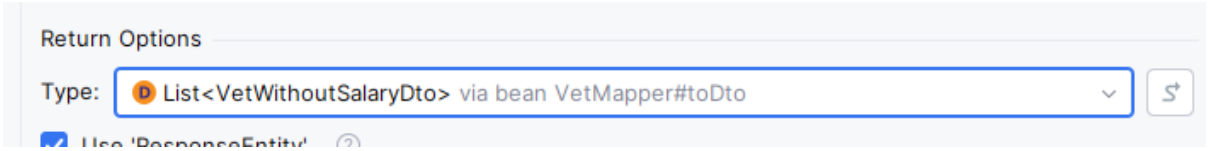

39
40
41
42
43
44
45
46
47
48
public interface VetRepository extends
Add Spring Repository Derived Method
Add Spring Repository Query Method
Delegate to...
Select in Amplicode Explorer
Go to "Vet" domain entity
You can also find these actions in the
Intentions (Alt+Enter) and Generate (Alt+Insert) menus

12. В открывшемся окне выберите VetRestController

13. В выпадающем списке для параметров метода Specialties выберите метод уже существующего репозитория SpecialtyRepository, который обернет полученный из запроса идентификатор в прокси-объект без дополнительных запросов в базу данных.

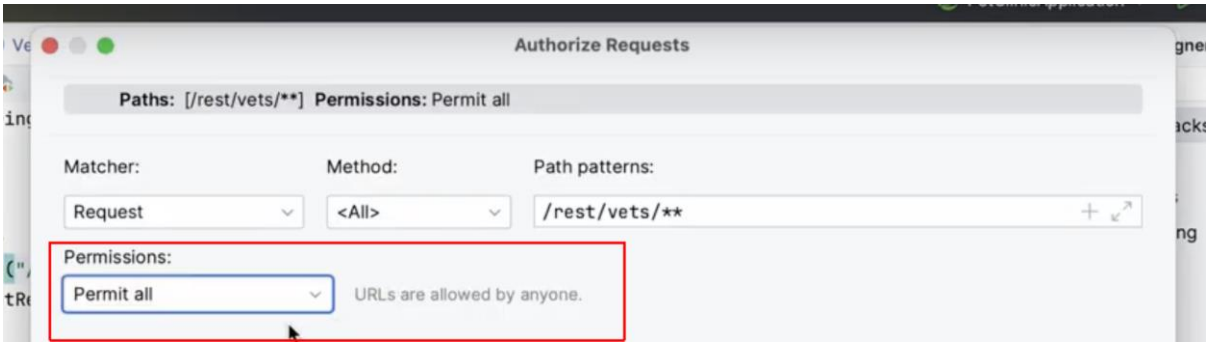


14. Измените возвращаемый тип на DTO и нажмите OK



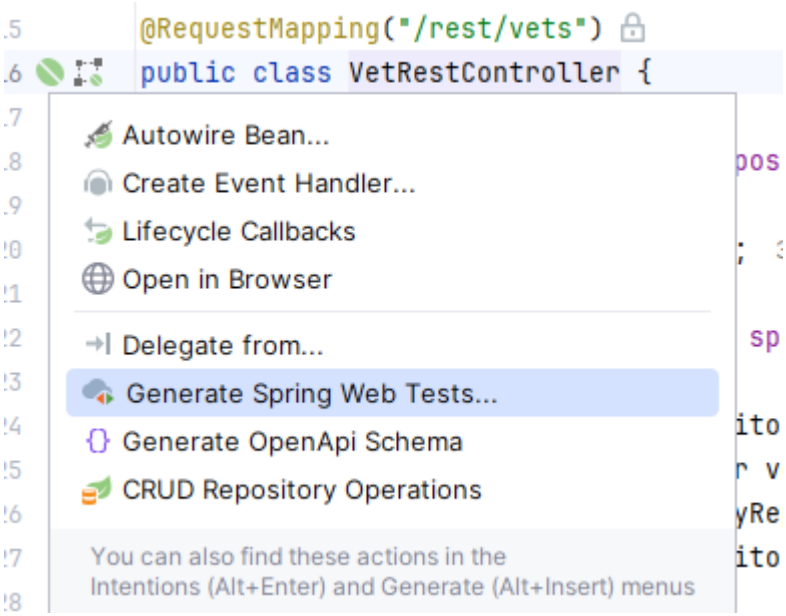
15. Через Amplicode Designer выберите путь к эндпоинту

16. Настройте Spring Security, сделав все REST эндпоинты общедоступными. Для этого воспользуйтесь замочком над классом и выберите в качестве правил доступа значение Permit all.

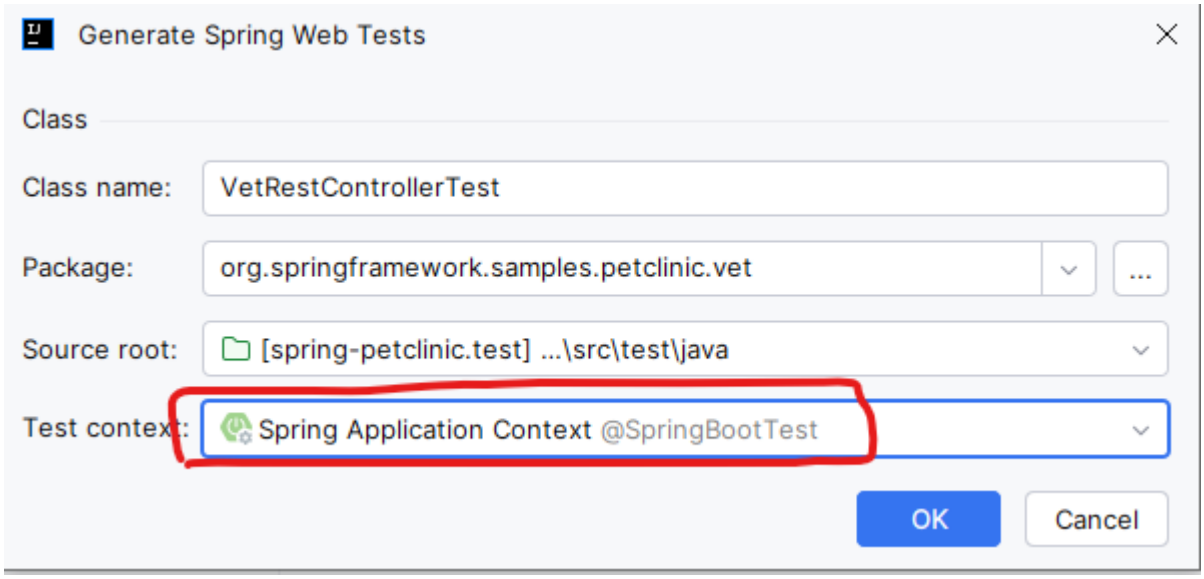


3.8 Тестирование эндпоинта

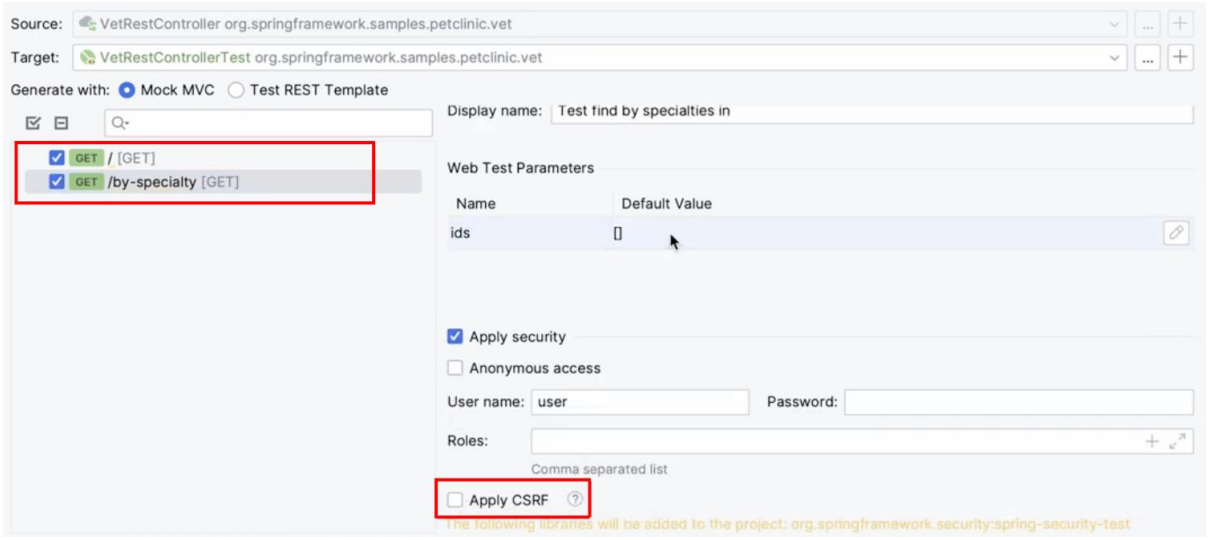
1. Откройте класс VetRestController → в Line Marker нажмите Generate Spring Web Tests



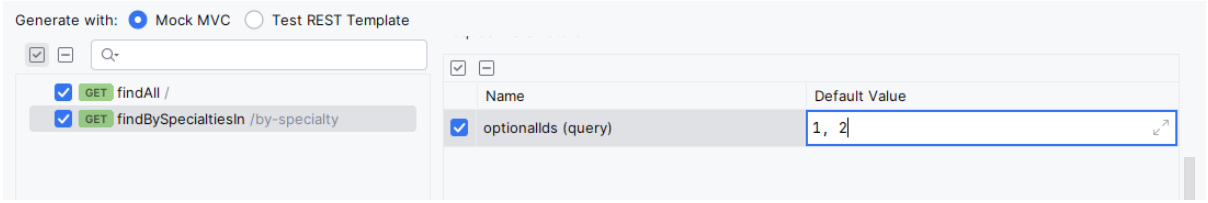
2. В диалоговом окне в поле Target нажмите Create (+) → Create Web Test → в качестве тестового контекста выберите Spring Boot Application.



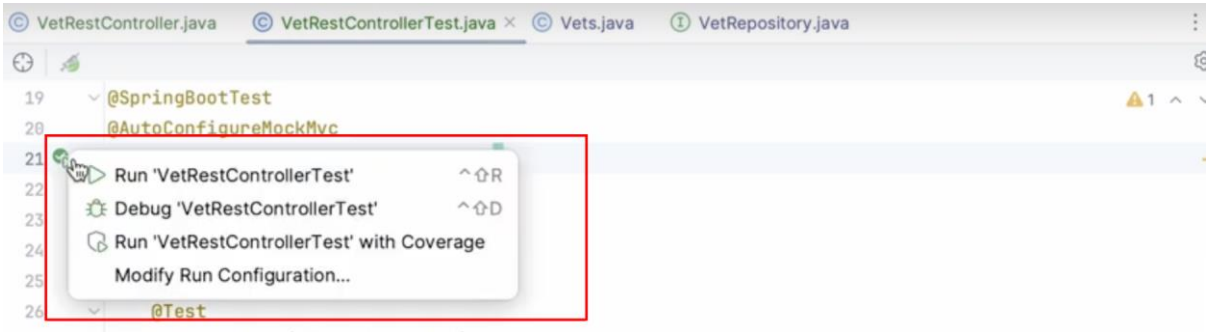
3. Выберите оба метода → выберите для обоих методов опцию Apply security и отключите CSRF



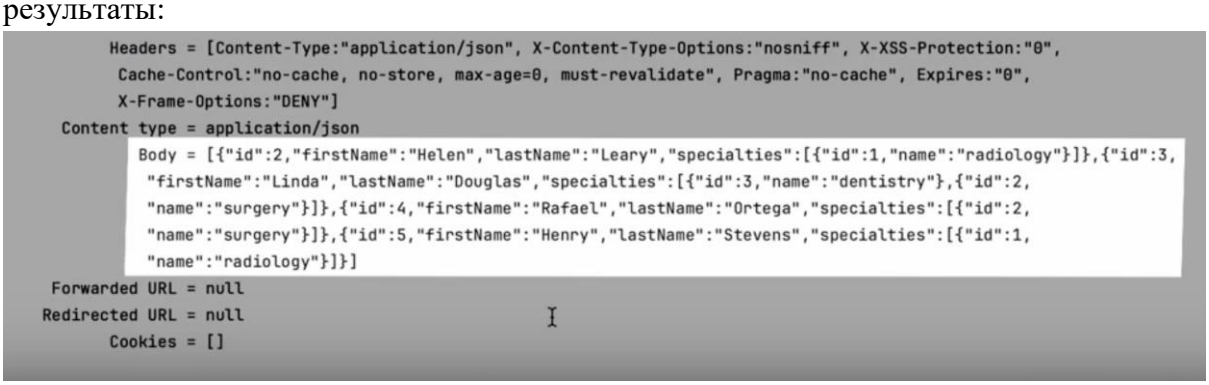
4. Для метода поиска по специальности добавьте значения 1, 2, как показано на рисунке.



5. Нажмите ОК
6. Запустите тесты, нажав специальную кнопку в исходном коде тестового класса



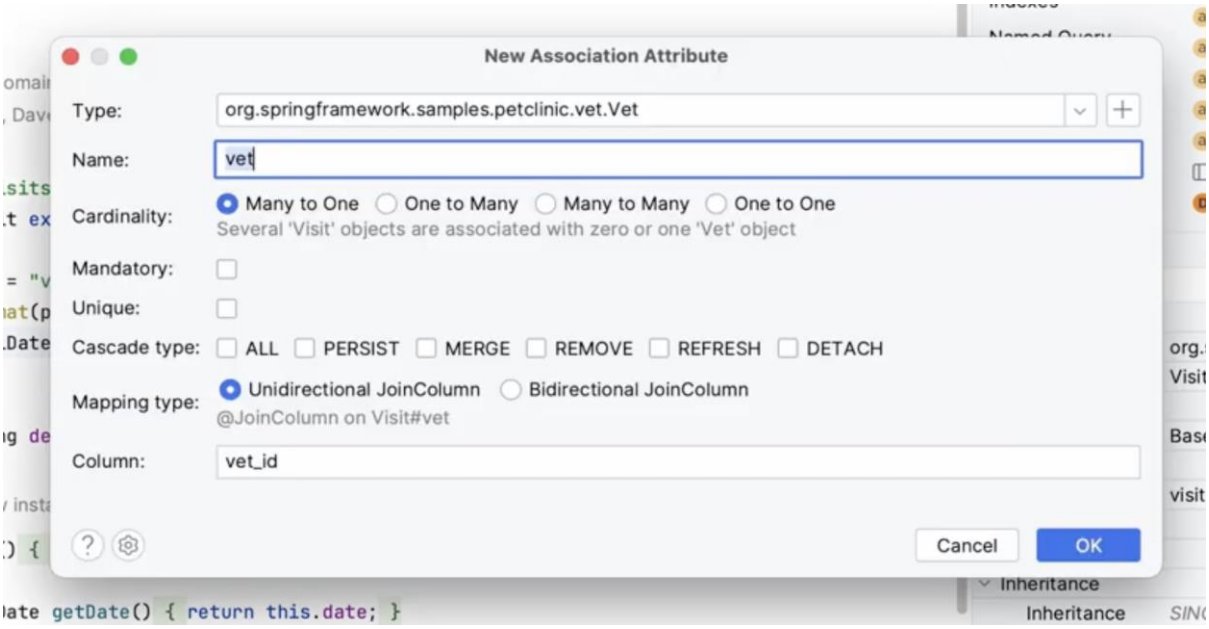
При удачном выполнении тестов должны появиться примерно следующие результаты:



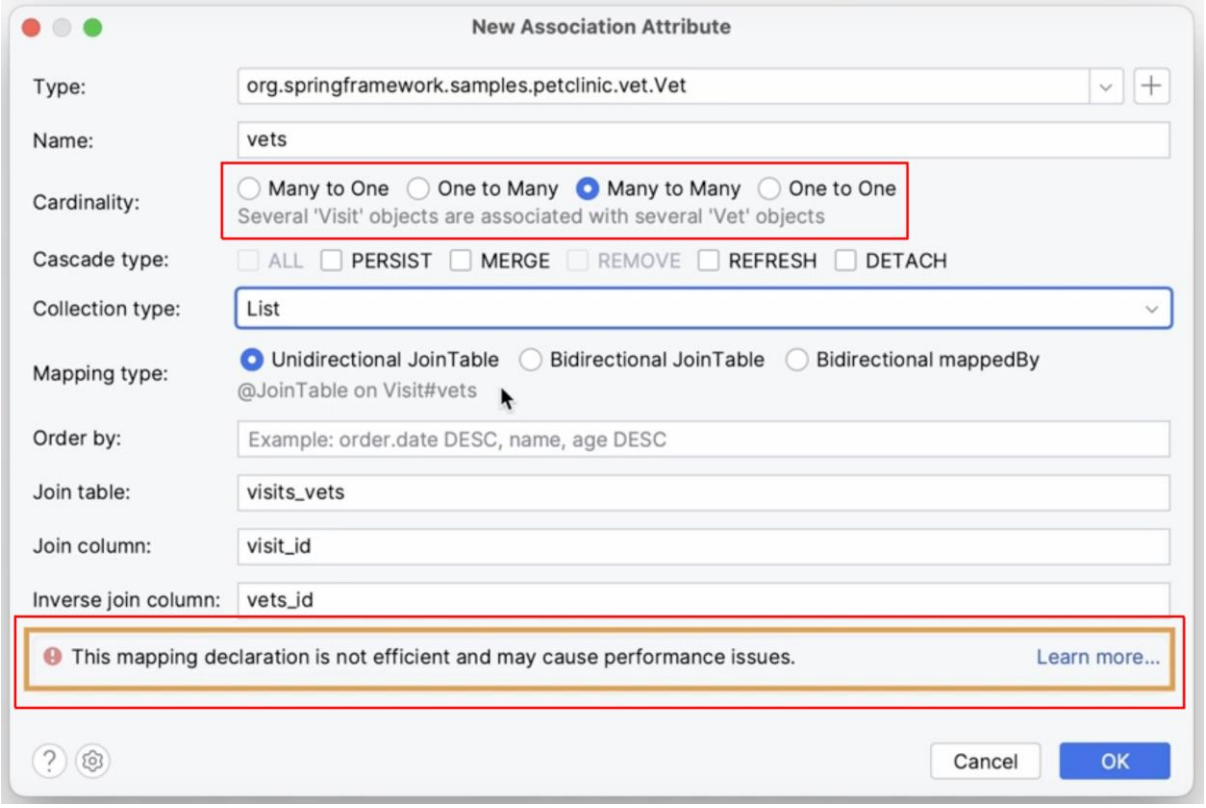
3.9 Изменение модели: добавление ассоциативной связи

Следующей задачей, которую предстоит решить, является реализация логики по автоматическому назначению ветеринара новому визиту. Для того чтобы реализовать подобную функциональность, сначала нужно вновь расширить существующую модель, добавив ассоциативную связь "Многие к одному" от визита к ветеринару. Воспользуйтесь уже знакомым нам действием по добавлению нового атрибута. В качестве типа атрибута укажем ветеринара.

1. Откройте сущность Visit → Amplicode Designer → выберите пункт Association
2. Выберите тип Vet



3. В поле Cardinality выберите связь Many to many. Все параметры в данном случае нас полностью устраивают. Стоит отметить, что Amplicode позволяет создавать любой тип ассоциативной связи; более того, он всегда сможет подсказать вам, какая из реализаций будет лучше в плане производительности, благодаря умным подсказкам.



4. Оставьте все параметры без изменений и нажмите OK. Атрибут готов. Не забудьте внести изменения на UI, а также сгенерировать скрипт миграции.

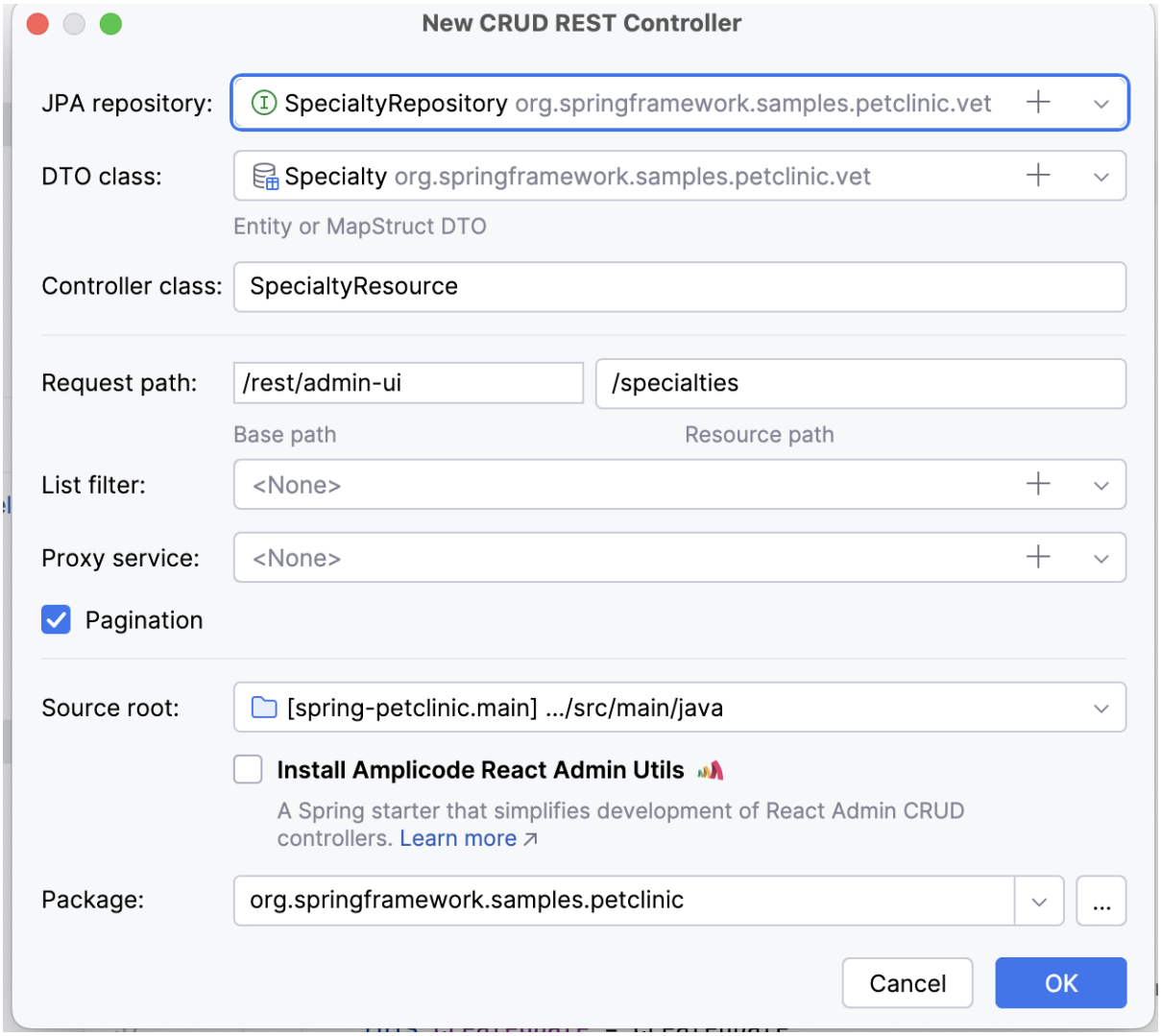
3.10 Добавление CRUD REST-контроллера

В Amplicode есть удобная функциональность по генерации CRUD REST-контроллера. При созданиии класса добавляются также все необходимые методы. А в самом классе контроллера можно добавлять CRUD методы уже по выбору. Давайте создадим подобный контроллер для новой сущности.

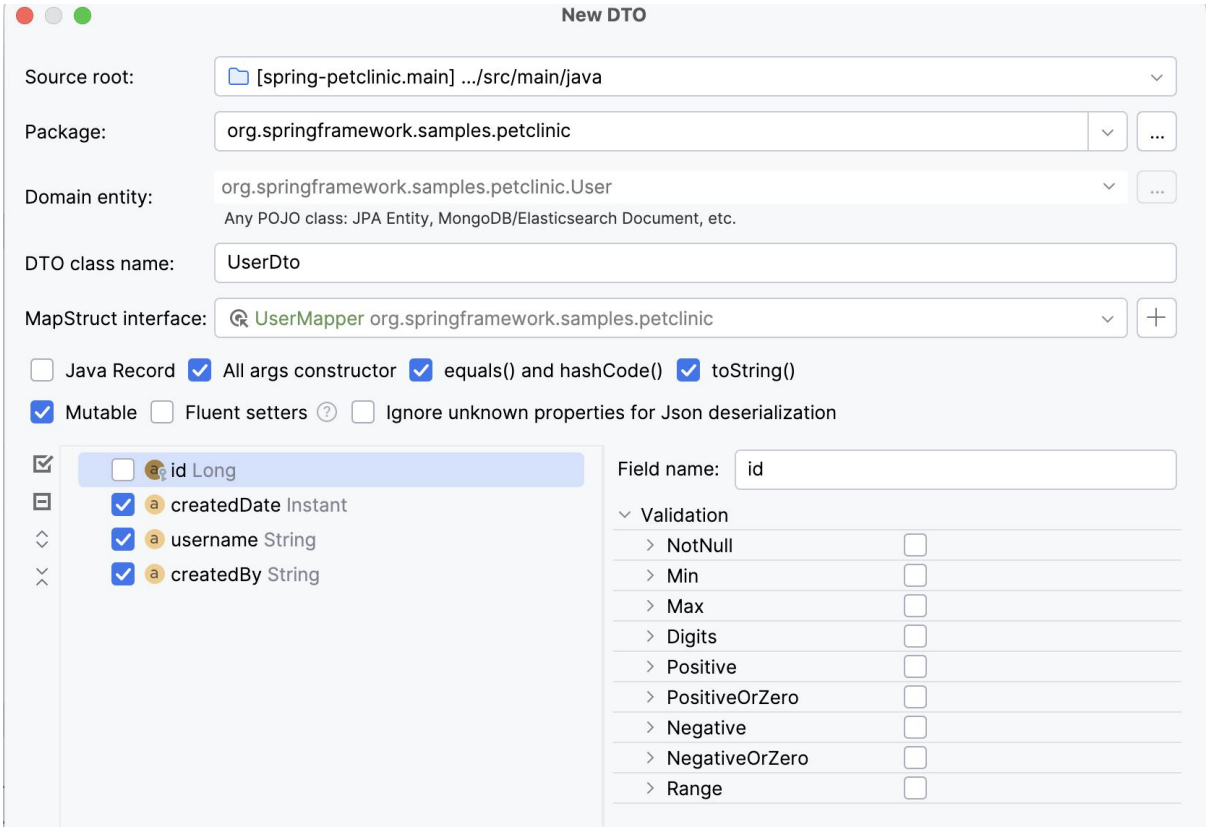
1. Amplicode Explorer → Create → JPA → JPA Entity
2. Введите наименование сущности User → выберите Id: Long → нажмите OK
3. Добавьте следующие атрибуты: String username, Instant createdDate, String createdBy. Ниже на рисунке показано, как выглядит в классе сущности:

```
@Id  2 usages
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id", nullable = false)
private Long id;

@Column(name = "created_date")  2 usages
private Instant createdDate;

@Column(name = "username")  2 usages
private String username;

@Column(name = "created_by")  2 usages
private String createdBy;
```

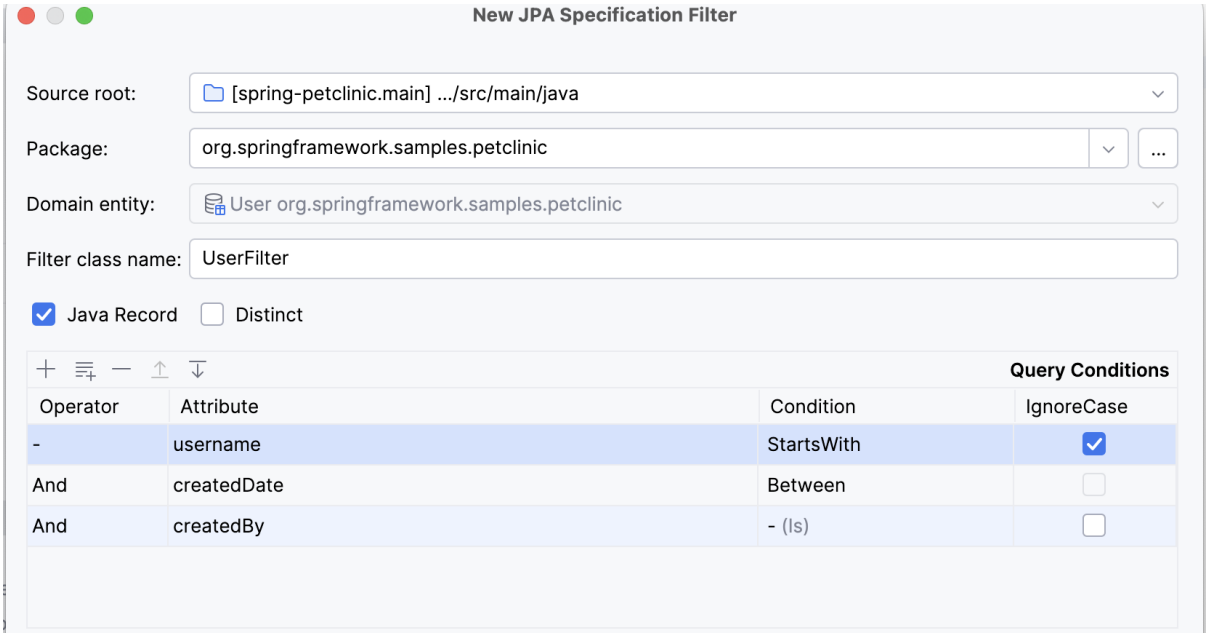
4. В дереве проекта нажать ПКМ → New → CRUD REST Controller



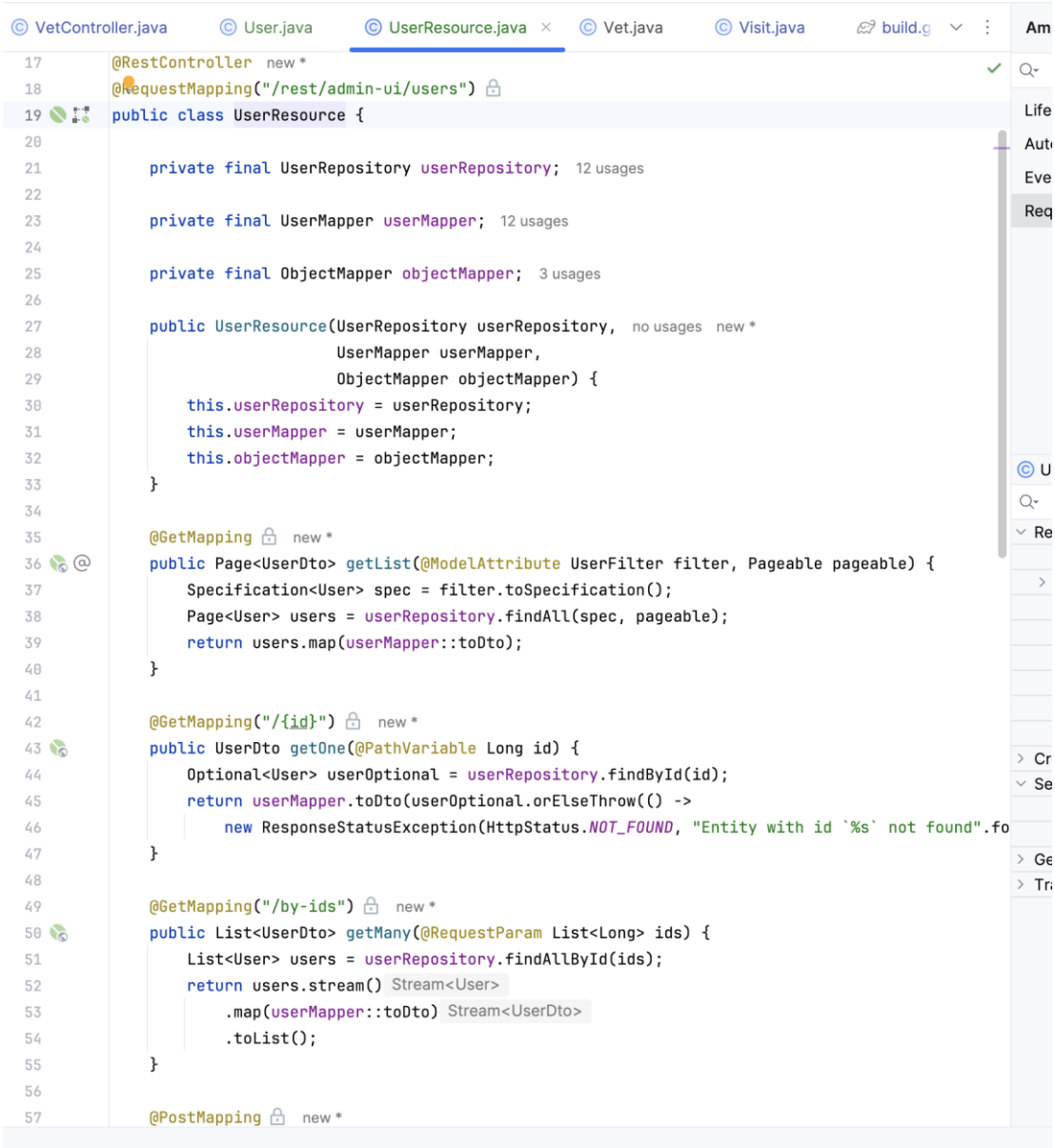
5. В поле JPA Repository нажмите Create (+) → для сущности User создадите новый UserRepository → OK. После этого действия имя контроллера автоматически сменилось на UserResource
6. Хорошей практикой является создание DTO классов для фиксации представления эндпоинта для внешнего мира. Поэтому в поле DTO Class нажмите Create (+) → в поле MapStruct interface нажмите + и создадите UserMapper → отметьте все поля, кроме id



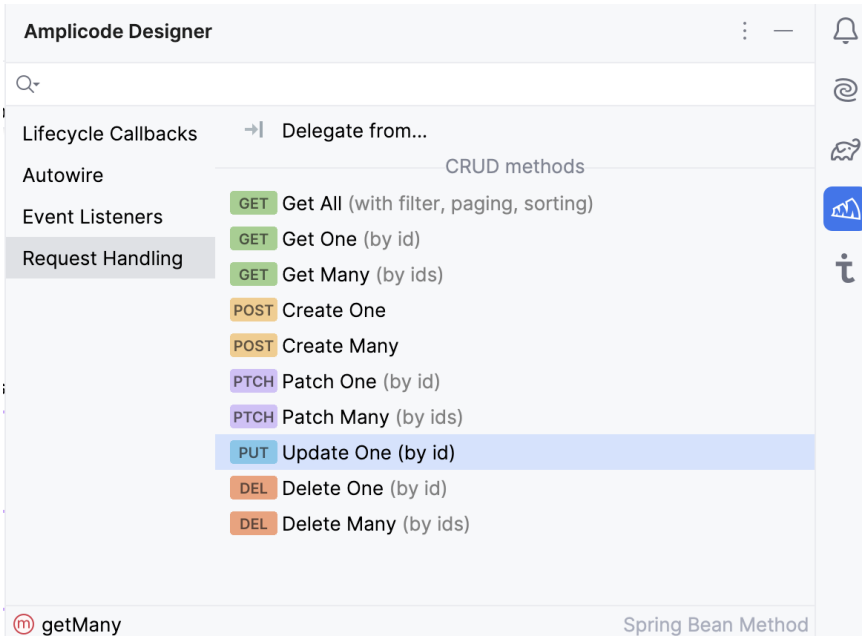
7. Нажмите OK
8. Настроим фильтрации через создание фильтра прямо из диалогового окна создания CRUD REST-контроллера. Для этого в поле List Filter нажмите + и настройте критерии фильтрации данных, как показано на картинке ниже:



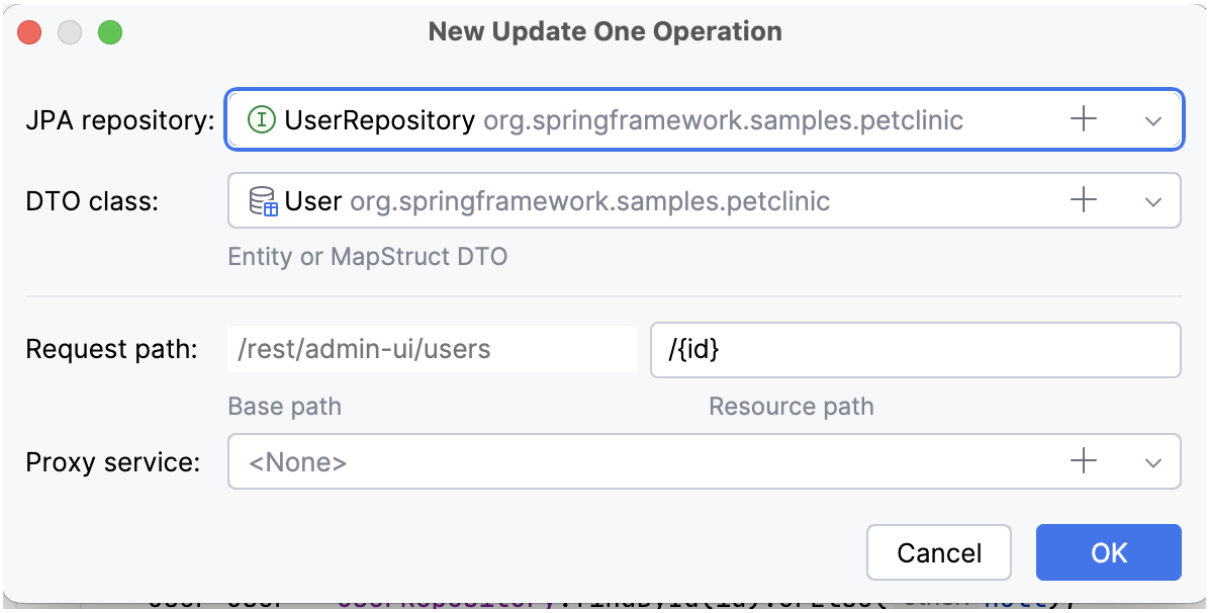
9. Нажмите OK и создадите CRUD REST Controller. Amplicode создал контроллер с 8 эндпоинтами, а также и все остальные классы.



10. Помимо этого, Amplicode позволяет создавать CRUD методы непосредственно из самого класса контроллера. Откройте только что созданный UserResource → Amplicode Designer → Request Handling



11. Двойным кликом выберите Update One → в появившемся диалоговом окне выберите UserRepository

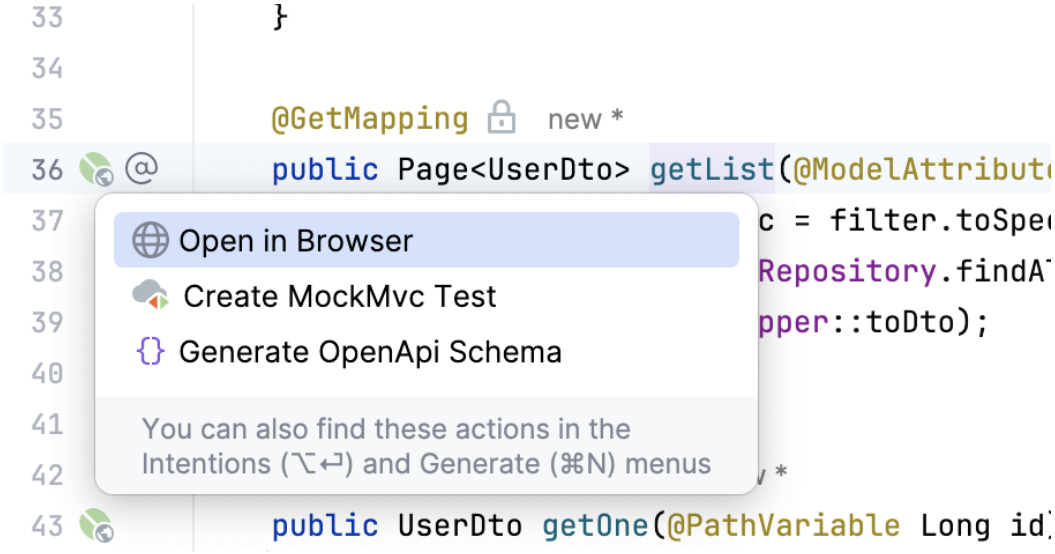


12. Нажмите OK. Amplicode успешно сгенерировал корректный эндпоинт update с аннотацией @PutMapping

```
@PutMapping("/{id}")  new *
public User update(@PathVariable Long id, @RequestBody User user) {
    if (!userRepository.existsById(id)) {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND,
            "Entity with id `%s` not found".formatted(id));
    }
    return userRepository.save(user);
}
```

13. Теперь давайте проверим корректность работы контроллера. Запустите приложение

14. Рядом с методом getList нажмите на Gutter icon Show method actions → Open in Browser



15. Введите admin / admin, если требуется авторизация. На экран выведется примерно следующая информация

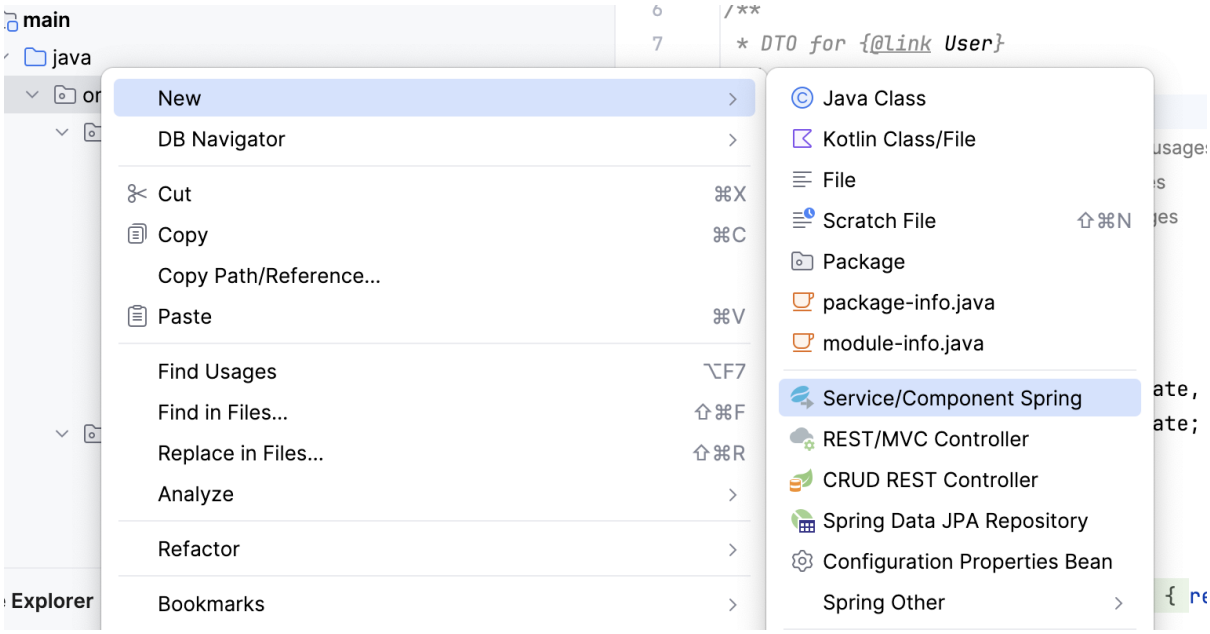
Автоформатировать


```
{
  "content": [],
  "pageable": {
    "sort": {
      "empty": true,
      "sorted": false,
      "unsorted": true
    },
    "offset": 0,
    "pageNumber": 0,
    "pageSize": 20,
    "paged": true,
    "unpaged": false
  },
  "totalElements": 0,
  "totalPages": 0,
  "last": true,
  "size": 20,
  "number": 0,
  "sort": {
    "empty": true,
    "sorted": false,
    "unsorted": true
  },
  "numberOfElements": 0,
  "first": true,
  "empty": true
}
```

3.11 Написание бизнес-логики

Бизнес логику в Spring приложениях принято реализовывать в классах, помеченных аннотацией @Service. Давайте создадим такой класс.

1. Нажмите на Service/Component Spring в меню New, как показано на рисунке.



2. Введите наименование класса VetAutoAssignmentService и нажмите ОК
3. Логика нахождения подходящего ветеринара будет довольно простой. В случае, если у питомца уже были визиты в клинику, то выберем последнего ветеринара, работавшего с питомцем. В случае же, если визитов ранее не было, будем назначать любого из ветеринаров с профессией хирурга.
   Для начала добавьте атрибут vet для сущности Visit, если еще не добавлен:

```
@ManyToOne(fetch = FetchType.LAZY)  2 usages
@JoinColumn(name = "vet_id")
private Vet vet;

public Vet getVet() {  new *
    return vet;
}

public void setVet(Vet vet) {  new *
    this.vet = vet;
}
```

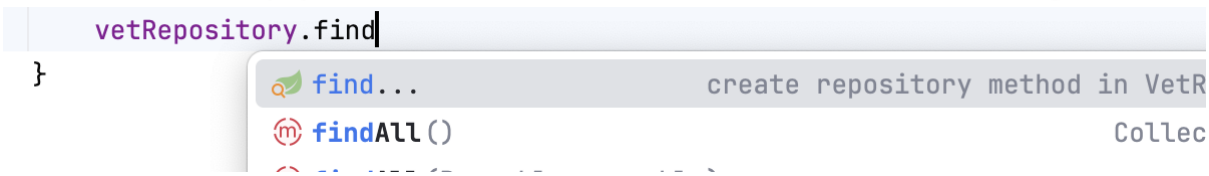
   И создадите метод в сервисе VetAutoAssignmentService, как показано на рисунке:

```
public Vet findAppropriateVet(Pet pet) {  no usages  new *
    List<Visit> visits = pet.getVisits()  Collection<Visit>
        .stream()  Stream<Visit>
        .toList();
    if (visits.isEmpty() || visits.stream().noneMatch(visit -> visit.getVet() != null)) {

    }
```

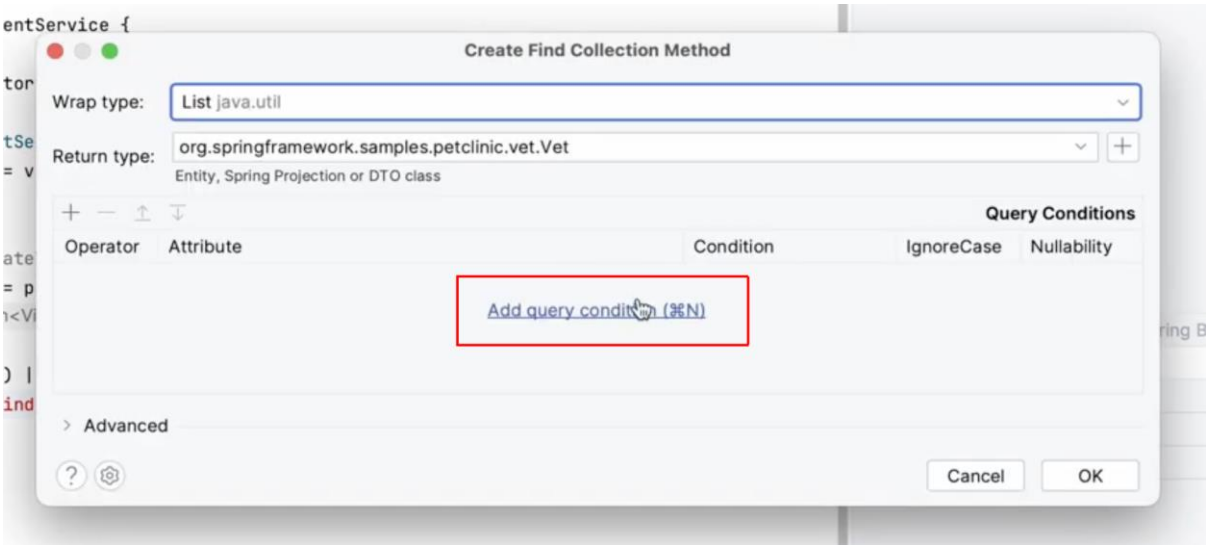
4. Начните писать имя нужного нам Bean, выбрав его из списка, тем самым Amplicode добавит все автоматически

```
private final VetRepository vetRepository;  3 usages

public VetAutoAssignmentService(VetRepository vetRepository) {  no usages  new *
    this.vetRepository = vetRepository;
}

public Vet findAppropriateVet(Pet pet) {  no usages  new *
    List<Visit> visits = pet.getVisits()  Collection<Visit>
        .stream()  Stream<Visit>
        .toList();
    if (visits.isEmpty() || visits.stream().noneMatch(visit -> visit.getVet() != null)) {
        vetRepository
    }
```

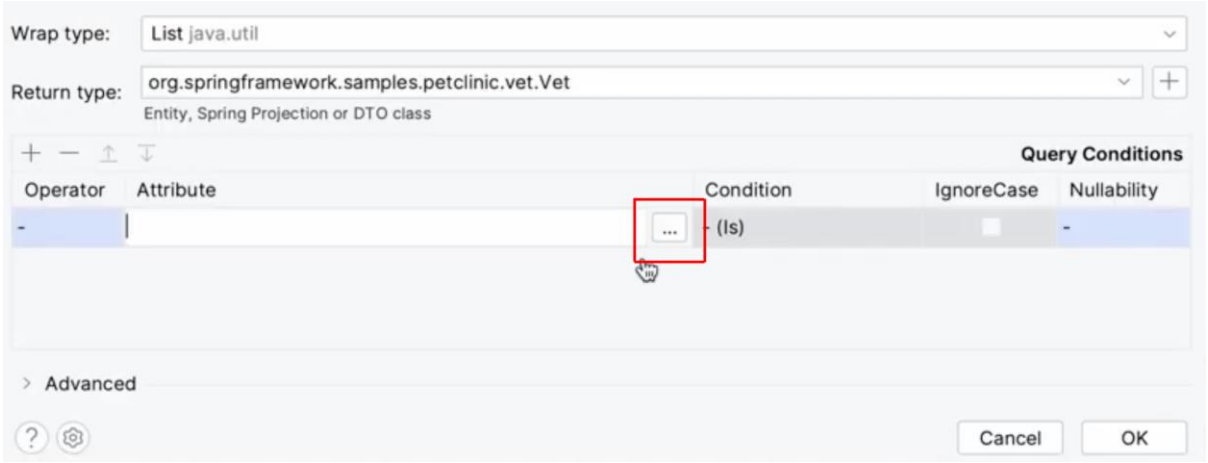
5. После vetRepository поставьте точку и начните писать find и выберите первый вариант с многоточием



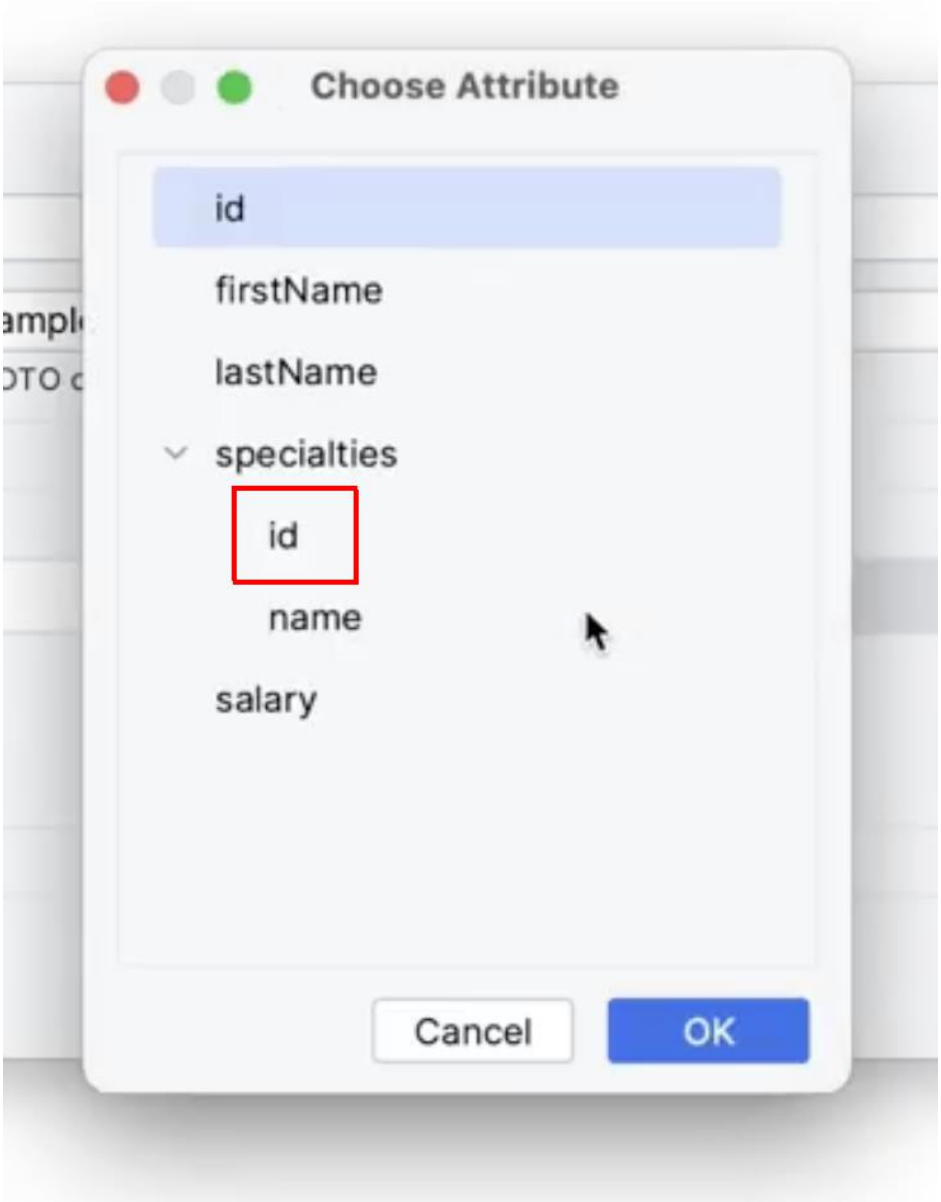
6. Кликните на Method Find Collection. После чего Amplicode откроет диалог создания нового метода в уже существующем репозитории. Для этого даже не требуется покидать текущий редактор кода. Нажмите на ссылку Add query condition.



7. Нажмите на икону с тремя точками в колонке Attribute в таблице



8. Выберите поле id из specialties

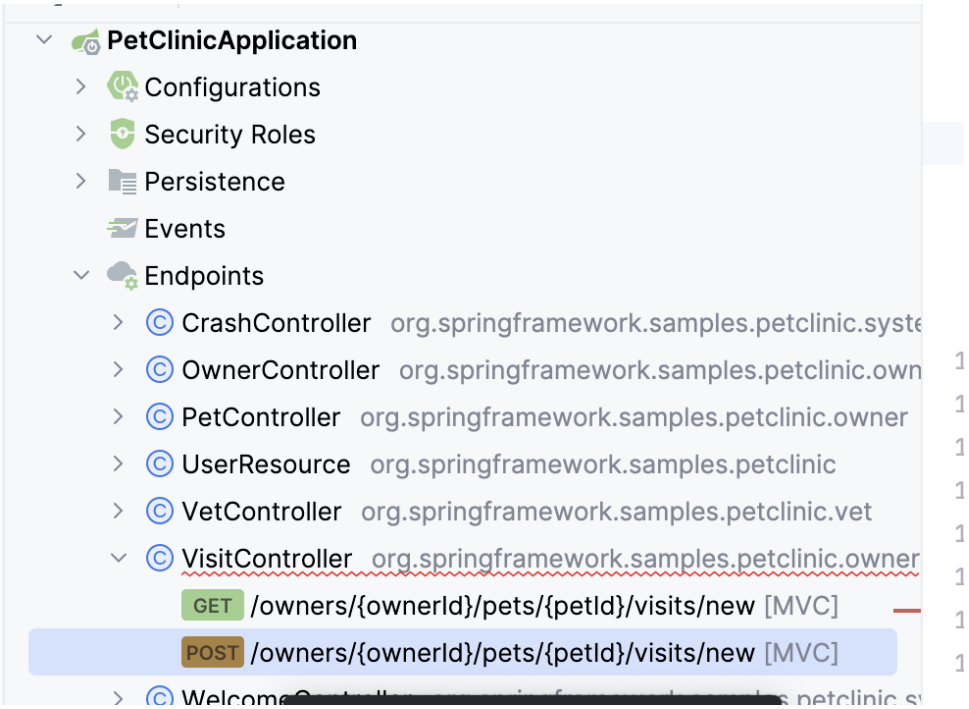


9. Нажмите OK → Выберите условие In из списка в колонке Condition → Нажмите OK

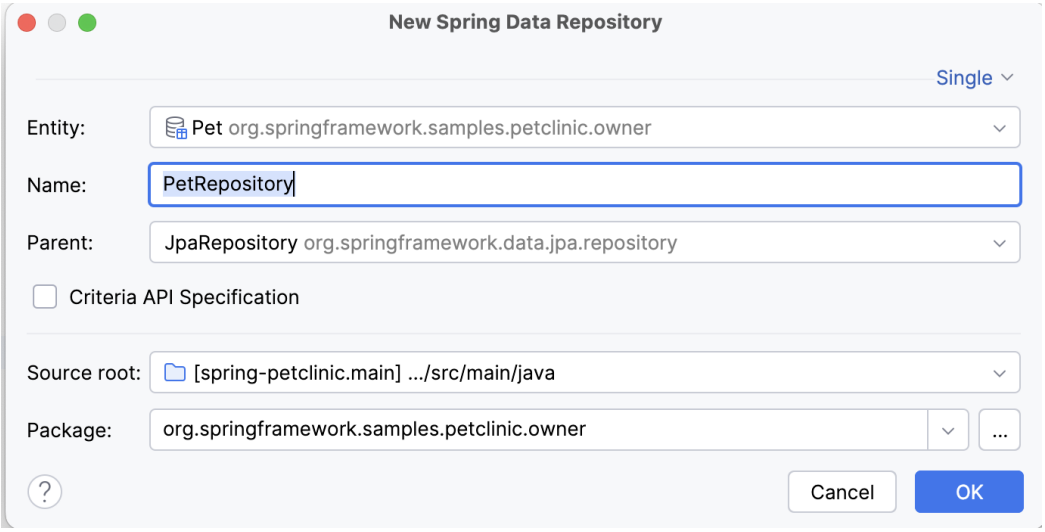
10. Метод в репозитории создан, и его вызов сгенерирован в нашем сервисе. Осталось только реализовать логику нахождения ветеринара, работавшего последним с нашим питомцем. Финальный код метода будет иметь примерно следующий вид:

```
@Service  new *
public class VetAutoAssignmentService {

    private static final Integer SURGERY_ID = 2;  1 usage
    private final VetRepository vetRepository;  2 usages

    public VetAutoAssignmentService(VetRepository vetRepository) {  no usages  new *
        this.vetRepository = vetRepository;
    }

    public Vet findAppropriateVet(Pet pet) {  no usages  new *
        List<Visit> visits = pet.getVisits()  Collection<Visit>
            .stream()  Stream<Visit>
            .toList();
        if (visits.isEmpty() || visits.stream().noneMatch(visit -> visit.getVet() != null)) {
            vetRepository.findBySpecialties_IdIn(Collections.singleton(SURGERY_ID))  List<Vet>
                .stream()  Stream<Vet>
                .findAny()  Optional<Vet>
                .orElseThrow();
        }
        return visits.stream()  Stream<Visit>
            .filter(v -> v.getVet() != null)
            .max(Comparator.comparing(Visit::getDate))  Optional<Visit>
            .orElseThrow()  Visit
            .getVet();
    }
}
```

11. Сервис готов. Через Amplicode Explorer найдите эндпоинт создания нового визита и перейдите к нему



12. Обратитесь к сервису по имени vetAutoAssignmentService: **vetAutoAssignmentService.findAppropriateVet(pet);**
13. В текущем эндпоинте нет объекта pet, есть только его идентификатор. Заинжектируйте репозиторий для сущности Pet и обратитесь к методу getReferenceById, чтобы обернуть идентификатор в объект. Начните писать pet и выберите petRepository



14. В диалоговом окне создания нового репозитория нажмите ОК, после чего будет создан новый класс в проекте
15. Напишите точку после petRepository и выберите метод getReferenceById()
16. Внутри метода введите petId и дополните код, чтобы получилось как на картинке

```
Pet pet = petRepository.getReferenceById(petId);
Vet vet = vetAutoAssignmentService.findAppropriateVet(pet);
visit.setVet(vet);
```

17. Запустите приложение и попробуйте создать новый визит. Если все сделано правильно, ветеринар будет успешно назначен автоматически.