**Предметная область** данной курсовой работы связана с разработкой интерактивного приложения, основанного на концепции виртуального питомца (по аналогии с популярной когда-то игрушкой «Тамагочи»). Ниже представлены ключевые аспекты предметной области**:**

1. **Для чего создается приложение:**Приложение разрабатывается с целью обеспечить пользователю возможность взаимодействия с виртуальным питомцем. Для конечного пользователя приложение будет выступать в роли развлекательного интерактивного средства, а с точки зрения учебного процесса — в качестве демонстрации знаний и навыков, полученных при изучении языка Java и сопутствующих технологий.
2. **Что из себя будет представлять данное приложение:**Приложение выступит в роли игры, в которой у каждого зарегистрированного пользователя будет свой виртуальный питомец. Клиентская часть (JavaFX) обеспечит удобный графический интерфейс, позволяющий пользователю:
   * Авторизоваться в системе с помощью учётных данных.
   * Просматривать состояние своего питомца: его настроение, здоровье, сытость, уровень энергии и прочие параметры.
   * Взаимодействовать с питомцем: кормить, поить, играть с ним, давать отдых и выполнять другие действия, влияющие на его характеристики.

Серверная часть (на основе Spring) будет отвечать за:

* + Хранение данных о пользователях и питомцах в базе данных (PostgreSQL).
  + Обработку запросов от клиента: аутентификация пользователей, получение и обновление данных о питомце.
  + Логику изменения параметров питомца с учетом действий, совершаемых пользователем.

Таким образом, итоговое приложение — это система, способная в режиме «клиент-сервер» обеспечивать доступ к данным о питомце, обновлять его состояние и предоставлять удобный пользовательский интерфейс для игрового взаимодействия.

1. **Какова цель приложения:**Цель проекта — создание учебного демонстрационного продукта, который реализует основные принципы клиент-серверной архитектуры, обеспечивает авторизацию и хранение данных о пользователях, а также управляет жизненным циклом виртуального питомца. С точки зрения концепции «Тамагочи», цель — поддерживать интерес пользователя к своему питомцу, стимулировать его время от времени заходить в приложение, ухаживать за виртуальным существом и наблюдать, как оно изменяется с течением времени.

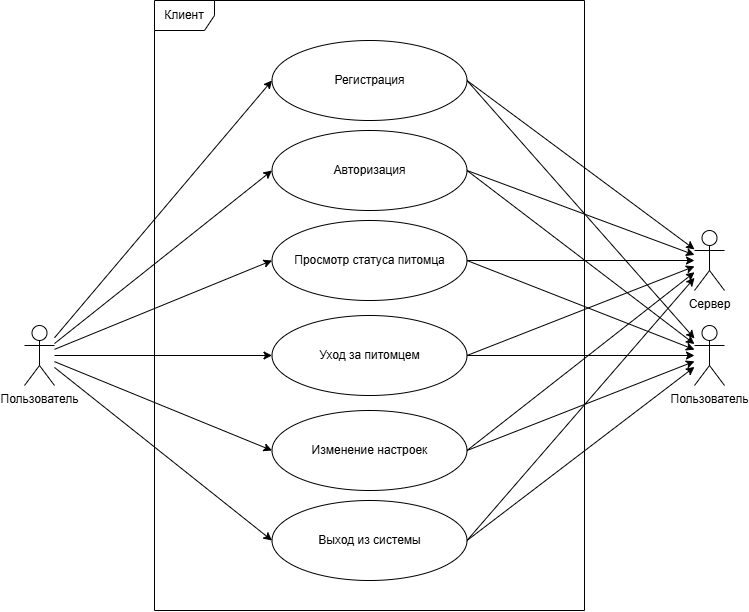
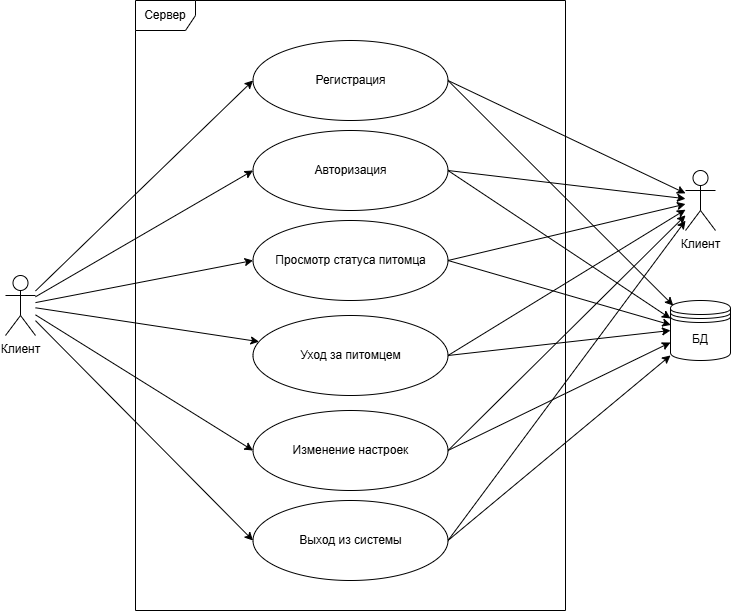


Диаграмма (1)

  
Диаграмма (2)

**Диаграммы прецедентов**

**ДИАГРАММА (1)**

**Общее описание:**

На диаграмме мы видим систему, в которой пользователь взаимодействует с клиентским приложением, а клиентское приложение, в свою очередь, обращается к серверу для выполнения определённых действий.

**Акторы и границы системы:**

* Слева находится **Пользователь** (фигура человечка), который совершает действия через клиентское приложение. Пользователь инициирует различные прецеденты, расположенные внутри границы «Клиент».
* Внутри прямоугольной области «Клиент» перечислены основные функции (прецеденты), которые доступны пользователю:
  + **Регистрация**
  + **Авторизация**
  + **Просмотр статуса питомца**
  + **Уход за питомцем**
  + **Изменение настроек**
  + **Выход из системы**

Эти овалы отражают функциональность приложения, доступную на клиентской стороне.

**Логика взаимодействия:**

1. **Пользователь** (слева) через интерфейс клиентского приложения инициирует запросы: например, хочет авторизоваться, зарегистрироваться или получить статус питомца.
2. **Клиентское приложение** (представленное прямоугольной областью) принимает эти действия и отправляет соответствующие запросы к **Серверу**.
3. **Сервер**, получив запрос, осуществляет всю бизнес-логику: проверяет данные пользователя, обращается к базе данных, обновляет информацию о питомце, настройки, и т.д. Затем формирует ответ на запрос.
4. После обработки запроса на сервере результат возвращается обратно клиентской части, а затем клиент отображает эти результаты конечному пользователю.

**Зачем показан пользователь и на стороне сервера?**

Это может быть условным указанием на то, что все операции сервера тесно связаны с конкретным пользователем (идентификация, проверка прав, получение данных пользователя). Таким образом, диаграмма подчеркивает двусторонний характер взаимодействия: пользователь → клиент → сервер → обратно к пользователю. Такой подход использован, чтобы подчеркнуть логику и маршрут данных.

**Диаграмма (2)**

**Общее описание:**

В центре показан Сервер, внутри которого находятся основные прецеденты.

Слева мы видим одного и того же Клиента, представленного в виде человечка (пользователя), а также овалов прецедентов внутри границы сервера. Основная идея в том, что клиентское приложение инициирует запросы к серверу и получает от него ответы, а сервер в свою очередь работает с базой данных (БД) для получения и сохранения информации.

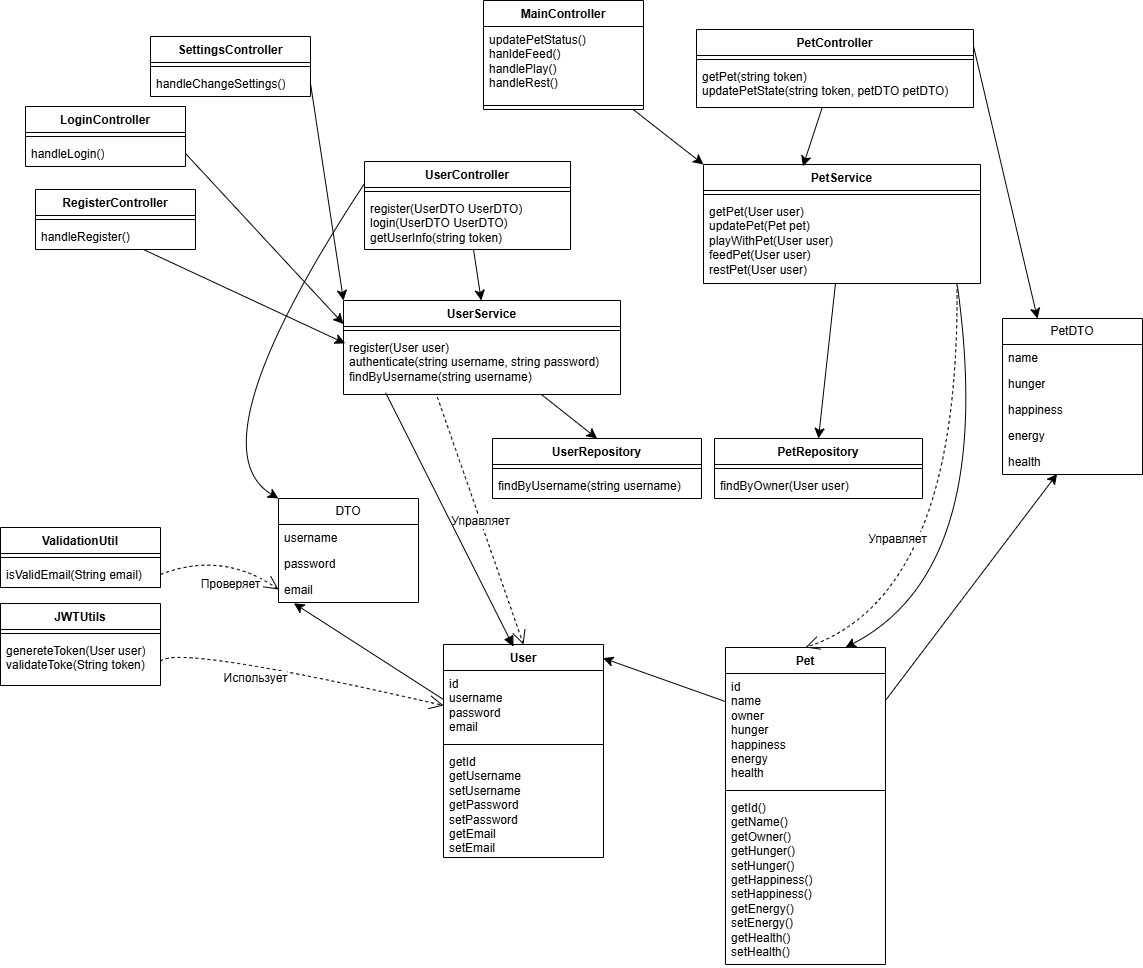
**Логика взаимодействия:**

1. Клиент (слева) — это пользовательское приложение (UI), с которого пользователь управляет своим питомцем. Клиент взаимодействует с сервером, отправляя запросы (например, на регистрацию, авторизацию, просмотр статуса питомца и т.д.).
2. Сервер (в центре) — это компонент, принимающий запросы от клиента. Он выполняет всю бизнес-логику:
   * При получении запроса от клиента сервер проверяет права доступа, состояние пользователя, состояние питомца и т.д.
   * Для выполнения операции сервер может обратиться к базе данных (например, чтобы сохранить нового пользователя, обновить параметры питомца, получить текущее состояние).
   * После выполнения логики сервер формирует ответ, который отправляет обратно клиенту.
3. База данных (БД, справа внизу) — система хранения данных. Сервер получает и обновляет информацию в БД:
   * Сохранение новых пользователей, питомцев.
   * Обновление параметров питомца.
   * Извлечение необходимых данных для формирования ответа клиенту.
4. Клиент (справа) в данной диаграмме — это условное обозначение того же самого клиента, который изначально находится слева. Иными словами, мы имеем дело с двухсторонним взаимодействием:
   * Клиент отправил запрос слева → Сервер обработал запрос → Сервер отправил ответ обратно клиенту (это можно интерпретировать как взаимодействие, где клиент как будто бы присутствует с обеих сторон: инициирует взаимодействие и получает результат).

По сути, клиент один и тот же, просто диаграмма пытается отразить двунаправленный характер обмена данными: клиент не только запрашивает, но и получает ответы от сервера. Сервер — это промежуточное звено.

**Главная мысль:**

Клиент-серверное приложение работает по схеме запрос-ответ. Клиент (одно и то же лицо/приложение) шлёт запросы на определённые действия (регистрация, авторизация, уход за питомцем и т.д.). Сервер обрабатывает эти запросы, взаимодействует с базой данных, формирует ответы и возвращает их тому же клиенту. В результате клиент, инициируя действия, всегда полагается на сервер для получения актуальных данных и результатов своих действий.

  
**Диаграмма классов**

1. **Общие классы**

Эти классы используются для передачи данных между клиентом и сервером и/или для представления логики работы приложения.

1.1. **User**

* Назначение: Представляет пользователя системы.
* **Поля**:
  + id: Long
  + username: String
  + password: String (в зашифрованном виде)
  + email: String
* **Методы**:
  + Геттеры и сеттеры.
  + Проверка валидности данных пользователя (например, email).

1.2. **Pet**

* **Назначение**: Представляет питомца, связанного с пользователем.
* **Поля**:
  + id: Long
  + name: String (фиксированное имя, без возможности изменения)
  + owner: User
  + hunger: int (уровень голода)
  + happiness: int (уровень счастья)
  + energy: int (уровень энергии)
  + health: int (уровень здоровья)
* **Методы**:
  + Геттеры и сеттеры.
  + Методы для обновления состояния питомца (например, feed(), play(), rest()).

2. **Серверная часть (Spring)**

Классы, необходимые для работы серверной части приложения.

2.1. **UserController**

* **Назначение:** Обработка запросов, связанных с пользователями (авторизация, регистрация).
* **Методы:**
  + register(UserDTO userDTO): ResponseEntity
  + login(UserDTO userDTO): ResponseEntity
  + getUserInfo(String token): ResponseEntity

2.2. **PetController**

* **Назначение:** Обработка запросов, связанных с питомцами.
* **Методы:**
  + getPet(String token): ResponseEntity<PetDTO>
  + updatePetState(String token, PetDTO petDTO): ResponseEntity

2.3. **UserService**

* **Назначение**: Бизнес-логика для управления пользователями.
* **Методы:**
  + register(User user): User
  + authenticate(String username, String password): User
  + findByUsername(String username): Optional<User>

2.4. **PetService**

* **Назначение**: Бизнес-логика для управления питомцами.
* **Методы**:
  + getPet(User user): Pet
  + updatePet(Pet pet): Pet
  + Методы для обновления состояния питомца (вызов действий кормления, игры и т.д.).

2.5. **UserRepository**

* **Назначение**: Доступ к данным о пользователях в БД.
* **Методы**:
  + findByUsername(String username): Optional<User>

2.6. **PetRepository**

* **Назначение**: Доступ к данным о питомцах в БД.
* **Методы**:
  + findByOwner(User user): Optional<Pet>

3. **Клиентская часть (JavaFX)**

Классы для реализации графического интерфейса пользователя и взаимодействия с сервером.

3.1. **LoginController**

* **Назначение**: Управляет экраном авторизации.
* **Методы**:
  + handleLogin()
  + handleRegister()

3.2. **RegisterController**

* **Назначение**: Управляет экраном регистрации.
* **Методы**:
  + handleRegister()

3.3. **MainController**

* **Назначение**: Управляет основным экраном игры (статус питомца и взаимодействие).
* **Методы**:
  + updatePetStatus()
  + handleFeed()
  + handlePlay()
  + handleRest()

3.4. **SettingsController**

* **Назначение**: Управляет экраном настроек пользователя (без возможности изменения имени питомца).
* **Методы**:
  + handleChangeSettings() (например, изменение пароля, email и других настроек пользователя)

4. **DTO (Data Transfer Object)**

Объекты для передачи данных между клиентом и сервером.

4.1. **UserDTO**

* **Поля**:
  + username: String
  + password: String
  + email: String (опционально для регистрации)

4.2. **PetDTO**

* **Поля**:
  + name: String (фиксированное значение, отправляется только при создании питомца)
  + hunger: int
  + happiness: int
  + energy: int
  + health: int

5. **Утилитарные классы**

5.1. **JWTUtil**

* **Назначение**: Генерация и проверка JWT-токенов для авторизации.
* **Методы**:
  + generateToken(User user): String
  + validateToken(String token): boolean

5.2. **ValidationUtil**

* **Назначение**: Проверка входных данных (логин, пароль, email и т.д.).
* **Методы**:
  + isValidEmail(String email): boolean