Санкт-Петербургский государственный университет

Кафедра Системного программирования

Группа 22.Б15-мм

Разработка серверной части раздела «Отловщики» веб-сервиса для помощи бездомным животным

Перевалов Ефим Сергеевич

Отчёт по производственной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н., Д. В. Луцив

Консультант:
инженер-разработчик, YADRO, И. Д. Шеремет

Санкт-Петербург
2025

Оглавление

Введение

Проблема бездомных животных является глобальной, характеризующейся как этическими, так и практическими сложностями. Общеизвестно, что эти животные, лишенные постоянного крова и опеки, подвержены многочисленным рискам, включая голод, болезни, травмы и жестокое обращение. Существующие механизмы помощи, как правило, фрагментированы и децентрализованы и опираются на индивидуальные действия граждан, локальные приюты и ветеринарные клиники. Такой подход характеризуется отсутствием единой платформы, обеспечивающей координацию усилий и оперативный обмен информацией между потенциальными помощниками и организациями, предоставляющими помощь. Это приводит к неоптимальному распределению ресурсов и задержке возможности содействия. Сложность во взаимодействии между гражданами, желающими помочь, и организациями, предоставляющими помощь, является значимой проблемой для успешной реализации гуманитарных инициатив.

Несмотря на общепризнанность проблемы бездомных животных и наличие отдельных структур, оказывающих помощь, существуют значительные сложности, связанные с эффективностью текущих подходов и их масштабируемостью. Недостаточно распространены оптимальные механизмы интеграции различных структур (волонтеры, приюты, ветеринарные клиники, службы отлова) и их взаимодействия для обеспечения оперативной и целенаправленной помощи. Отсутствуют систематизированные данные о количестве бездомных животных, их географическом распределении, видовом составе, и потребностях.

Создание и внедрение веб-сервиса, выступающего в роли агрегатора информации о бездомных животных и содействующих ресурсов, может значительно повысить результативность процесса оказания помощи и сократить время от момента обнаружения животного до его спасения. Целью данного проекта является разработка и тестирование веб-сервиса, который обеспечит централизованный доступ к информации о бездомных животных, приютах, ветеринарных клиниках, волонтерах

и службах отлова, способствующий более оперативной, координированной и эффективной помощи.

Для достижения цели проекта, необходимо провести комплексное изучение архитектуры и внутренней логики существующего программного продукта. Этот этап будет включать разбор кода, анализ API, а также структуры базы данных, чтобы определить оптимальный способ интеграции новых компонентов. Основываясь на этом анализе, предстоит разработать техническую спецификацию недостающих частей, включая описание структуры данных и логики работы.

1. Постановка задачи

Целью работы является разработка серверной части раздела «Отловщики». Для ее достижения необходимо выполнить следующие задачи:

1. Добавление структур, отвечающих за хранение, создание, обновление и получение информации об отловщике, и интерфейсов для описания логики работы модуля.
2. Расширение схемы базы данных.
3. Реализация взаимодействия бизнес-логики модуля с базой данных.
4. Добавление валидации входных данных.
5. Подготовка взаимодействия серверной части модуля с клиентской.
6. Тестирование.

2. Обзор

2.1. Golang

Компилируемый, статически типизированный язык программирования, разработанный в компании Google. Он характеризуется высокой производительностью и поддержкой параллелизма благодаря встроенным механизмам горутин и каналов. Язык отличает лаконичный синтаксис, нацеленный на повышение читаемости и упрощение сопровождения программного кода. Благодаря этим характеристикам, Golang представляет собой эффективный инструмент для разработки высоконагруженных веб-сервисов, API и микросервисных архитектур. [3]

2.2. PostgreSQL

Объектно-реляционная система управления базами данных (СУБД) поддерживающая широкий спектр типов данных, включая сложные типы, такие как массивы, JSON и геометрические данные, а также обладает продвинутыми функциями индексирования, транзакционности и управления параллельным доступом. Помимо этого, благодаря наличию специализированных драйверов и ORM-библиотек, интеграция с проектами на Golang проста и эффективна. [4]

2.3. Fiber

Минималистичный и высокопроизводительный веб-фреймворк для Golang. Предоставляет простой и мощный механизм маршрутизации для обработки запросов к различным URL-адресам. Дает удобные методы доступа к параметрам, заголовкам и телу запроса. Также есть функции для отправки ответов в различных форматах. [1]

2.4. Swagger

Инструмент для описания и документирования RESTful API. В Go-проектах Swagger помогает сделать API более понятным и удобным для

использования как для разработчиков, так и для сторонних интеграций. [5]

2.5. GORM

Это ORM библиотека для языка программирования Go, предоставляющая абстракцию над реляционными базами данных. Она обеспечивает разработчикам возможность манипулировать данными, используя структуры Go, вместо непосредственного написания SQL-запросов. [2]

3. Реализация

3.1. Добавление необходимых структур и интерфейсов

Для разграничения взаимодействий с отловщиком было созданно несколько core структур, которые упрощали и изолировали процесс взаимодействия для пользователя:

- `Seeker` — вся информация об отловщике;
- `UpdateSeeker` — информация, которая может быть обновлена;
- `GetAllSeekersParams` — предназначен для предоставления информации о нескольких отловщиках с учетом выбранной сортировки.

Также требовалось написать интерфейсы, разделяющие логику на сервисную и store части:

- `SeekersService` — отвечает за обработку запросов;
- `SeekersStore` — взаимодействует с базой данных.

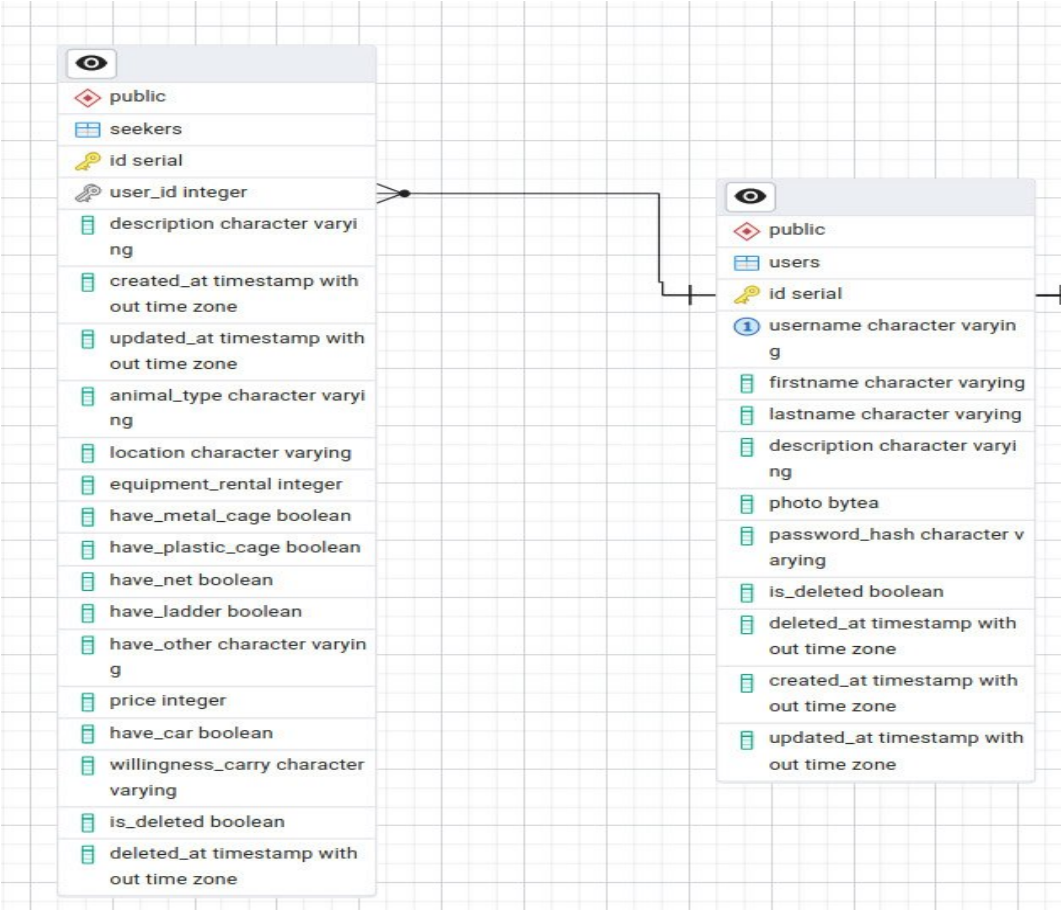
3.2. Добавление в базу данных

После того, как архитектура продумана и нужные структуры и интерфейсы написаны, необходимо дополнить существующую информацию в базе данных. Изначально для таблицы отловщиков уже хранился некоторый минимум, поэтому одной из задач данной работы было его расширение в соответствии с новыми требованиями:

```
ALTER TABLE IF EXISTS seekers
    ADD animal_type       VARCHAR   NOT NULL,
    ADD location          VARCHAR   NOT NULL,
    ADD equipment_rental  INTEGER   NOT NULL,
    ADD have_metal_cage   BOOLEAN   NOT NULL,
    ADD have_plastic_cage BOOLEAN   NOT NULL,
    ADD have_net          BOOLEAN   NOT NULL,
    ADD have_ladder       BOOLEAN   NOT NULL,
    ADD have_other        VARCHAR   NOT NULL,
    ADD price             INTEGER   NOT NULL,
    ADD have_car          BOOLEAN   NOT NULL,
    ADD willingness_carry VARCHAR   NOT NULL,
    ADD is_deleted        BOOLEAN   NOT NULL DEFAULT false,
    ADD deleted_at        TIMESTAMP;
```

```
ALTER TABLE IF EXISTS seekers
    DROP COLUMN IF EXISTS animal_type,
    DROP COLUMN IF EXISTS location,
    DROP COLUMN IF EXISTS equipment_rental
    DROP COLUMN IF EXISTS equipment
    DROP COLUMN IF EXISTS price,
    DROP COLUMN IF EXISTS have_car,
    DROP COLUMN IF EXISTS willingness_carry,
    DROP COLUMN IF EXISTS is_deleted,
    DROP COLUMN IF EXISTS deleted_at;
```

После этого новая сущность и связь выглядят следующим образом:



В таблицу добавлены основные поля, отвечающие за экипировку, животного, локацию, наличие машины и возможность назначения цены за услуги, и несколько вспомогательных, отвечающих за удаление и время выполнения операции.

3.3. Добавление взаимодействия с базой данных

На данном этапе необходимо написать логику сохранения и обновления информации в базе данных. Для это были реализованы следующие функции:

- `CreateSeeker` — сохранение нового отловщика в базу данных;
- `GetSeeker` — получение пользователя из базы данных;
- `UpdateSeeker` — обновление данных об отловщике в базе данных;

- `DeleteSeeker` — добавляет информацию для пользователя о том, что он больше не является отловщиком;
- `GetAllSeekers` — получение списка всех отловщиков по заданным параметрам.

3.4. Добавление сервисной части

Перед тем как передать данные в базу данных их необходимо провалидировать. Для этого и предназначена предобработка, после которой происходит передача на store часть:

- `CreateSeeker` — производит проверку, чтобы пользователь не пытался несколько раз отправить данные на добавление своего аккаунта к отловщикам;
- `GetSeeker` — добавляет обработку ситуации, когда отловщика не существует и когда он удален из базы данных;
- `UpdateSeeker` — при попытке обновить информацию проверяет полученные значения и производит обновления только по ним, а так же убеждается, что пользователь передал какие—то необходимые данные;
- `DeleteSeeker` — проверяет, существует ли отловщик, перед тем, как пытаться удалить его;
- `GetAllSeekers` — проверка валидности сортировки, которую запросил пользователь.

3.5. Добавление новых структур для запросов

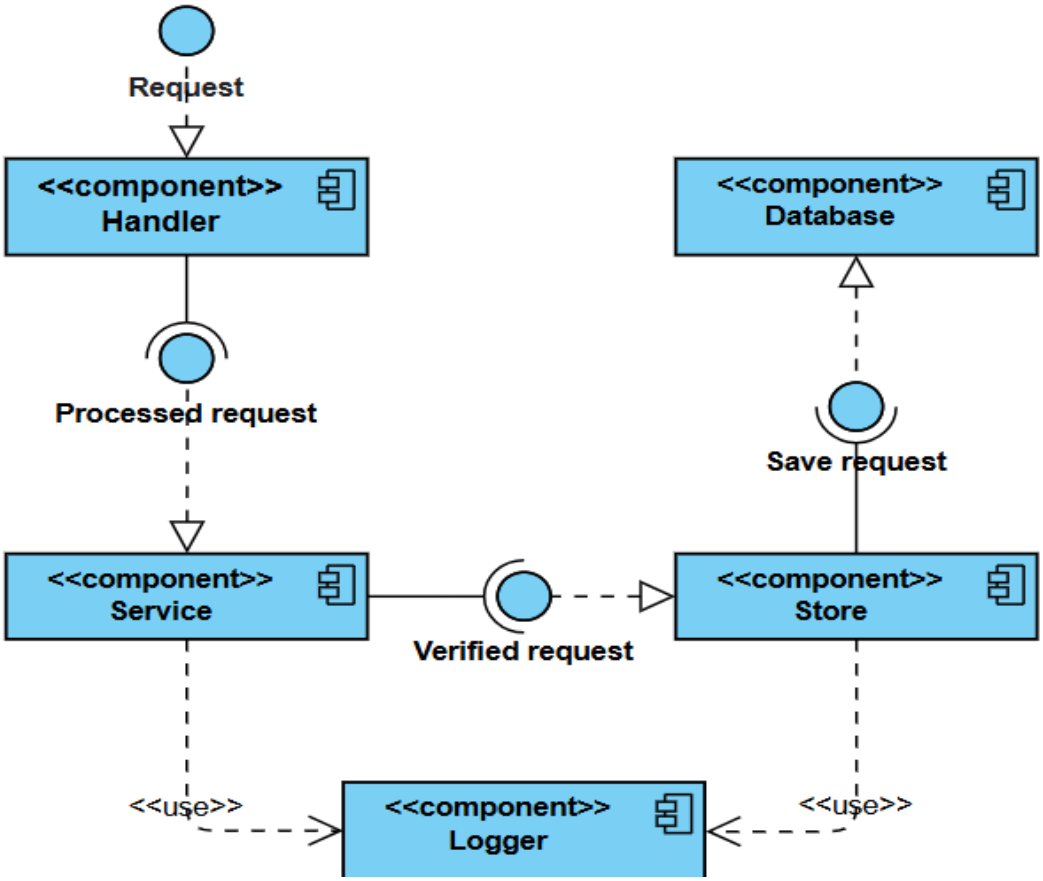
Следующим этапом следует добавить обработку запросов, которые будут поступать с клиентской части, и ответов на них. Для этого потребуются новые типы, которые и будут хранить полученную или отправленную информацию:

- `CreateSeeker` — промежуточный тип, отвечающий за полученные данные о создаваемом отловщике;
- `UpdateSeeker` — промежуточный тип для сохранения информации о том, какие данные отловщику потребовалось обновить;
- `GetAllSeekerParams` — промежуточный тип для информации о том, по каким полям нужно отсортировать либо отфильтровать отловщиков из базы данных, и в каком количестве их нужно передать;
- `ResponseSeeker` — тип, используемый для передачи обработанных данных об отловщике;
- `ResponseSeekers` — тип для хранения списка отловщиков, запрашиваемого с клиентской части.

3.6. Реализация обработки запросов

На данном этапе необходимо обработать данные полученные с клиентской части. Для этого парсится полученный запрос, сохраняется необходимая информация в один из промежуточных типов, а дальше преобразуется в core. Поэтому появилась необходимость реализовать конвертер, что и было осуществлено. Теперь же все типы можно было без особых усилий преобразовывать и передавать на service часть, а затем получать обработанные данные и возвращать пользователю, конвертируя их в промежуточный тип для ответов.

3.7. Диаграмма компонентов



Первым этапом происходит получение запроса от клиентской части, после чего начинается формирование запроса с учетом полученной информации (Handller). Когда все преобразования закончены, данные передаются на проверку (Service). На данном этапе выполняется валидация полученной информации, при возникновении ошибки она логируется и пользователю возвращается соответствующее сообщение. В ином случае данные передаются на составление запроса для базы данных (Store). При успехе в Database появляется необходимая информация, а пользователь получает сообщение об успешной операции. Если же сохранить не получилось, ошибка логируется и возвращается в формате ответа. Последним этапом пользователю передается информация об успехе либо об ошибке с указанием проблемы.

3.8. Тестирование

Для проверки работоспособности и отладки сервиса был использован инструмент Swagger, благодаря которому можно без особых усилий отправлять данные на серверную часть. Его добавление позволило не только автоматизировать процесс создания документации API, но и провести более эффективное и структурированное тестирование. Таким образом, были проверены на работоспособность все необходимые операции: создание, получение, обновление и удаление. Swagger-документация была сгенерирована автоматически с помощью [6].

Работа велась в репозитории [8], основной репозиторий [7] .

Заключение

Результат практики осеннего семестра:

1. Добавлены структуры, отвечающие за хранение, создание, обновление и получение информации об отловщике, и интерфейсов для описания логики работы модуля. Для этого в core создан соответствующий файл seeker, в котором и содержатся данные объявления.

2. Расширена схема база данных. Для этого в нее внесены все новые поля, указанные в требованиях.

3. Реализовано взаимодействие бизнес-логики модуля с базой данных. Для этого в store добавлен соответствующий файл, отвечающий за взаимодейтсвие с базой данных для каждой операции.

4. Добавлена валидация входных данных. В service реализована проверка входных данных перед началом работы с базой данных.

5. Подготовлено взаимодействие серверной части модуля с клиентской. Созданы новые промежуточные типы, отвечающие за сохранение информации для ответа или запроса. Реализована логика, отвечающая за парсинг данных, полученных с клиентской части и связывание запросов с соответсвующими функциями.

6. Протестированы все операции для отловщиков, а именно: удаление, создание, обновление, получение как одного, так и списка отловщиков.

Список литературы

[1] Fiber. — URL: https://github.com/gofiber/fiber (дата обращения: 2 января 2025 г.).

[2] GORM. — URL: https://github.com/go-gorm/gorm (дата обращения: 2 января 2025 г.).

[3] Golang. — URL: https://go.dev (дата обращения: 2 января 2025 г.).

[4] PostgreSQL. — URL: https://www.postgresql.org (дата обращения: 2 января 2025 г.).

[5] Swagger. — URL: https://en.wikipedia.org/wiki/Swagger_(software) (дата обращения: 2 января 2025 г.).

[6] swago/swag. — URL: https://github.com/swaggo/swag (дата обращения: 2 января 2025 г.).

[7] Основной репозиторий. — URL: https://github.com/kotopesp/sos-kotopes (дата обращения: 2 января 2025 г.).

[8] Рабочий репозиторий. — URL: https://github.com/kotopesp/sos-kotopes/tree/feature/seekers (дата обращения: 2 января 2025 г.).