Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
д-р техн. наук, профессор
_____ С.П. Сущенко
подпись
«03» июня 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

ВЕБ-ПРИЛОЖЕНИЕ ДЛЯ СОЗДАНИЯ ФОТОГАЛЕРЕЙ

по направлению подготовки 09.03.03 Прикладная информатика
направленность (профиль) «Прикладная информатика»

Черных Егор Михайлович

Руководитель ВКР
канд. техн. наук, доцент
_____ С.В. Аксёнов
подпись
«28» мая 2023 г.

Автор работы
студент группы № 931806
_____ Е.М. Черных
подпись
«28» мая 2023 г.

Томск – 2023

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Наименование учебного структурного подразделения

УТВЕРЖДАЮ
Руководитель ООП
д-р техн. наук, профессор
_____ С.П. Сущенко
подпись
«10» ноября 2022 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы бакалавра обучающемуся Черных Егору Михайловичу
Фамилия Имя Отчество обучающегося
по направлению подготовки 09.03.03 Прикладная информатика, направленность (профиль) «Прикладная информатика»

1 Тема выпускной квалификационной работы
«Веб-приложение для создания фотогалерей»

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:
а) в учебный офис / деканат – 28.05.23г. б) в ГЭК – 07.06.23г.

3 Исходные данные к работе:
Объект исследования – Веб-приложение для создания фотогалерей
Предмет исследования – Разработка веб-приложения для создания фотогалерей
Цель исследования – Реализация веб-приложения для создания фотогалерей

Задачи:
Провести анализ существующих решений. Сформировать требования к веб-приложению. Спроектировать веб-приложение. Разработать веб-приложение

Методы исследования:
Теоретическое исследование и практическая реализация

Организация или отрасль, по тематике которой выполняется работа, –
Веб-разработка

4 Краткое содержание работы
Проведен анализ существующих решений, исходя из которых сформулированы требования к веб-приложению. Выбран архитектурный паттерн, на основе которого спроектировано и реализовано веб-приложение.

Руководитель выпускной квалификационной работы

доцент кафедры		С. В. Аксёнов
должность, место работы	*подпись*	*И.О. Фамилия*

Задание принял к исполнению

Студент группы 931806		Е. М. Черных
должность, место работы	*подпись*	*И.О. Фамилия*

АННОТАЦИЯ

Выпускная квалификационная работа содержит 51 страницу, 44 рисунка, 7 источников.

ВЕБ-РАЗРАБОТКА, ВЕБ-ПРИЛОЖЕНИЕ, GO, POSTGRESQL, BOOTSTRAP, GORM.

Целью данной работы является разработка веб-приложения, позволяющего пользователям создавать галереи с фотографиями.

В работе проведен аналог существующих решений в области сайтов с функционалом для создания фотогалерей. На основе анализа были выявлены требования к веб приложению, такие как безопасная авторизация пользователя, возможность создавать и редактировать фотогалереи.

Результат работы – разработано веб-приложение с лаконичным интерфейсом, с помощью которого пользователи могут создавать и редактировать фотогалереи.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Благодаря повсеместному распространению мобильных телефонов, сейчас практически у каждого человека есть возможность создавать и делиться собственными фотографиями. Желание в публикации личных фотографий могут удовлетворить социальные сети. Однако популярные социальные сети специально спроектированы для потокового потребления информации, и часто они обладают обширным функционалом, не сконцентрированным только на фотографии. Если человек занимается фотографией профессионально или просто использует ее как форму самовыражения, у него или нее может возникнуть потребность в инструменте, функционал которого специально создан для такой формы медиа как фотография.

Целью данной работы является проектирование и разработка веб-приложения, позволяющего пользователям создавать галереи с фотографиями. Данное веб-приложение будет иметь необходимый функционал для создания пользовательских фотогалерей. Оно может быть полезно как для профессиональных фотографов, которым нужно показать свое портфолио клиентам, так и для людей, увлекающихся фотографией, и для которых функционал социальных сетей излишен. Понимание требований пользователей, реализация надежных функций и использование современных веб-технологий позволит создать доступное и удобное решение, которое упростит процесс организации и обмена коллекциями фотографий.

При создании данного веб-приложения нужно решить следующие задачи:

1. Провести анализ существующих решений;
2. Провести анализ технических средств и определить стек технологий
3. Описать функциональные и нефункциональные требования;

4. Определить архитектуру приложения, компоненты и их взаимодействия;
5. Спроектировать базу данных
6. Разработать веб-приложение

1 Анализ предметной области

1.1 Анализ существующих решений

1.1.1 Flickr

Flickr ориентирован в первую очередь на обмен фотографиями, предоставляющее фотографам и художникам платформу для демонстрации своих работ, взаимодействия с сообществом и поиска изображений. Одной из его отличительных особенностей является большое и активное сообщество фотографов, позволяющее пользователям общаться с и получать комментарии к своим фотографиям. Платформа предлагает обширное пространство для хранения данных, предоставляя пользователям 1 000 ГБ бесплатного хранилища, что позволяет фотографам хранить большое количество изображений высокого разрешения. Flickr поддерживает систему лицензирования Creative Commons, предоставляя пользователям контроль над правами на использование их фотографий, что может повысить узнаваемость и одновременно защитить их работу.

Flickr имеет заметные недостатки. Пользовательский интерфейс может быть сложным и менее интуитивным по сравнению с другими платформами обмена фотографиями. Навигация по различным функциям и опциям требует времени. Еще одним недостатком Flickr является падение его популярности с годами. Снижение активности пользователей может ограничить возможности для знакомств в сообществе Flickr. Также, хотя Flickr предлагает бесплатный тарифный план, некоторые расширенные функции и возможности требуют платной подписки. Пользователям, желающим получить дополнительные преимущества, такие как просмотр без рекламы, подробная статистика и улучшенное качество изображений, необходимо перейти на премиум-план.

1.1.2 SmugMug

SmugMug – веб-приложение для фотографов с возможностью создания и управления онлайн-портфолио и галереями. SmugMug имеет обширный набор функций:возможность настраивать свои галереи с помощью различных тем, макетов и вариантов отображения. Это позволяет фотографам демонстрировать свои работы в визуально привлекательной и персонализированной манере. Платформа также предоставляет расширенные инструменты организации, позволяющие пользователям эффективно распределять фотографии по категориям и управлять ими на основе тем, событий или любых других критериев. SmugMug уделяет особое внимание конфиденциальности и безопасности. Приложение обеспечивает надежные настройки конфиденциальности, позволяя пользователям контролировать, кто может просматривать их галереи. SmugMug обеспечивает безопасное хранение фотографий пользователей.

Однако у SmugMug есть несколько минусов. Платформа может оказаться непосильной для новичков из-за сложного пользовательского интерфейса. Навигация по различным функциям и возможностям может потребовать времени и объяснений, что делает ее менее интуитивной для пользователей. Также, SmugMug может быть относительно дорогим по сравнению с другими подобными платформами, особенно для фотографов, которым требуются расширенные функции или дополнительное пространство для хранения. Ценовая политика может отпугнуть фотографов от полного использования возможностей платформы.

1.1.3 Zenfolio

Zenfolio - это веб-приложение для фотографов, предлагающее комплексную платформу для создания и управления онлайн-портфолио. Одним из его достоинств является удобный интерфейс, который облегчает настройку портфолио. Zenfolio предоставляет ряд настраиваемых шаблонов и

вариантов дизайна, что позволяет пользователям создавать визуально привлекательные галереи, соответствующие их личному стилю и брендингу. Приложение предлагает надежные инструменты организации, позволяющие фотографам эффективно классифицировать и демонстрировать свои работы на основе тем, событий или других критериев по своему выбору.

Недостатком является ограниченный ассортимент доступных шаблонов и вариантов дизайна. Портфолио могут выглядеть похожими. Еще одним недостатком является ценовая структура Zenfolio. Хотя платформа предлагает различные уровни подписки, некоторые из более продвинутых функций и опций настройки доступны только в более дорогих планах.

1.1.4 500px

500px - платформа, являющаяся сообществом фотографов и платформой для демонстрации их работ. Платформа поощряет социальное взаимодействие с помощью таких функций, как комментарии, лайки и подписки на других фотографов. 500px имеет визуально привлекательный интерфейс и отображает снимки в высоком разрешении. Кроме того, 500px предлагает курируемые коллекции и торговые площадки, где демонстрируются исключительные фотографии, предоставляя фотографам возможность получить известность и потенциально продать свои работы.

Одним из недостатков является то, что платформа использует алгоритмическую систему курации на основе популярности. Эта система может оценивать фотографии, основываясь наа популярности фотографа, а не на каких-либо других качествах самой фотографии.

1.2 Определение требований к системе

1.2.1 Определение функциональных требований к системе

Выполнив анализ аналогов разрабатываемого приложения, можно определить следующие функциональные требования приложения:

- регистрация и авторизация с помощью почты и пароля
- создание фотогалерей
- загрузка изображений в фотогалереи
- удаление изображений из фотогалерей
- редактирование и удаление фотогалерей
- просмотр фотогалерей

1.2.2 Определение нефункциональных требований к системе

Нефункциональные требования данного приложения в основном связаны с защитой данных пользователя, то есть его пароля и cookie-файлов, а также с защитой от различных уязвимостей веб-приложения:

- В базе данных хранится хеш пароля
- Установлено ограничение на длину пароля
- Защита от внедрения SQL-кода (SQL-injection) и JS-кода (Cross-site scripting)
- Защита от фальсификации cookie-файлов

1.3 Выбор инструментов разработки

1.3.1 Серверная часть приложения

Для разработки серверной части приложения был выбран язык Go (Golang), так как стандартная библиотека данного языка позволяет создавать веб-приложения [1].

Go – это статически типизированный компилируемый высокоуровневый язык программирования. Синтаксически он похож на C, но имеет защиту памяти, сборку мусора, структурную типизацию и параллелизм.

В отличие от интерпретируемых языков JavaScript и Python, которые часто используются для написания веб-приложений, Go является компилируемым языком. Официальный компилятор Golang поддерживает операционные системы, основанные на UNIX, включая macOS, а также Linux и Windows. Строгая статическая типизация языка позволяет объявлять тип данных переменной при ее создании. Тип данных переменной не может измениться. Неиспользуемые переменные рассматриваются как ошибка компиляции. Явное указание зависимостей позволяет легко собирать код из различных компонентов, что упрощает разработку больших проектов. Сборщик мусора сканирует код и находит объекты, которые замедляют его работу, после чего удаляет их. "Сборщик мусора" играет важную роль в обеспечении высокой производительности программы и эффективного использования ресурсов. В некоторых языках общего назначения отсутствует встроенный "сборщик мусора", и память должна очищаться вручную, как, например, в C++.

1.3.2 СУБД

Приложение будет хранить информацию о пользователях и галереях, и эти данные будут связаны между собой, поэтому нам понадобится реляционная база данных.

PostgreSQL [2] – это мощная реляционная система управления базами данных с открытым исходным кодом, известная своей надежностью, расширяемостью и соответствием стандартам SQL. Она предлагает расширенные возможности, всестороннюю поддержку свойств ACID, широкий выбор типов данных и надежные меры безопасности. PostgreSQL очень универсальна и широко используется в различных приложениях, от небольших проектов до крупных корпоративных систем.

1.3.3 ORM

Для связи базы данных и серверной части веб-приложения понадобится объектно-реляционное отображение (ORM). ORM – технология программирования, суть которой заключается в создании «виртуальной объектной базы данных». Системы ORM обеспечивают способ отображения данных между объектно-ориентированной моделью, используемой в коде приложения, и реляционной моделью, используемой в базах данных. Основная цель ORM – упростить и оптимизировать процесс взаимодействия с базами данных путем абстрагирования от низкоуровневых деталей операций с базами данных. С помощью ORM можно работать с объектами баз данных как с объектами на выбранном языке программирования, а ORM выполняет преобразование данных между объектно-ориентированной и реляционной моделями.

Для языка Go существует библиотека GORM [3], обеспечивающая удобный способ взаимодействия с базами данных путем отображения структур Go на таблицы базы данных. У GORM имеются функции для автоматического создания и миграции таблиц, построения запросов,

проверку данных и т. д, также есть поддержка отношений между таблицами: "один к одному", "один ко многим" и "многие ко многим".

1.3.4 Клиентская часть веб-приложения

Для простоты написания клиентской части выбран фреймворк Bootstrap [4], который предоставляет компоненты CSS и JavaScript, которые можно легко настроить.

2. Проектирование

2.1 Диаграмма вариантов использования

Диаграмма вариантов использования (Рисунок 1) представляет собой графическое представление акторов и сценариев использования (или прецедентов) с использованием унифицированного языка моделирования UML (Unified Modeling Language). Она позволяет определить взаимодействие пользователя с системой, описывая, какие действия может выполнять актор, но не указывая, каким образом они выполняются.

Рисунок 1 - Диаграмма вариантов использования

2.2 Model-View-Controller

Модель-Представление-Контроллер [5] (Рисунок 2), или MVC, - это паттерн, используемый для организации и структурирования кода. Написанный код может классифицироваться как модель, представление или контроллер.

Представления отвечают за отображение данных. Учитывая конкретную страницу, которую мы хотим отобразить, и данные для этой страницы, наше представление отвечает за генерацию правильного вывода. В случае нашего приложения отображаемыми данными будет HTML-код. Представления должны содержать как можно меньше логики, их задача в отображении данных

Контроллеры действуют как посредники между моделями и представлениями. Контроллеры направляют входящие данные в соответствующие модели, представления и другие пакеты, которые могут быть использованы для выполнения требуемой работы. Как и представления, контроллер не должен содержать слишком много бизнес-логики. Вместо этого они должны в основном просто анализировать данные и передавать их другим функциям, типам и т.д. для обработки.

Основной задачей моделей является взаимодействие с данными приложения. В случае нашего приложения, модели отвечают за взаимодействие с базой данных, но модели также могут взаимодействовать с данными, поступающими из других служб или API. Кроме того, модели отвечают за проверку или нормализацию данных.

Поток данных в паттерне MVC обычно происходит следующим образом: пользователь взаимодействует с представлением, которое уведомляет контроллер о действиях пользователя. Контроллер обновляет модель на основе полученных данных, а затем обновляет представление измененными данными. Представление представляет обновленную информацию пользователю, и цикл продолжается.

Разделяя проблемы управления данными, пользовательского интерфейса и логики приложения, паттерн MVC способствует модульности, сопровождаемости и масштабируемости при разработке программного обеспечения. Он позволяет упростить повторное использование кода, тестирование и гибкость при внесении изменений в различные компоненты без ущерба для остальных.

Паттерн MVC разделяет управление данными, пользовательский интерфейс и логику приложения, обеспечивая модульность, сопровождаемость и масштабируемость при разработке.

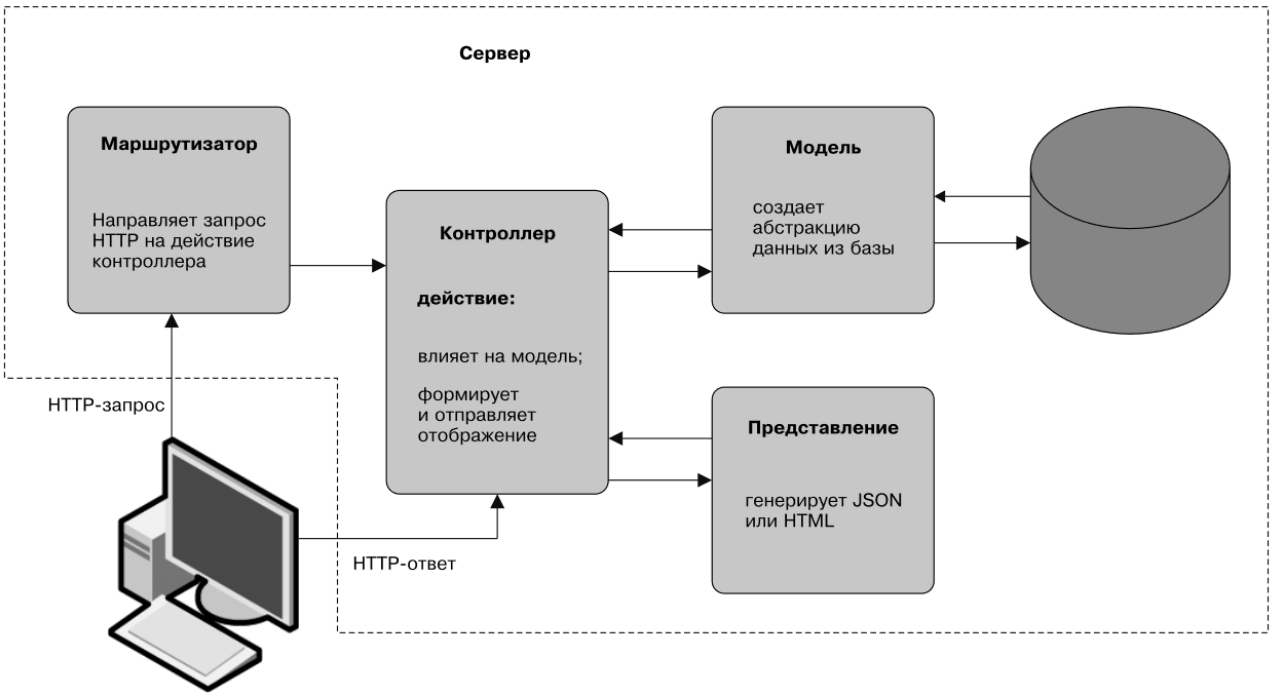


Рисунок 2 – Графическое представление MVC

2.3 ER-модель базы данных

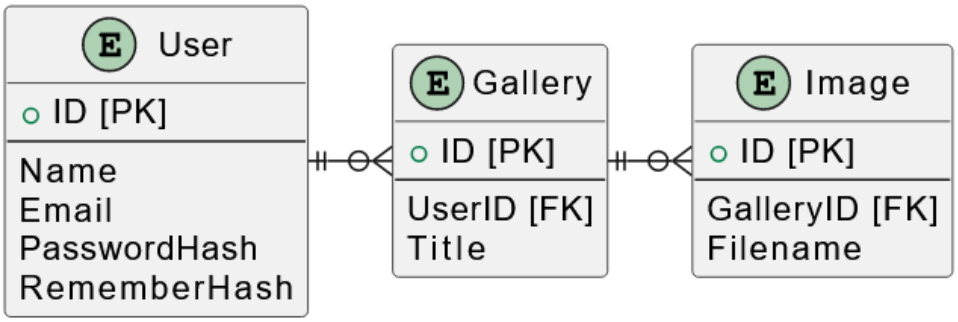


Рисунок 3 – ER-модель базы данных приложения

ER-модель (Рисунок 3) имеет три сущности: Пользователь, Галерея и Изображение.

- Пользователь: Представляет учетную запись пользователя в приложении. Он имеет такие атрибуты, как ID (первичный ключ), Name, Email, PasswordHash и RememberHash. ID является уникальным идентификатором для каждого пользователя, а Email используется для аутентификации. PasswordHash и RememberHash хранят хешированные значения пароля и маркера пользователя соответственно.
- Галерея: Представляет собой коллекцию изображений, принадлежащих пользователю. Она имеет атрибуты, включая ID (первичный ключ), UserID (внешний ключ) и Title. UserID означает идентификатор пользователя, которому принадлежит галерея. Каждая галерея имеет название, которое описывает ее содержимое.
- Изображение: Представляет собой изображение, связанное с галереей. Оно имеет такие атрибуты, как ID (первичный ключ), GalleryID (внешний ключ) и Filename. GalleryID указывает на идентификатор галереи, к которой принадлежит изображение. Каждое изображение связано с определенной галереей и имеет имя файла.

Один пользователь может иметь несколько галерей. Это указывает на связь "один ко многим", когда пользователь может владеть несколькими галереями, но каждая галерея связана только с одним пользователем. Одна галерея может иметь несколько изображений. Это также отношение "один-ко-многим", то есть галерея может содержать несколько изображений, но каждое изображение принадлежит только одной галерее.

2.3 Аутентификация пользователя

Аутентификация - это одна из важных частей разрабатываемого нами веб-приложения. Плохо выполненная система аутентификации может не только подвергнуть риску данные пользователей в нашем приложении , но и привести к утечке данных пользователей на сторонних ресурсах, так как часто пользователи используют один и тот же пароль на различных ресурсах [6]. Таким образом, ни в коем случае нельзя хранить незащищенный пароль в базе данных.

Кроме того, мы не можем использовать шифрование пароля (Рисунок 4), так как зашифрованные данные могут быть расшифрованы, если известен ключ шифрования. Более безопасный способ хранения пароля — преобразование его в данные, которые нельзя преобразовать обратно в исходный пароль. Такой способ называется хешированием (Рисунок 5). Хеширование происходит с помощью хеш-функции. Хеш-функция – это функция, которую можно использовать для получения данных произвольного размера и преобразования их в данные фиксированного размера. Применение хеш-функции называют "хешированием", а значения, возвращаемые хеш-функцией, называют хеш-значениями или хешами Если хранить только хешированную версию паролей, то даже в случае взлома базы данных злоумышленникам будет гораздо сложнее получить настоящие пароли. Им придется использовать такие методы, как перебор или предварительно вычисленные таблицы поиска (радужные таблицы), чтобы попытаться угадать оригинальные пароли на основе хранящихся хешей. При

аутентификации пользователя, мы будем хешировать введенный пароль, после чего сравнивать введенный хешированный пароль с хешем, который хранится в нашей базе данных.

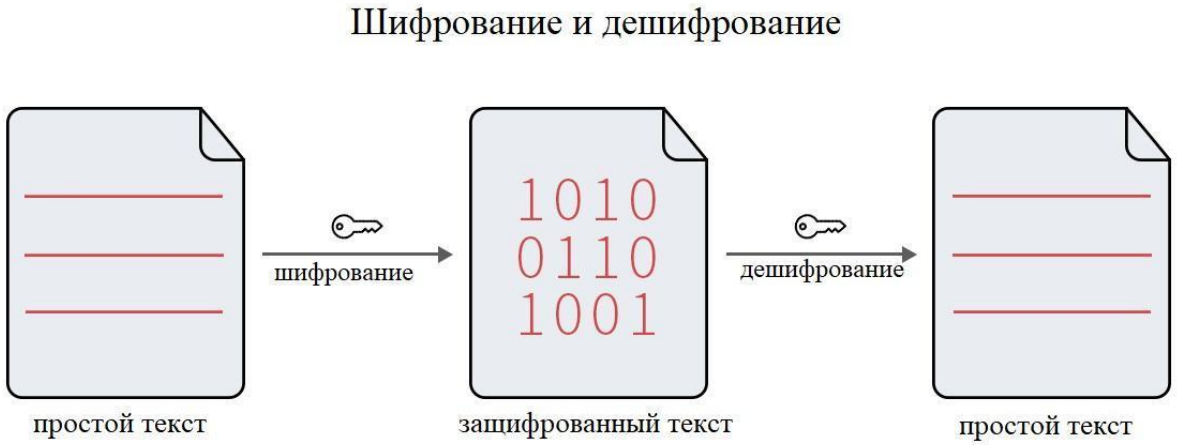


Рисунок 4 – Шифрование и дешифрование

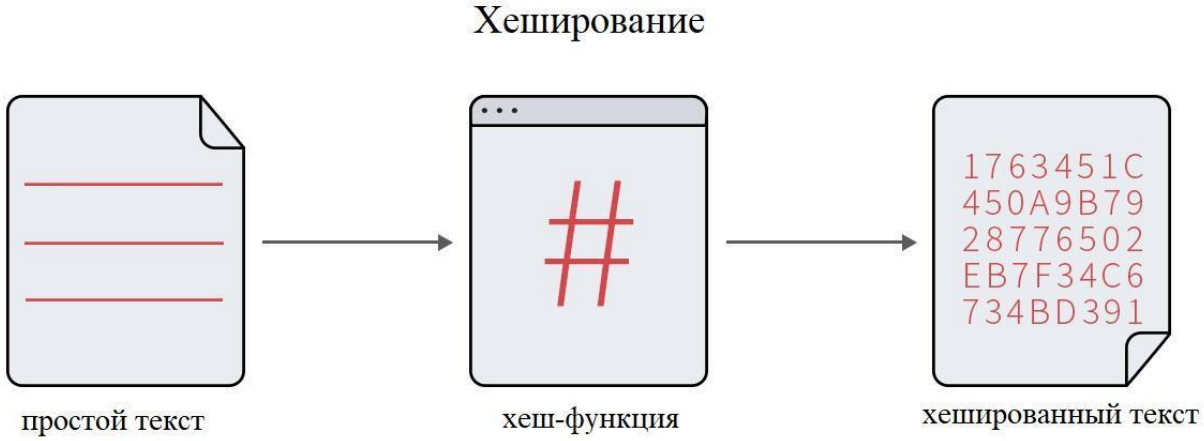


Рисунок 5 – Хеширование

Однако только хешированние не гарантирует защиту пароля. Существует атака, при которой злоумышленник генерирует большой набор возможных паролей, хеширует их и сохраняет полученные пары хеш-значение в таблице. Такая таблица называется “радужной таблицей”.

Чтобы взломать хешированный пароль с помощью радужной таблицы, злоумышленник просматривает хеш-значение в таблице и находит соответствующее значение простого текста. Для защиты от таких атак обычно используется техника, называемая солением (salting). Соление заключается в добавлении уникального случайного значения (соли) к каждому паролю перед хешированием. Это означает, что даже если у двух пользователей одинаковые пароли, результирующий хеш будет отличаться благодаря уникальным солям. Нужно отметить, что соли не являются приватными данными. Злоумышленник может получить доступ к соли пользователя, но поскольку у каждого пользователя должна быть своя соль, злоумышленнику все равно придется генерировать новую радужную таблицу для каждого пользователя, что делает этот способ атаки непрактичным.

3 Разработка

Для разработки веб-приложения был использован редактор исходного кода Visual Studio Code (VS Code), поскольку он является бесплатным, не требует больших ресурсов, поддерживает множество языков программирования, обеспечивает автодополнение, подсветку синтаксиса, отладку программ и имеет возможность установки плагинов и других расширений.

Файловая структура приложения представлена на Рисунке 6:

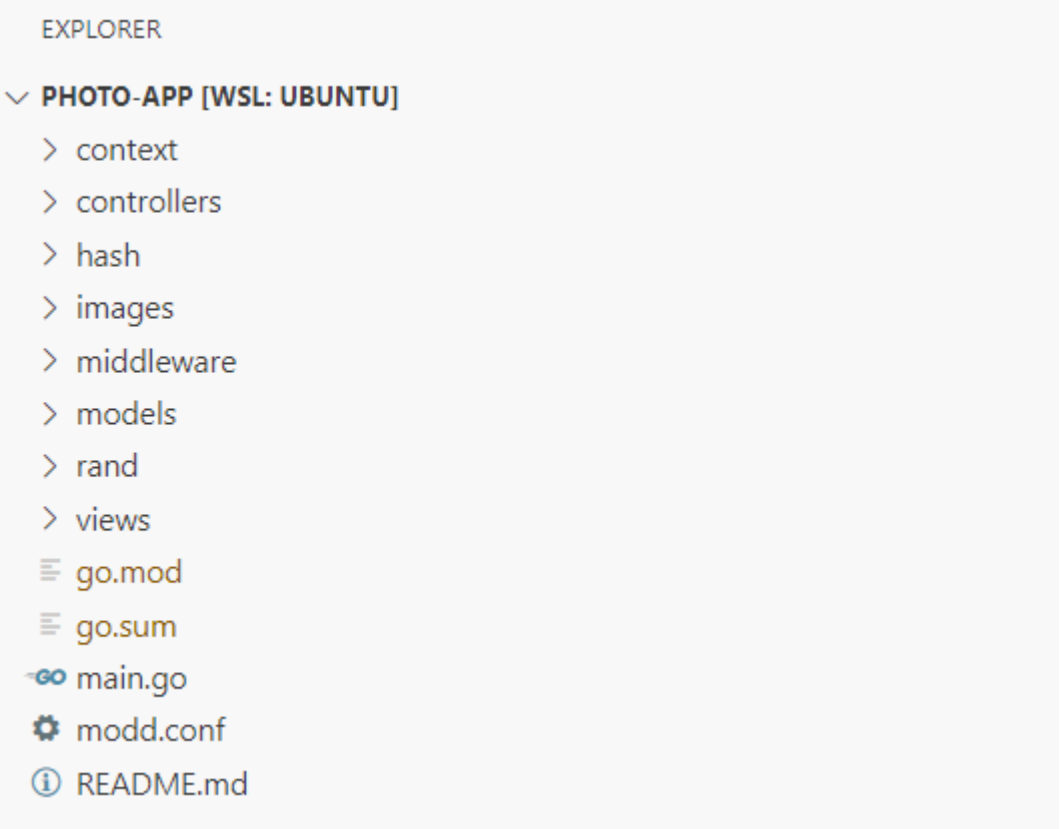


Рисунок 6 – Файловая структура проекта

3.1 Уровень контроллеров

3.1.1 Контроллер Users

Рисунок 7 – Контроллер User

Создадим контроллер Users (Рисунок 7), который будет содержать все функции обработчиков для страниц, взаимодействующих с пользователем.

NewUsers (Рисунок 8) создает новый экземпляр контроллера Users, который отвечает за обработку операций, связанных с пользователями в веб-приложении.

Он принимает models.UserService в качестве параметра для взаимодействия с данными пользователя и возвращает указатель на структуру Users.

```
func NewUsers(us models.UserService) *Users {
  return &Users{
    NewView: views.NewView("bootstrap", "users/new"),
    LoginView: views.NewView("bootstrap", "users/login"),
    us: us,
  }
}
```

Рисунок 8 – Функция NewUsers

Структура Users (Рисунок 9) представляет контроллер пользователя. Она содержит представления (Views) для создания новых пользователей и входа в систему, а также ссылку на models.UserService.

```
type Users struct {
    NewView *views.View
    LoginView *views.View
    us models.UserService
}
```

Рисунок 9 – Структура Users

Далее следует функция New, функция-обработчик для маршрута GET /signup:

```
func (u *Users) New(w http.ResponseWriter, r *http.Request) {
    u.NewView.Render(w, r, nil)
}
```

Она отображает представление регистрации нового пользователя, используя шаблон NewView.

Далее опишем структуру SignupForm (Рисунок 10), которая представляет структуру данных для формы регистрации пользователя. В ней есть три поля: Email, Пароль и Имя. Каждое поле аннотировано с помощью schema:"<name>", где <name> обозначает имя соответствующего поля ввода в HTML-форме. Эта аннотация обычно используется для сопоставления данных формы с полями структуры. Используя структуру SignupForm, код обеспечивает удобный способ обработки и проверки пользовательского ввода из формы регистрации в структурированной манере.

```
type SignupForm struct {
    Email string `schema:"email"`
    Password string `schema:"password"`
    Name string `schema:"name"`
}
```

Рисунок 10 – Структура SignupForm

Теперь опишем метод Create (Рисунок 11), которая выполняется, когда HTTP POST запрос сделан к маршруту "/signup". Эта функция отвечает за обработку данных формы регистрации пользователя и создание новой учетной записи пользователя.

Если парсинг формы прошел успешно, то создается новая структура models.User с использованием данных из формы. Для создания пользователя в базе данных вызывается метод Create экземпляра models.UserService. Если пользователь успешно создан, вызывается метод signIn для аутентификации вновь созданного пользователя. Если в процессе регистрации возникла ошибка, ответ перенаправляется на маршрут "/login". Если все прошло успешно, ответ перенаправляется на маршрут "/galleries". В целом, этот код обрабатывает отправку формы регистрации пользователя, обрабатывает данные формы, создает нового пользователя, регистрирует его и перенаправляет его на соответствующие страницы в зависимости от результатов этих операций.

```
func (u *Users) Create(w http.ResponseWriter, r *http.Request) {
  var vd views.Data
  var form SignupForm
  if err := parseForm(r, &form); err != nil {
    log.Println(err)
    vd.SetAlert(err)
    u.NewView.Render(w, r, vd)
  }
  fmt.Println(form)
  user := models.User{
    Name: form.Name,
    Email: form.Email,
    Password: form.Password,
  }
  if err := u.us.Create(&user); err != nil {
    vd.SetAlert(err)
    u.NewView.Render(w, r, vd)
    return
  }
  err := u.signIn(w, &user)
  if err != nil {
    http.Redirect(w, r, "/login", http.StatusFound)
    return
  }
  http.Redirect(w, r, "/galleries", http.StatusFound)
}
```

Рисунок 11 – Метод Create

Создадим структуру, похожую на SignupForm – LoginForm, представляющую структуру данных для формы входа пользователя в систему. Она содержит два поля: Email и Пароль:

```
type LoginForm struct {
   Email string `schema:"email"`
   Password string `schema:"password"`
}
```

И создадим метод-обработчик Login (Рисунок 12), выполняющуюся при HTTP POST запросе по маршруту "/login". Она обрабатывает отправку формы входа пользователя и аутентифицирует его. Код вызывает parseForm(r, &form), чтобы заполнить структуру формы данными из данных формы запроса. Если парсинг формы прошел успешно, код вызывает функцию u.us.Authenticate для аутентификации пользователя на основе предоставленных электронной почты и пароля. Функция возвращает объект пользователя и ошибку. Если при аутентификации возникла ошибка, код проверяет тип ошибки и обрабатывает ее. Если аутентификация прошла успешно, код вызывает u.signIn для входа пользователя в систему. Если в процессе входа в систему произошла ошибка, код возвращает внутреннюю ошибку сервера (HTTP 500) с сообщением об ошибке. И в конце, если вход в систему прошел успешно, код перенаправляет пользователя на страницу галереи ("/galleries") с помощью http.Redirect. В итоге данный код обрабатывает отправку формы входа пользователя, аутентифицирует пользователя и перенаправляет его на соответствующие страницы в зависимости от результатов процесса аутентификации. Он также обрабатывает случаи ошибок, выводя соответствующие сообщения об ошибках или возвращая внутренние ошибки сервера.

```
func (u *Users) Login(w http.ResponseWriter, r *http.Request) {
  form := LoginForm{}
  if err := parseForm(r, &form); err != nil {
    panic(err)
  }

  user, err := u.us.Authenticate(form.Email, form.Password)
  if err != nil {
    switch err {
    case models.ErrNotFound:
      fmt.Fprintln(w, "Invalid email address")
    case models.ErrPasswordIncorrect:
      fmt.Fprintln(w, "Invalid password provided")
    default:
      http.Error(w, err.Error(), http.StatusInternalServerError)
    }
  }
  err = u.signIn(w, user)
  if err != nil {
    http.Error(w, err.Error(), http.StatusInternalServerError)
    return
  }
  http.Redirect(w, r, "/galleries", http.StatusFound)
  // fmt.Fprintln(w, user)
}
```

Рисунок 12 – Метод Login

Напишем метод signIn (Рисунок 13). Он используется для идентификации данного пользователя путем записи токена в cookie.

Если поле Remember объекта user пустое, это означает, что у пользователя не установлен токен запоминания. В этом случае код генерирует новый токен с помощью rand.RememberToken() и присваивает его полю Remember пользователя. Затем код вызывает u.us.Update(user) для обновления информации о пользователе, токен сохраняется в базе данных. Далее создаем куки с именем "remember_token", в качестве значения которого устанавливается токен пользователя. Cookie помечен как HttpOnly, то есть доступ к нему и его изменение может осуществляться только сервером и недоступно для JavaScript на стороне клиента. Далее устанавливаем куки в HTTP-ответ с помощью http.SetCookie(w, &cookie).

```
func (u *Users)signIn(w http.ResponseWriter, user *models.User) error {
  if user.Remember == "" {
    token, err := rand.RememberToken()
    if err != nil {
      return err
    }
    user.Remember = token
    err = u.us.Update(user)
    if err != nil {
      return nil
    }
  }
  cookie := http.Cookie{
    Name: "remember_token",
    Value: user.Remember,
    HttpOnly: true,
  }
  http.SetCookie(w, &cookie)
  return nil
}
```

Рисунок 13 – Метод signIn

Напишем еще один метод-обработчик Logout (Рисунок 14) для маршрута /logout с HTTP-методом POST. Она обрабатывает процесс выхода пользователя из системы, аннулируя куки сессии пользователя (remember_token). Logout создает HTTP-куки с "remember_token" с пустым значением и устанавливает время истечения срока действия куки на текущее время, с помощью чего мы удаляем куки.

```
func (u *Users) Logout(w http.ResponseWriter, r *http.Request) {
  cookie := http.Cookie{
    Name:     "remember_token",
    Value:    "",
    Expires:  time.Now(),
    HttpOnly: true,
  }
  http.SetCookie(w, &cookie)

  user := context.User(r.Context())
  token, _ := rand.RememberToken()
  user.Remember = token
  u.us.Update(user)
  http.Redirect(w, r, "/", http.StatusFound)
}
```

Рисунок 14 – Метод Logout

3.1.2 Контроллер Galleries

Создадим контроллер Galleries (Рисунок 15).

Рисунок 15 – Контроллер Galleries

Структура Galleries(Рисунок 16) представляет контроллер для обработки операций, связанных с фотогалереями. Имеет следующие поля:

- New: Представляет объект представления для создания новой галереи
- ShowView: Представляет объект представления для отображения конкретной галереи
- EditView: Представляет объект представления для редактирования галереи
- IndexView: Представляет объект представления для отображения списка галерей.
- gs: Служба галереи, которая отвечает за обработку операций, связанных с галереями. Имеет тип models.GalleryService
- is: Сервис изображений, который отвечает за обработку операций, связанных с изображениями. Имеет тип models.ImageService

- r: Представляет объект router из пакета gorilla/mux. Он используется для определения и обработки маршрутов и имеет тип *mux.Router.

Определив структуру Galleries с данными полями, мы обеспечиваем структурированный способ организации необходимых представлений, сервисов и объектов маршрутизатора, необходимых для обработки операций, связанных с галереями.

```
type Galleries struct {
  New *views.View
  ShowView *views.View
  EditView *views.View
  IndexView *views.View
  gs models.GalleryService
  is models.ImageService
  r *mux.Router
}
```

Рисунок 16 – Структура Galleries

Напишем функцию NewGalleries (Рисунок 17), которая инициализирует и возвращает указатель на структуру Galleries. Эта функция принимает три параметра: gs типа models.GalleryService, предоставляющий службу галереи, которая отвечает за обработку операций, связанных с галереями; is типа models.ImageService, предоставляющий сервис изображений, который отвечает за обработку операций, связанных с изображениями; r типа *mux.Router, предоставляющий объект router из пакета gorilla/mux, который используется для определения и обработки маршрутов. Функция NewGalleries создает новый экземпляр структуры Galleries и инициализирует его поля.

```
func NewGalleries(gs models.GalleryService, is models.ImageService, r *mux.Router) *Galleries {
  return &Galleries{
    New: views.NewView("bootstrap", "galleries/new"),
    ShowView: views.NewView("bootstrap", "galleries/show"),
    EditView: views.NewView("bootstrap", "galleries/edit"),
    IndexView: views.NewView("bootstrap", "galleries/index"),
    gs: gs,
    is: is,
    r: r,
  }
}
```

Рисунок 17 – Функция NewGalleries

Напишем метод-обработчик Index (Рисунок 18) для маршрута GET /galleries. В общем, этот код обрабатывает маршрут GET /galleries, извлекает галереи, принадлежащие аутентифицированному пользователю, и отображает IndexView с извлеченными галереями. Если в процессе возникают какие-либо ошибки, возвращается соответствующий ответ об ошибке.

```
func (g *Galleries) Index(w http.ResponseWriter, r *http.Request) {
  user := context.User(r.Context())
  galleries, err := g.gs.ByUserID(user.ID)
  if err != nil {
    log.Println(err)
    http.Error(w, "Что-то пошло не так.", http.StatusInternalServerError)
    return
  }
  var vd views.Data
  vd.Yield = galleries
  g.IndexView.Render(w, r, vd)
}
```

Рисунок 18 – Метод Index

Далее напишем метод-обработчик Show (Рисунок 19) для маршрута GET /galleries/:id. Он извлекает и отображает конкретную галерею на основе предоставленного ID.

```
func (g *Galleries) Show(w http.ResponseWriter, r *http.Request) {
  gallery, err := g.galleryByID(w, r)
  if err != nil {
    return
  }
  var vd views.Data
  vd.Yield = gallery
  g.ShowView.Render(w,r, vd)
}
```

Рисунок 19 – Метод Show

Также создадим метод Edit (Рисунок 20). Данная функция обрабатывает маршрут GET /galleries/:id/edit, извлекает конкретную галерею на основе предоставленного ID галереи, проверяет, есть ли у пользователя разрешение на редактирование галереи, и отображает страницу редактирования галереи. Если в процессе поиска возникают ошибки или у пользователя нет прав, код возвращает соответствующий ответ об ошибке.

```
func (g *Galleries) Edit(w http.ResponseWriter, r *http.Request) {
  gallery, err := g.galleryByID(w, r)
  if err != nil {
    return
  }
  user := context.User(r.Context())
  if gallery.UserID != user.ID {
    http.Error(w, "Галлерея не найдена", http.StatusNotFound)
    return
  }
  var vd views.Data
  vd.Yield = gallery
  g.EditView.Render(w, r, vd)
}
```

Рисунок 20 – Метод Edit

Теперь напишем метод Update (Рисунок 21). Данный метод обрабатывает маршрут POST /galleries/:id/update, извлекает конкретную галерею на основе предоставленного ID, проверяет, есть ли у пользователя разрешение на обновление галереи, обновляет заголовок галереи с помощью

введенной формы и отображает EditView (страницу редактирования галереи) с соответствующими сообщениями, указывающими на успех или любые ошибки, возникшие в процессе обновления.

```
func (g *Galleries) Update(w http.ResponseWriter, r *http.Request) {
  gallery, err := g.galleryByID(w, r)
  if err != nil {
    return
  }
  user := context.User(r.Context())
  if gallery.UserID != user.ID {
    http.Error(w, "Галерея не найдена", http.StatusNotFound)
    return
  }
  var vd views.Data
  vd.Yield = gallery
  var form GalleryForm
  if err := parseForm(r, &form); err != nil {
    vd.SetAlert(err)
    g.EditView.Render(w, r, vd)
    return
  }
  gallery.Title = form.Title
  err = g.gs.Update(gallery)
  if err != nil {
    vd.SetAlert(err)
    g.EditView.Render(w, r, vd)
    return
  }
  vd.Alert = &views.Alert{
    Level: views.AlertLvlSuccess,
    Message: "Галерея обновлена",
  }
  g.EditView.Render(w, r, vd)
}
```

Рисунок 21 – Метод Update

Напишем метод Create (Рисунок 22) для создания новой галереи. Он будет обрабатывать маршрут POST /galleries, где пользователь может отправить форму для создания новой галереи. Данный код обрабатывает данные формы, предоставленные пользователем, создавая новый объект галереи с отправленными пользователем значениями и сохраняя их в базе

данных. Если в процессе разбора формы или создания галереи возникают какие-либо ошибки, код выдает предупреждающее сообщение и отображает представление New с заполненной формой и сообщением с ошибкой. Если галерея успешно создана, код перенаправляет пользователя на страницу редактирования созданной галереи.

```
func (g *Galleries) Create(w http.ResponseWriter, r *http.Request) {
  var vd views.Data
  var form GalleryForm
  if err := parseForm(r, &form); err != nil {
    vd.SetAlert(err)
    g.New.Render(w, r, vd)
    return
  }
  user := context.User(r.Context())
  gallery := models.Gallery{
    Title:  form.Title,
    UserID: user.ID,
  }
  if err := g.gs.Create(&gallery); err != nil {
    vd.SetAlert(err)
    g.New.Render(w, r, vd)
    return
  }
  url, err := g.r.Get("edit_gallery").URL("id", fmt.Sprintf("%v", gallery.ID))
  if err != nil {
    log.Println(err)
    http.Redirect(w, r, "/galleries", http.StatusFound)
    return
  }
  http.Redirect(w, r, url.Path, http.StatusFound)
}
```

Рисунок 22 – Метод Create

Далее напишем метод ImageUpload (Рисунок 23), содержащий код для загрузки изображений в галерею. Она обрабатывает маршрут POST /galleries/:id/images, где пользователь может загрузить несколько изображений в определенную галерею. Данный метод обрабатывает загрузку нескольких изображений в определенную галерею. Он выполняет парсинг формы, проверяет, есть ли у пользователя разрешение на загрузку изображений, то

есть является принадлежит ли галерея пользователю, сохраняет информацию о изображениях в базе данных и перенаправляет пользователя на страницу редактирования галереи после завершения загрузки.

```
func (g *Galleries) ImageUpload(w http.ResponseWriter, r *http.Request) {
  gallery, err := g.galleryByID(w, r)
  if err != nil {
    return
  }
  user := context.User(r.Context())
  if gallery.UserID != user.ID {
    http.Error(w, "Галерея не найдена", http.StatusNotFound)
    return
  }
  var vd views.Data
  vd.Yield = gallery
  err = r.ParseMultipartForm(maxMultipartMem)
  if err != nil {
    vd.SetAlert(err)
    g.EditView.Render(w, r, vd)
    return
  }
  files := r.MultipartForm.File["images"]
  for _, f := range files {
    // Open the uploaded file
    file, err := f.Open()
    if err != nil {
      vd.SetAlert(err)
      g.EditView.Render(w, r, vd)
      return
    }
    defer file.Close()
    err = g.is.Create(gallery.ID, file, f.Filename)
    if err != nil {
      vd.SetAlert(err)
      g.EditView.Render(w, r, vd)
      return
    }
  }
  url, err := g.r.Get("edit_gallery").URL("id", fmt.Sprintf("%v", gallery.ID))
  if err != nil {
    http.Redirect(w, r, "/galleries", http.StatusFound)
    return
  }
  http.Redirect(w, r, url.Path, http.StatusFound)
}
```

Рисунок 23 – Метод ImageUpload

Напишем метод для удаления изображения из галереи ImageDelete (Рисунок 23). Он обрабатывает маршрут POST /galleries/:id/images/:filename/delete, по которому пользователь может удалить конкретное изображение из галереи. В методе происходит проверка прав пользователя на удаление, удаляется информация о изображениях из базы

данных и происходит перенаправление на измененную страницу редактирования галереи после завершения удаления.

```
func (g *Galleries) ImageDelete(w http.ResponseWriter, r *http.Request) {
  gallery, err := g.galleryByID(w, r)
  if err != nil {
    return
  }
  user := context.User(r.Context())
  if gallery.UserID != user.ID {
    http.Error(w, "Галерея не найдена", http.StatusNotFound)
    return
  }
  filename := mux.Vars(r)["filename"]
  i := models.Image{
    Filename:  filename,
    GalleryID: gallery.ID,
  }
  err = g.is.Delete(&i)
  if err != nil {
    var vd views.Data
    vd.Yield = gallery
    vd.SetAlert(err)
    g.EditView.Render(w, r, vd)
    return
  }
  url, err := g.r.Get("edit_gallery").URL("id", fmt.Sprintf("%v", gallery.ID))
  if err != nil {
    log.Println(err)
    http.Redirect(w, r, "/galleries", http.StatusFound)
    return
  }
  http.Redirect(w, r, url.Path, http.StatusFound)
}
```

Рисунок 23 – ImageDelete

Также напишем вспомогательный метод galleryByID (Рисунок 24), с помощью которого многие методы-обработчики в контроллере Galleries получают конкретную галерею из базы данных на основе ID галереи. galleryByID извлекает галерею из базы данных на основе ее ID и заполняет ее соответствующими изображениями.

```go
func (g *Galleries) galleryByID(w http.ResponseWriter, r *http.Request) (*models.Gallery, error) {
  vars := mux.Vars(r)
  idStr := vars["id"]
  id, err := strconv.Atoi(idStr)
  if err != nil {
    log.Println(err)
    http.Error(w, "Недопустимый идентификатор галереи", http.StatusNotFound)
    return nil, err
  }
  gallery, err := g.gs.ByID(uint(id))
  if err != nil {
    switch err {
    case models.ErrNotFound:
      http.Error(w, "Галерея не найдена", http.StatusNotFound)
    default:
      log.Println(err)
      http.Error(w, "Что-то пошло не так.", http.StatusInternalServerError)
    }
    return nil, err
  }
  images, _ := g.is.ByGalleryID(gallery.ID)
  gallery.Images = images
  return gallery, nil
}
```

Рисунок 24 – galleryByID

3.2 Уровень моделей

3.2.1 Модель User

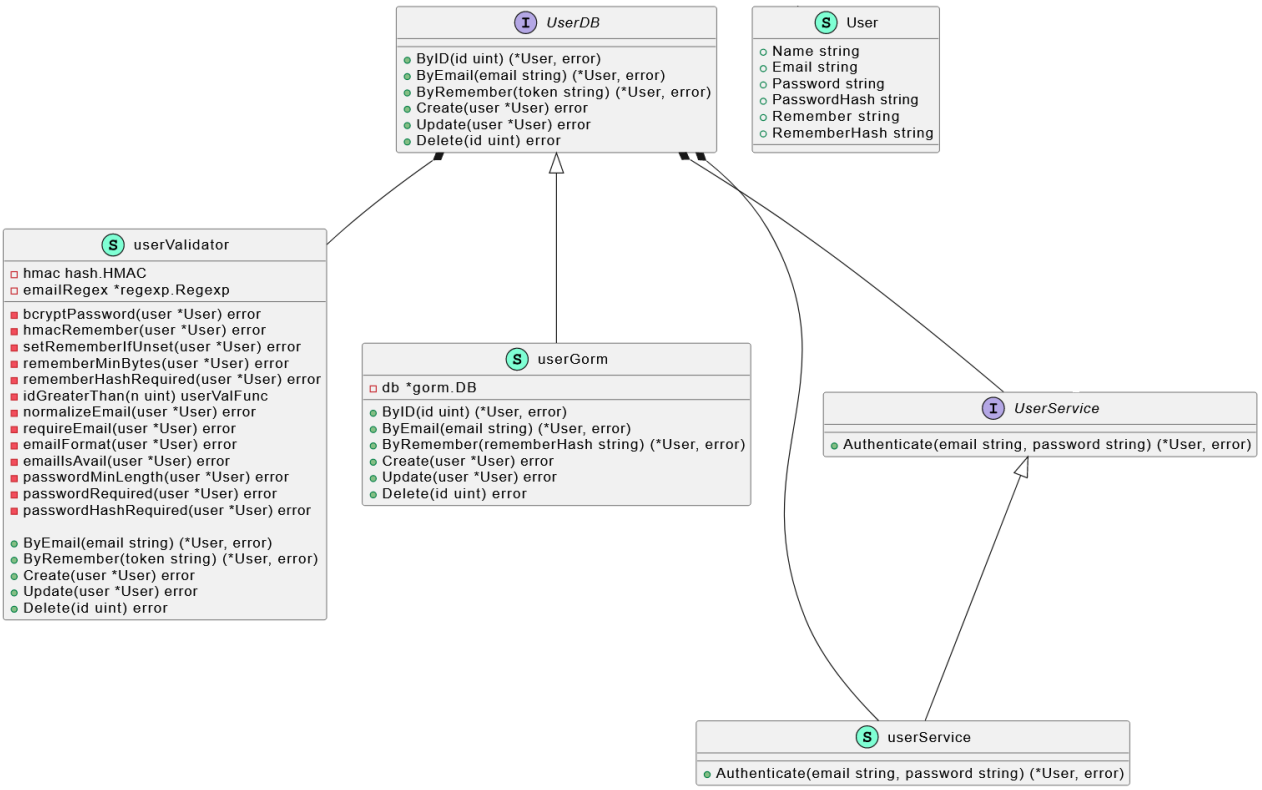


Рисунок 25 – Модель User

Рассмотрим модель User (Рисунок 25).

Структура User **(**Рисунок 26**)** представляет учетную запись пользователя в приложении. Она содержит следующие поля:

- Поле Name обозначает имя пользователя.
- Поле Email представляет собой адрес электронной почты пользователя. Оно имеет тег gorm, указывающий, что поле не должно быть нулевым и должно иметь уникальный индекс в базе данных, обеспечивая уникальность каждого адреса электронной почты.

- Поле Password используется для хранения пароля пользователя виде обычного текста. Оно имеет тег gorm, указывающий, что поле должно игнорироваться во время операций с базой данных, то есть оно не будет сохраняться в базе данных, так как хранить пароль как простой текст небезопасно.
- Поле PasswordHash хранит хешированную версию пароля пользователя. Оно имеет тег gorm, указывающий, что поле не может быть нулевым в базе данных.
- Поле Remember используется для токена remember, который является токеном, сгенерированным для сессии пользователя. Он используется для аутентификации пользователя, не требуя от него каждый раз входить в систему. Оно имеет тег gorm, указывающий, что поле следует игнорировать во время операций с базой данных.
- Поле RememberHash хранит хешированную версию маркера запоминания. Оно имеет тег gorm, указывающий, что поле не должно быть нулевым в базе данных и должно иметь уникальный индекс.

```
type User struct {
  gorm.Model
  Name string
  Email string `gorm:"not null;unique_index"`
  Password string `gorm:"-"`
  PasswordHash string `gorm:"not null"`
  Remember string `gorm:"-"`
  RememberHash string `gorm:"not null;unique_index"`
}
```

Рисунок 26 – Структура User

Далее создадим интерфейс UserDB (Рисунок 27) для взаимодействия с базой данных пользователей. UserDB определяет набор методов, которые можно использовать для запросов, создания, обновления и удаления записей пользователей. Опишем методы интерфейса:

- ByID(id uint) (*User, error): метод принимает на вход идентификатор пользователя и возвращает указатель на объект User, соответствующий этому идентификатору. Он извлекает одного пользователя из базы данных на основе предоставленного ID.
- ByEmail(email string) (*User, error): метод принимает в качестве входных данных адрес электронной почты и возвращает указатель на объект User, соответствующий этому адресу. Он извлекает одного пользователя из базы данных на основе предоставленного адреса электронной почты.
- ByRemember(token string) (*User, error): метод принимает на вход токен и возвращает указатель на объект User, соответствующий этому токену. Он извлекает одного пользователя из базы данных на основе предоставленного токена.
- Create(user *User) error: метод принимает указатель на объект User в качестве входных данных и создает новую запись пользователя в базе данных. Возвращает ошибку, если операция не удалась.
- Update(user *User) error: метод принимает на вход указатель на объект User и обновляет существующую запись пользователя в базе данных. Возвращает ошибку, если операция не удалась.
- Delete(id uint) error: Этот метод принимает на вход идентификатор пользователя и удаляет соответствующую запись пользователя из базы данных. Возвращает ошибку, если операция не удалась.

```
type UserDB interface {
  ByID(id uint) (*User, error)
  ByEmail(email string) (*User, error)
  ByRemember(token string) (*User, error)

  Create(user *User) error
  Update(user *User) error
  Delete(id uint) error
}
```

Рисунок 27 – Интерфейс UserDB

Создадим интерфейс UserService (Рисунок 28), в котором определены методы для работы с моделью пользователя. Данный интерфейс расширяет интерфейс UserDB, включая все методы, определенные в UserDB, в дополнение к своим собственным методам. User Service содержит:

- Authenticate(email, password string) (*User, error): метод принимает в качестве входных данных адрес электронной почты и пароль и пытается аутентифицировать пользователя. Он сверяет предоставленные учетные данные с хранящимися данными пользователя и возвращает указатель на объект аутентифицированного пользователя, если аутентификация прошла успешно. В противном случае возвращается ошибка, указывающая на причину неудачной аутентификации.
- UserDB: встроенный интерфейс указывает на то, что интерфейс UserService включает все методы, определенные в интерфейсе UserDB. Встраивая UserDB, интерфейс UserService наследует эти методы, позволяя службе пользователя взаимодействовать с базой данных пользователей, используя методы, определенные в UserDB.

Интерфейс UserService предоставляет абстракцию более высокого уровня для работы с операциями, связанными с пользователем [7]. Он объединяет функциональность аутентификации с основными операциями с базой данных, определенными в UserDB.

```
type UserService interface {
  Authenticate(email, password string) (*User, error)
  UserDB
}
```

Рисунок 28 – Интерфейс UserService

Напишем функцию NewUserService (Рисунок 29), которая инициализирует новый пользовательский сервис, создавая необходимые зависимости, такие как соединение с базой данных, хранилище

пользователей (реализуемое userGorm) и валидатор (реализуемый newUserValidator). Затем она возвращает новый экземпляр структуры userService, которая представляет пользовательский сервис, обеспечивающий абстракцию более высокого уровня для работы с операциями, связанными с пользователем.

```
func NewUserService(db *gorm.DB, pepper, hmacKey string) UserService {
  ug := &userGorm{db}
  hmac := hash.NewHMAC(hmacKey)
  uv := newUserValidator(ug, hmac, pepper)
  return &userService{
    UserDB:    uv,
    pepper:    pepper,
    pwResetDB: newPwResetValidator(&pwResetGorm{db}, hmac),
  }
}
```

Рисунок 29 – NewUserService

Данный код определяет конкретную реализацию интерфейса UserService с именем userService:

```
type userService struct {
   UserDB
}
```

Реализуем метод Authenticate (Рисунок 30) структуры userService. Метод аутентифицирует пользователя, сравнивая предоставленные им email и пароль с сохраненными значениями в базе данных. Он извлекает пользователя из базы данных на основе предоставленной электронной почты, используя метод ByEmail. Если при извлечении пользователя произошла ошибка, метод возвращает nil и ошибку. Метод сравнивает предоставленный пароль с хешем сохраненного пароля с помощью функции bcrypt.CompareHashAndPassword. Сохраненный хеш пароля извлекается из поля PasswordHash объекта foundUser. Если сравнение паролей не удалось из-за несовпадения, метод возвращает nil и ошибку ErrPasswordIncorrect.

Если во время сравнения паролей произошла любая другая ошибка, метод возвращает nil и ошибку. Если сравнение паролей прошло успешно, метод возвращает объект foundUser и nil для ошибки, что означает успешную аутентификацию.

```go
func (us *userService) Authenticate(email, password string) (*User, error) {
  foundUser, err := us.ByEmail(email)
  if err != nil {
    return nil, err
  }

  err = bcrypt.CompareHashAndPassword([]byte(foundUser.PasswordHash), []byte(password + userPwPepper))
  if err != nil {
    switch err {
    case bcrypt.ErrMismatchedHashAndPassword:
      return nil, ErrPasswordIncorrect
    default:
      return nil, err
    }
  }
  return foundUser, nil
}
```

Рисунок 30 – Метод Authenticate

3.2.2 Модель Gallery

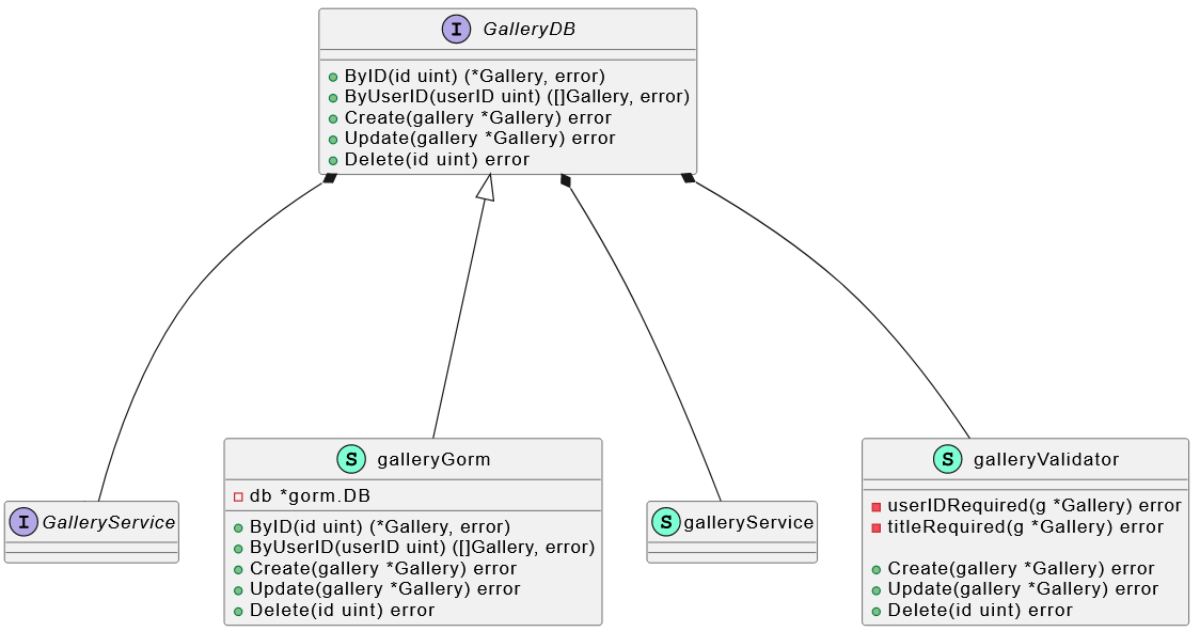


Рисунок 31

Рассмотрим модель галереи (Рисунок 31).

Структура Gallery (Рисунок 32) представляет модель галереи, она содержит следующие поля:

- поле gorm.Model встроено в структуру и предоставляет поля ID, CreatedAt, UpdatedAt и DeletedAt.
- поле UserID для хранения пользователя, которому принадлежит галерея. Оно имеет тип uint и тег gorm: "not_null;index", указывающий на то, что оно не может быть нулевым и должно быть проиндексировано в базе данных.
- поле Title для хранения названия галереи. Оно имеет тип string и тег gorm: "not_null", указывающий на то, что оно не может быть нулевым в базе данных.

Метод ImagesSplitN (рисунок) позволяет разделить изображения галереи на n отдельных групп, распределив изображения между ними. Это нужно для отображения изображений в виде сетки.

```
func (g *Gallery) ImagesSplitN(n int) [][]Image {
  ret := make([][]Image, n)
  for i := 0; i < n; i++ {
    ret[i] = make([]Image, 0)
  }
  for i, img := range g.Images {
    bucket := i % n
    ret[bucket] = append(ret[bucket], img)
  }
  return ret
}
```

Рисунок 32 – Метод ImagesSplitN

Интерфейс GalleryDB (Рисунок 33) определяет методы, которые используются для взаимодействия с базой данных для CRUD (Create, Read, Update, Delete) операций, связанных с моделью Gallery. Опишем методы:

- ByID(id uint) (*Gallery, error): метод извлекает галерею из базы данных на основе ее ID и возвращает соответствующий объект Gallery или ошибку, если он не существует.
- ByUserID(userID uint) ([]Gallery, error): извлекает все галереи, связанные с определенным ID пользователя, и возвращает срез Gallery или ошибку
- Create(gallery *Gallery) error: создает новую галерею в базе данных на основе предоставленного объекта Gallery, возвращает ошибку, если операция не удалась.
- Update(gallery *Gallery) error: обновляет существующую галерею в базе данных данными из объекта Gallery и возвращает ошибку, если обновление не удалось.
- Delete(id uint) error: удаляет галерею из базы данных на основе ее ID и возвращает ошибку, если операция удаления не удалась.

```
type GalleryDB interface {
  ByID(id uint) (*Gallery, error)
  ByUserID(userID uint) ([]Gallery, error)
  Create(gallery *Gallery) error
  Update(gallery *Gallery) error
  Delete(id uint) error
}
```

Рисунок 33– Интерфейс GalleryDB

Также напишем интерфейс GalleryService (Рисунок 34), включающий в себя интерфейс GalleryDB.Это нужно для того, чтобы инкапсулировать операции с базой данных для работы с галереями, отделяя функциональность, специфичную для базы данных (реализованную в GalleryDB), от более общих операций, связанных с галереей (определенных в GalleryService). Благодаря GalleryService, можно создавать различные реализации, которые взаимодействуют с различными базами данных или

источниками данных, обеспечивая при этом доступность необходимых методов для работы с галереями.

```
type GalleryService interface {
  GalleryDB
}
```

Рисунок 34 – Интерфейс GalleryService

3.2.3 Модель Image

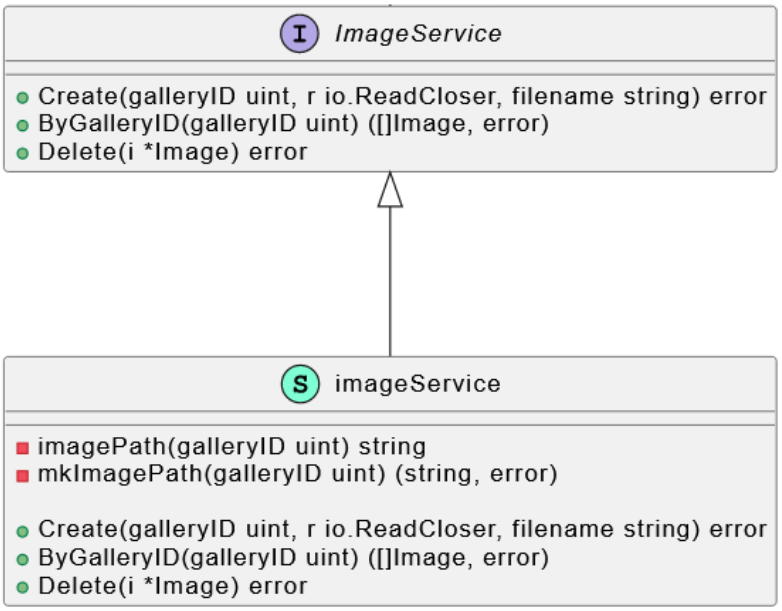


Рисунок 35 – Модель Image

Рассмотрим модель Image (Рисунок 35)

Структура Image (Рисунок 36) представляет изображение, связанное с галереей. Структура Image имеет два поля:

- GalleryID - целое число без знака, представляющее собой идентификатор галереи, к которой принадлежит изображение.
- Filename - строка, содержащая имя изображения. Изображения не хранятся непосредственно в базе данных, а хранятся в файловой системе. Поле GalleryID устанавливает связь между изображением и

соответствующей галереей с помощью ссылки на идентификатор галереи.

Интерфейс ImageService (Рисунок 36) определяет набор методов, используемых для манипулирования и работы с данными о изображениях. Имеет следующие методы:

- Create(galleryID uint, r io.ReadCloser, filename string) error: отвечает за создание нового изображения. Он принимает ID галереи, к которой принадлежит изображение, io.ReadCloser, представляющий данные файла изображения, и имя файла изображения. Она возвращает ошибку, если в процессе создания возникают какие-либо проблемы.
- ByGalleryID(galleryID uint) ([]Image, error): извлекает список изображений, связанных с определенным идентификатором галереи.
- Delete(i *Image) error: метод используется для удаления изображения

```
type ImageService interface {
  Create(galleryID uint, r io.ReadCloser, filename string) error
  ByGalleryID(galleryID uint) ([]Image, error)
  Delete(i *Image) error
}
```

Рисунок 36 – Структура Image

3.3 Хеширование пароля

Для пароля понадобится два поля в модели User – Password и PasswordHash. Поле Password Первое будет использоваться для нехешированного пароля. Мы не будем сохранять это поле в базе данных. Библиотека GORM предоставляет нам способ гарантировать это с помощью структурного тега “-”. Второе поле – PasswordHash будет содержать хешированный пароль. Это поле мы будем хранить в нашей базе данных.

Используя два поля пароля, мы гарантируем, что их нельзя легко перепутать. Если бы мы использовали одно поле пароля, было бы трудно определить, был ли пароль хеширован или нет, что может привести к случайному хэшированию пароля, который уже был хэширован, или, что еще хуже. можно забыть хешировать необработанный пароль.

```
func (uv *userValidator) bcryptPassword(user *User) error {
  if user.Password == "" {
    return nil
  }
  pwBytes := []byte(user.Password + userPwPepper)
  hashedBytes, err := bcrypt.GenerateFromPassword(pwBytes, bcrypt.DefaultCost)
  if err != nil {
    return err
  }
  user.PasswordHash = string(hashedBytes)
  user.Password = ""
  return nil
}
```

Рисунок 37 – Хеширование пароля

Рассмотрим хеширование пароля в нашем приложении (Рисунок 37). Функция GenerateFromPassword принимает два аргумента: пароль в виде среза байтов ([]byte) и параметры хеширования (cost int). Параметр cost определяет стоимость (или время) выполнения хеширования, который влияет на сложность работы алгоритма и, соответственно, на безопасность полученного хеша. Более высокое значение cost требует больше времени и ресурсов для вычисления хеша, что делает его более стойким к атакам перебора. В нашем случае мы использовали DefaultCost. Получившееся значение мы сохраняем в поле PasswordHash, а поле Password очищаем, присвоив ей пустую строку.

4 Результат

На Рисунке 38 представлена страница регистрации пользователя. Пользователь вводит имя, электронный адрес и пароль, отвечающий определенным характеристикам.

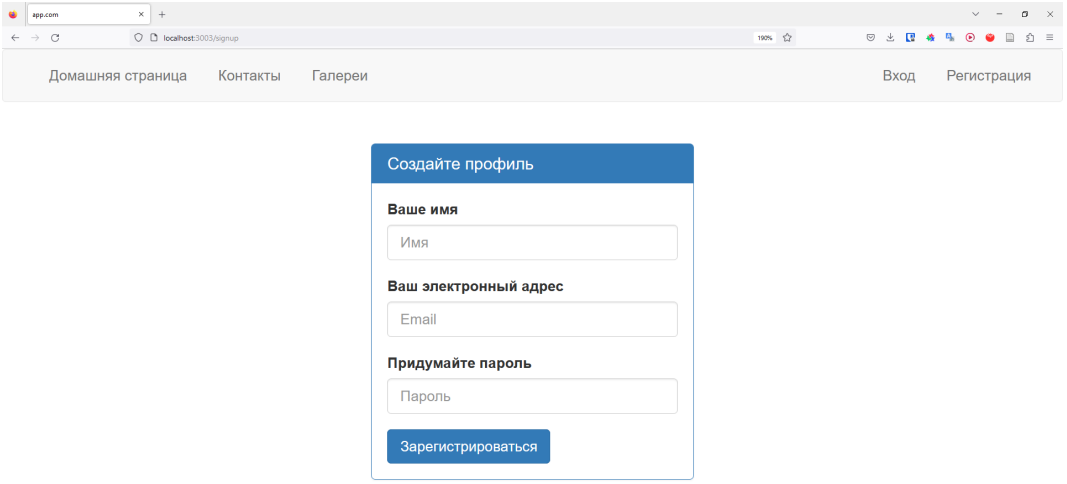


Рисунок 38 – Страница регистрации профиля

На Рисунке 39 изображена страница входа в профиль.

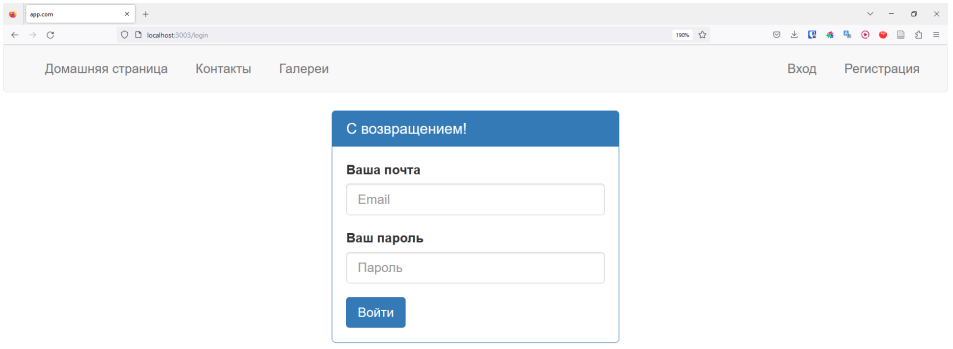


Рисунок 39 – Страница входа в профиль

При авторизации пользователь перенаправляется на страницу со списком галерей пользователя. Страница показана на Рисунке 40.

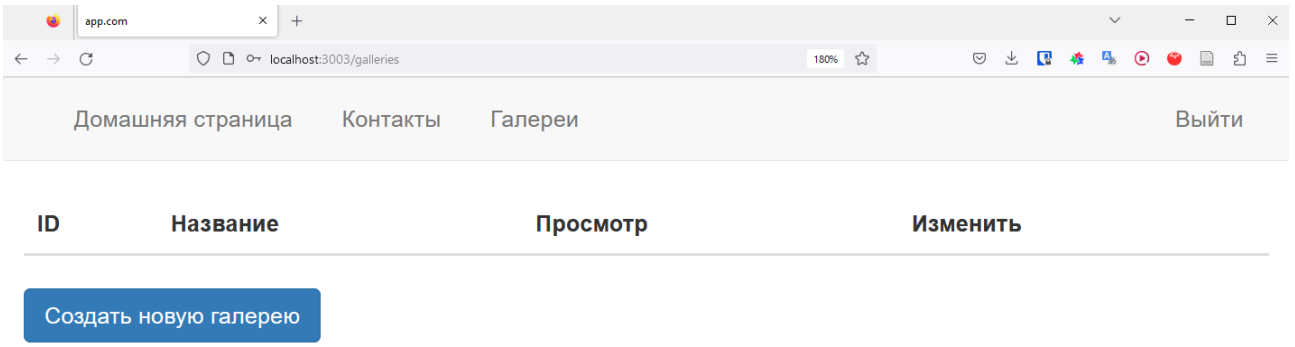


Рисунок 40 – Страница со списком галерей пользователя, на которую перенаправляется пользователь при авторизации

Когда пользователь нажимают кнопку “Создать новую галерею”, происходит переход на страницу создания галереи, показанную на Рисунке 41, где пользователь вписывает название для своей галереи.

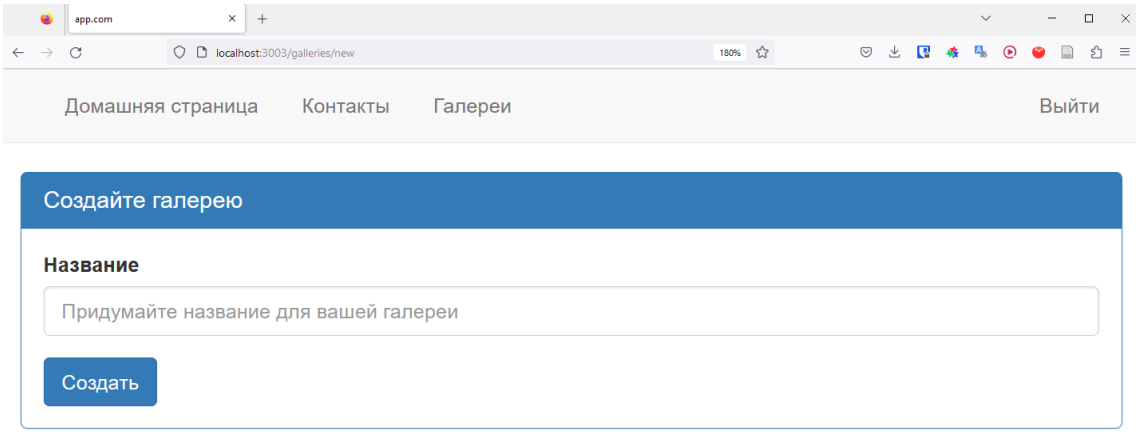


Рисунок 41 – Страница создания галерей

После того как пользователь создает галерею, он переходит на страницу редактирования галереи (Рисунок 42), где пользователь загружает изображения. Также есть возможность удалить определенные изображения или удалить всю галерею.

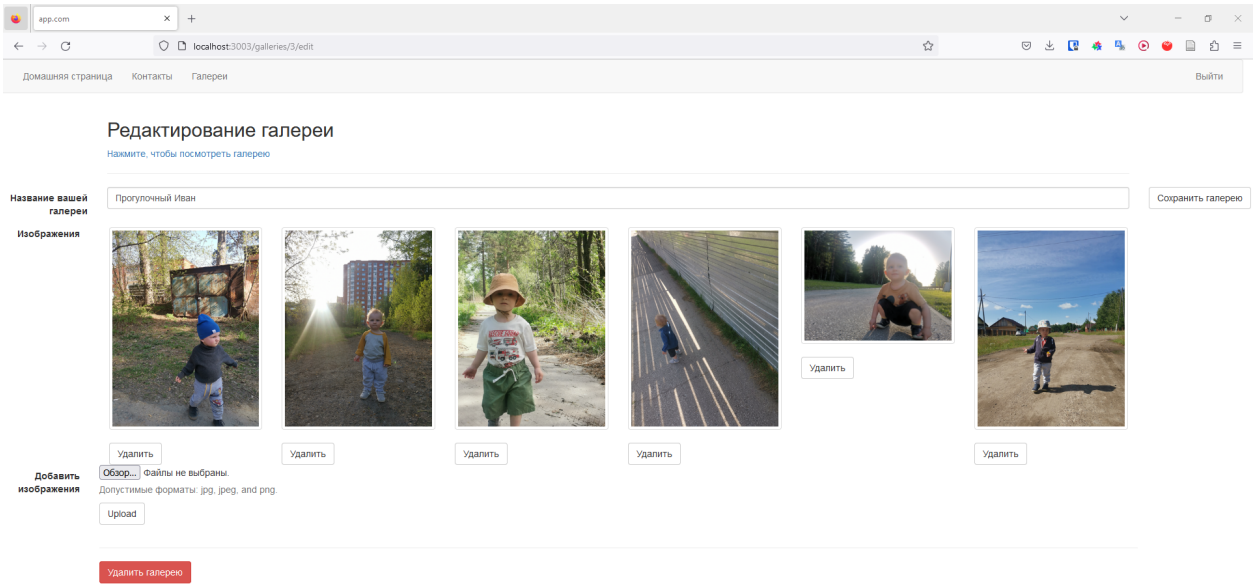


Рисунок 42 – Страница редактирования галереи

После загрузки изображений, пользователь может просмотреть созданную им галерею (Рисунок 43).

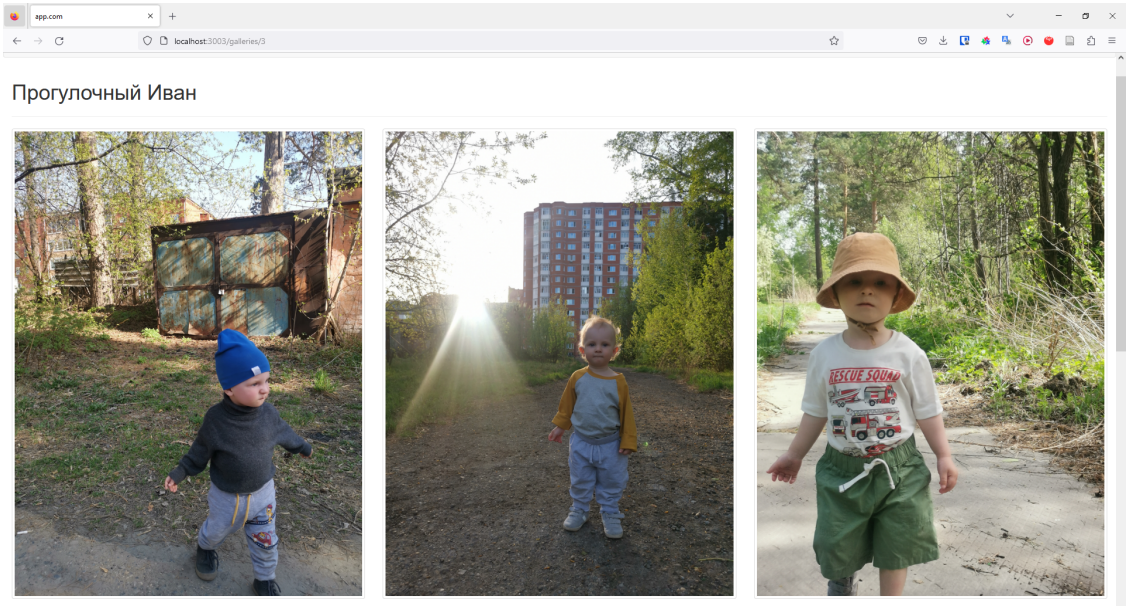


Рисунок 43 – Страница просмотра галереи

Созданная галерея пользователя отображается в списке всех галерей данного пользователя (Рисунок 44).

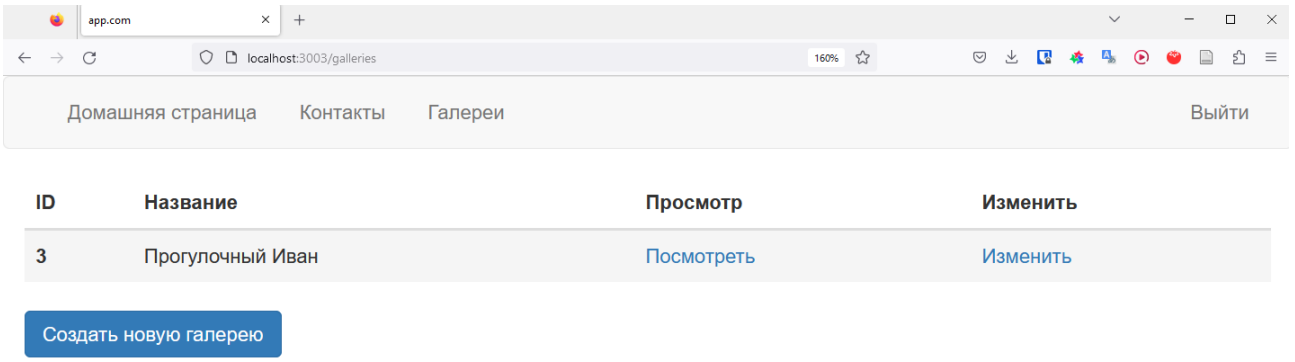


Рисунок 44 – Страница просмотра всех галерей пользователя

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы был проведен анализ существующих решений, определен стек технологий для разработки веб-приложения, описаны требования к приложению, определена архитектура приложения, спроектирована база данных, описана разработка приложения.

Результатом ВКР является веб-приложение для создания фотогалерей пользователями. Веб-приложение дает возможность создавать именованные фотогалереи, загружать и удалять изображения из галерей, удалять галереи, просматривать галереи, а также делиться ими с помощью URL-ссылки.

Существуют дальнейшие способы улучшения системы, например:

- добавление социальных функций, например, возможность добавление комментариев
- Добавление поиска по галереям
- Добавление системы тегов

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Writing Web Applications [Электронный ресурс] - URL: https://go.dev/doc/articles/wiki/ (дата обращения: 02.05.2023).

2. PostgreSQL: The world's most advanced open source database [Электронный ресурс] - URL: https://www.postgresql.org/ (дата обращения: 02.05.2023)

3. GORM - The fantastic ORM library for Golang [Электронный ресурс] - URL: https://gorm.io/ (дата обращения: 02.05.2023)

4. Get started with Bootstrap [Электронный ресурс] - URL: https://getbootstrap.com/docs/3.4/ (дата обращения: 02.05.2023)

5. MVC - MDN Web Docs Glossary [Электронный ресурс] - URL: https://developer.mozilla.org/en-US/docs/Glossary/MVC (дата обращения: 02.05.2023)

6. Hashing Passwords: One-Way Road to Security [Электронный ресурс] - URL: https://auth0.com/blog/hashing-passwords-one-way-road-to-security/ (дата обращения: 02.05.2023)

7. Go (Golang) - understanding the object oriented features with structs, methods, and interfaces [Электронный ресурс] - URL: https://unixsheikh.com/articles/go-understanding-the-object-oriented-features-with-structs-methods-and-interfaces.html (дата обращения: 02.05.2023)

АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

ознакомлен
Акимов С.В.

Отчет о проверке на заимствования №1

Автор: Черных Егор Михайлович
Проверяющий: Нечаева Татьяна Сергеевна
Организация: Томский Государственный Университет

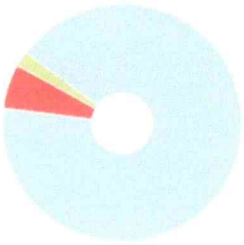
Отчет предоставлен сервисом «Антиплагиат» - http://tsu.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 423
Начало загрузки: 07.06.2023 08:16:18
Длительность загрузки: 00:00:10
Имя исходного файла: вкр Черных2-8.pdf
Название документа: Веб-приложение для создания фотогалерей
Размер текста: 46 кБ
Тип документа: Выпускная квалификационная работа
Символов в тексте: 46918
Слов в тексте: 5440
Число предложений: 354

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 07.06.2023 08:16:29
Длительность проверки: 00:01:41
Комментарии: не указано
Поиск с учетом редактирования: да
Проверенные разделы: титульный лист с. 1, содержание с. 2, основная часть с. 3-50, библиография с. 51
Модули поиска: ИПС Адилет, Библиография, Сводная коллекция ЭБС, Интернет Плюс*, Сводная коллекция РГБ, Цитирование, Переводные заимствования (RuEn), Переводные заимствования по eLIBRARY.RU (EnRu), Переводные заимствования по коллекции Гарант: аналитика, Переводные заимствования по коллекции Интернет в английском сегменте, Переводные заимствования по Интернету (EnRu), Переводные заимствования по коллекции Интернет в русском сегменте, Переводные заимствования издательства Wiley , eLIBRARY.RU, СПС ГАРАНТ: аналитика, СПС ГАРАНТ: нормативно-правовая документация, Медицина, Диссертации НББ, Коллекция НБУ, Перефразирования по eLIBRARY.RU, Перефразирования по СПС ГАРАНТ: аналитика, Перефразирования по Интернету, Перефразирования по Интернету (EN), Перефразированные заимствования по коллекции Интернет в английском сегменте, Перефразированные заимствования по коллекции Интернет в русском сегменте, Перефразирования по коллекции издательства Wiley , Патенты СССР, РФ, СНГ, СМИ России и СНГ, Шаблонные фразы, Модуль поиска "tsu", Кольцо вузов, Издательство Wiley, Переводные заимствования



СОВПАДЕНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
5,51%	0%	2,15%	92,34%

Совпадения — фрагменты проверяемого текста, полностью или частично сходные с найденными источниками, за исключением фрагментов, которые система отнесла к цитированию или самоцитированию. Показатель «Совпадения» – это доля фрагментов проверяемого текста, отнесенных к совпадениям, в общем объеме текста.

Самоцитирования — фрагменты проверяемого текста, совпадающие или почти совпадающие с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа. Показатель «Самоцитирования» – это доля фрагментов текста, отнесенных к самоцитированию, в общем объеме текста.

Цитирования — фрагменты проверяемого текста, которые не являются авторскими, но которые система отнесла к корректно оформленным. К цитированиям относятся также шаблонные фразы; библиография; фрагменты текста, найденные модулем поиска «СПС Гарант: нормативно-правовая документация». Показатель «Цитирования» – это доля фрагментов проверяемого текста, отнесенных к цитированию, в общем объеме текста.

Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.

Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.

Оригинальный текст — фрагменты проверяемого текста, не обнаруженные ни в одном источнике и не отмеченные ни одним из модулей поиска. Показатель «Оригинальность» – это доля фрагментов проверяемого текста, отнесенных к оригинальному тексту, в общем объеме текста.

«Совпадения», «Цитирования», «Самоцитирования», «Оригинальность» являются отдельными показателями, отображаются в процентах и в сумме дают 100%, что соответствует полному тексту проверяемого документа.

Обращаем Ваше внимание, что система находит текстовые совпадения проверяемого документа с проиндексированными в системе источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности совпадений или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в тексте	Источник	Актуален на	Модуль поиска	Комментарии
[01]	2,44%	https://elib.sfu-kras.ru/bitstream/handle/2311/112887/belyaev_n.yu_.... https://elib.sfu-kras.ru	27 Апр 2023	Перефразированные заимствования по коллекции Интернет в русском сегменте	
[02]	2,15%	не указано	29 Сен 2022	Библиография	
[03]	1,81%	gogolev_d_v_razrabotka-informacionnoy-sistemy-dlya-podgotovki-tu...	26 Мар 2023	Кольцо вузов	
[04]	1,78%	Разработка веб-платформы управления и публикации научных ст... https://core.ac.uk	21 Янв 2023	Перефразированные заимствования по коллекции Интернет в русском сегменте	
[05]	1,22%	черновик Марников (1).docx	10 Июн 2020	Модуль поиска "tsu"	
[06]	1,2%	MAGISTERSKAYA_POLISHCHUK_022110- (1).docx	27 Мая 2023	Модуль поиска "tsu"	
[07]	1,02%	Диссертация	25 Мая 2021	Модуль поиска "tsu"	