

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В этом разделе подробно рассматривается функционирование программных модулей. Перечислены соответствующие классы, а также дано описание большинства их компонентов. Состав основных классов показан также на диаграмме классов (чертеж ГУИР.400201.047 РР.1).

3.1 Классы веб-приложения

3.1.1 Класс User

3.1.2 Этот класс является отображением структуры данных пользователя и предоставляет методы для работы с ними. Данный класс агрегирует класс Image отношением `has_many`, что означает, что у одного пользователя может быть много изображений с устройства.

В классе присутствуют поля.

Поля:

- `id` – поле типа `BigInt`, представляющее собой целое число, содержащее в себе идентификационный номер пользователя;
- `email` – поле типа `String`, представляющее собой строку, хранящую в себе адрес электронной почты пользователя;
- `encrypted_password` – поле типа `String`, представляющее собой строку, хранящую в себе зашифрованный пароль пользователя для авторизации в приложении;
- `reset_password_token` – поле типа `String`, представляющее собой строку, хранящую в себе ключ, необходимый для восстановления пароля пользователя для авторизации в приложении;
- `reset_password_sent_at` – поле типа `DateTime`, представляющее собой дату и время отсылки пользователю на адрес электронной почты сообщения для восстановления пароля;
- `remember_created_at` – поле типа `DateTime`, представляющее собой дату и время последней установки пользователем опции «Запомнить меня» при авторизации;
- `sign_in_count` – поле типа `Integer`, представляет собой целое число, которое отображает количество входов пользователя в приложение;
- `current_sign_in_at` – поле типа `DateTime`, представляющее собой дату и время авторизации пользователя в текущей сессии;
- `last_sign_in_at` – поле типа `DateTime`, представляющее собой дату и время авторизации пользователя в последней сессии;
- `current_sign_in_ip` – поле типа `String`, представляющее собой строку, хранящую в себе ip-адрес пользователя в текущей сессии;
- `last_sign_in_ip` – поле типа `String`, представляющее собой строку, хранящую в себе ip-адрес пользователя в последней сессии;

- `created_at` – поле типа `DateTime`, представляющее собой дату и время создания этого пользователя;
- `updated_at` – поле типа `DateTime`, представляющее собой дату и время последнего обновления данных этого пользователя;
- `user` – поле типа `Boolean`, представляющее собой флаг, который указывает является ли пользователь обычным пользователем в приложении;
- `admin` – поле типа `Boolean`, представляющее собой флаг, который указывает является ли пользователь администратором в приложении;
- `images` – поле типа `Array`, представляющее собой массив, содержащий объекты класса `Image`;

3.1.3 Класс `DeviseController`

Класс представляет из себя абстрактный класс, который описывает общее поведение всех классов, являющихся контроллерами пользователя.

В классе присутствуют методы.

Методы:

1. `resource()` – метод, который возвращает актуальный объект ресурса, хранящегося в переменной экземпляра данного класса.
2. `resource_name()` – метод, возвращающий строку с именем ресурса.
3. `resource_class()` – метод, возвращающий класс, экземпляром которого является данный ресурс.
4. `signed_in_resource()` – метод, возвращающий ресурс, который был авторизован в текущей сессии приложения.
5. `device_mapping()` – метод, который ищет соответствующий маршрут для контроллера на основании предоставленного пути запроса.
6. `assert_is_devise_resource!()` – метод, который проверяет, является ли объект ресурсом данного контроллера.
7. `navigational_formats()` – метод, который возвращает действительные форматы навигации, поддерживаемые Rails.
8. `unknown_action!(msg)` – метод, который создает исключение `ActionNotFound` с сообщением, переданным аргументом `msg`.
9. `resource=(new_resource)` – метод, который устанавливает значение ресурса данного контроллера равным аргументу `new_resource`.
10. `require_no_authentication()` – метод, который указывает, что данный ресурс не требует аутентификации в приложении.
11. `successfully_sent?(resource)` – метод, который проверяет, был ли ресурс, представленный аргументом `resource`, успешно послан.
12. `set_flash_message(key, kind, options)` – метод, который настраивает сообщения уведомлений с помощью ключа, представленного аргументом `key`, и опций, представленных в `options`, используя `il8n`. Можно настроить собственные уведомления для

определенного набора ресурсов, если ресурс не входит в этот набор, для него будет использовано стандартное сообщение.

13. `set_minimum_password_length()` – метод, который устанавливает минимальную длину пароля для пользователя.

14. `devise_i18n_options(options)` – метод, который устанавливает пользовательские опции для `i18n` в области применения данного контроллера, значение которых содержится в аргументе `options`.

15. `find_message(kind, options)` – метод, который ищет сообщение уведомления по заданным входным параметрам представленным аргументами `kind` и `options`.

16. `translation_scope()` – метод, который рекомендуется к переопределению наследниками `DeviseController`, так как он позволит им использовать расширенный набор переводов на различные языки для их сообщений, использующихся в уведомлениях.

17. `clean_up_passwords(object)` – метод, который удаляет все пароли относящиеся к этому объекту.

18. `respond_with_navigational(args, block)` – метод, который формирует ответ на основе навигационных форматов для данного ресурса.

19. `resource_params()` – метод, который возвращает параметры данного ресурса.

3.1.3 Класс `User::RegistrationsController`

Класс является контроллером, который отвечает за обработку запросов направленных на регистрацию пользователя в системе. Данный класс является наследником класса `DeviseController`, являющегося родителем для всех контроллеров в пространстве имен `User`. Класс содержит поля и методы.

Поля:

– `user` – объект класса `User`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

Методы:

1. `new()` – метод, который обрабатывает запрос на регистрацию нового пользователя, в свою очередь вызывает метод `create()`.

2. `create()` – метод, который создает нового пользователя и сохраняет его в базе данных.

3. `edit()` – метод, который обрабатывает запрос на изменение данных пользователя, вызывает метод `update()`.

4. `update()` – метод, который изменяет данные пользователя и сохраняет их в базе данных.

5. `destroy()` – метод, который обрабатывает запрос на удаление пользователя.

6. `cancel()` – метод, который сбрасывает несохраненные данные

пользователя после завершения сессии.

7. `update_needs_confirmation?(resource, previous)` – метод, который возвращает флаг, определяющий необходимо ли подтверждение в виде пароля для изменения данных пользователя.

8. `update_resource(resource, params)` – метод, который обновляет ресурс, представленный аргументом `resource`, в соответствии с параметрами из аргумента `params`.

9. `build_resource(params)` – метод, который создает ресурс с параметрами из аргумента `params`.

10. `sign_up(resource_name, resource)` – метод, который отвечает за авторизацию пользователя сразу после регистрации.

11. `after_sign_up_path_for(resource)` – метод, который возвращает путь для перенаправления пользователя сразу после регистрации.

12. `after_inactive_sign_up_path_for(resource)` – метод, который возвращает путь для перенаправления пользователя сразу после регистрации для пока неактивированных аккаунтов.

13. `after_update_path_for(resource)` – метод, который возвращает путь для перенаправления после изменения данных ресурса, представленного аргументом `resource`.

14. `authenticate_scope!()` – метод, который аутентифицирует текущий набор данных и получает доступ к текущей сессии.

15. `sign_up_params()` – метод, который обеспечивает фильтрацию параметров, которая допускает только параметры необходимые при регистрации.

16. `account_update_params()` – метод, который обеспечивает фильтрацию параметров, которая допускает только параметры необходимые при изменении данных пользователя.

17. `translation_scope()` – метод, который позволяет использовать расширенный набор переводов для сообщений уведомлений.

3.1.4 Класс `User::PasswordsController`

Класс является контроллером, который отвечает за обработку запросов направленных на взаимодействие с паролями пользователя в системе. Данный класс является наследником класса `DeviseController`, являющегося родителем для всех контроллеров в пространстве имен `User`. Класс содержит поля и методы.

Поля:

– `user` – объект класса `User`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

Методы:

1. `new()` – метод, который обрабатывает запрос на создание нового пароля, в свою очередь вызывает метод `create()`.

2. `create()` – метод, который создает новый пароль и сохраняет его в базу данных.

3. `edit()` – метод, который обрабатывает запрос на изменение пароля пользователя, вызывает метод `update()`.

4. `update()` – метод, который изменяет пароль пользователя и сохраняет новый пароль в базу данных.

5. `after_resetting_password_path_for(resource)` – метод, который возвращает путь для перенаправления после восстановления пароля для ресурса представленного в аргументе `resource`.

6. `after_sending_reset_password_instructions_path_for(resource_name)` – метод, возвращающий путь для перенаправления после того, как пользователю на электронную почту были высланы инструкции по восстановлению пароля от учетной записи.

7. `assert_reset_token_passed()` – метод, проверяющий наличие ключа, необходимого для восстановления пароля, в запросе.

8. `unlockable?(resource)` – метод, который проверяет возможность разблокировки ресурса представленного в аргументе `resource`.

9. `translation_scope()` – метод, который позволяет использовать расширенный набор переводов для сообщений уведомлений.

3.1.5 Класс `User::OmniauthCallbacksController`

Класс является контроллером, который отвечает за авторизацию в системе через аккаунты сторонних приложений, таких как Вконтакте, Facebook, Twitter. Данный класс является наследником класса `DeviseController`, являющегося родителем для всех контроллеров в пространстве имен `User`. Класс содержит поля и методы.

Поля:

– `user` – объект класса `User`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

Методы:

1. `passthru()` – метод, который обрабатывает запрос на авторизацию через стороннее приложение и авторизует, если параметры валидны.

2. `failure()` – метод, который обрабатывает неудачную попытку авторизации через стороннее приложение. Вызывает методы `failed_strategy()`, `failure_message()` и `after_omniauth_failure_path_for(scope)`.

3. `failed_strategy()` – метод, который формирует http-ответ на неудачную попытку авторизации.

4. `failure_message()` – метод, который возвращает сообщение о неудачной попытке авторизации и ее причинах.

5. `after_omniauth_failure_path_for(scope)` – метод,

который возвращает путь для перенаправления пользователя после неудачной попытки авторизации посредством стороннего приложения.

6. `translation_scope()` – метод, который позволяет использовать расширенный набор переводов для сообщений уведомлений.

3.1.6 Класс `User::ConfirmationsController`

Класс является контроллером, который отвечает за подтверждение и активацию пользователей в системе. Данный класс является наследником класса `DeviseController`, являющегося родителем для всех контроллеров в пространстве имен `User`. Класс содержит поля и методы.

Поля:

– `user` – объект класса `User`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

Методы:

1. `new()` – метод, который обрабатывает запрос на активацию аккаунта.

Вызывает метод `create()`.

2. `create()` – метод, который активирует аккаунт пользователя.

3. `show()` – метод, который обрабатывает запрос на отображение страницы с инструкцией по активации аккаунта для пользователя.

4.

`after_resending_confirmation_instructions_path_for(resource_name)` – метод, который возвращает путь для перенаправления пользователя после повторной высылки инструкций по активации аккаунта на электронную почту пользователя.

5. `after_confirmation_path_for(resource_name, resource)` – метод, который возвращает путь для перенаправления пользователя после активации аккаунта.

6. `translation_scope()` – метод, который позволяет использовать расширенный набор переводов для сообщений уведомлений.

3.1.7 Класс `User::SessionsController`

Класс является контроллером, который отвечает за аутентификацию пользователей в системе. Данный класс является наследником класса `DeviseController`, являющегося родителем для всех контроллеров в пространстве имен `User`. Класс содержит поля и методы.

Поля:

– `user` – объект класса `User`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

Методы:

1. `new()` – метод, который обрабатывает запрос на вход пользователя в систему. Вызывает метод `create()`.

2. `create()` – метод, который аутентифицирует пользователя в

системе и предоставляет ему доступ к текущей сессии.

3. `destroy()` – метод, который обрабатывает запрос на выход пользователя из системы и закрывает его сессию.

4. `auth_options()` – метод, который возвращает набор параметров необходимых для аутентификации пользователя в системе.

5. `serialize_options(resource)` – метод, который сериализует параметры ресурса и возвращает их в виде Hash.

6. `sign_in_params()` – метод, который обеспечивает фильтрацию параметров, которая допускает только параметры необходимые при аутентификации.

7. `translation_scope()` – метод, который позволяет использовать расширенный набор переводов для сообщений уведомлений.

3.1.8 Класс `User::UnlocksController`

Класс является контроллером, который отвечает за разблокировку учетной записи пользователя в системе. Данный класс является наследником класса `DeviseController`, являющегося родителем для всех контроллеров в пространстве имен `User`. Класс содержит поля и методы.

Поля:

– `user` – объект класса `User`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

Методы:

1. `new()` – метод, который обрабатывает запрос на разблокировку учетной записи пользователя. Вызывает метод `create()`.

2. `create()` – метод, который разблокирует учетную запись пользователя в системе.

3. `show()` – метод, который обрабатывает запрос на отображение страницы с формой запроса на разблокировку учетной записи.

4. `after_unlock_path_for(resource)` – метод, который возвращает путь для перенаправления пользователя после разблокировки учетной записи.

5.

`after_sending_unlock_instructions_path_for(resource)` – метод, который возвращает путь для перенаправления пользователя после отправки инструкций по разблокировке учетной записи на электронную почту пользователя.

6. `translation_scope()` – метод, который позволяет использовать расширенный набор переводов для сообщений уведомлений.

3.1.9 Класс `ActiveRecord::Base`

Класс, являющийся классом-родителем для всех классов-моделей в веб-приложении. Объекты `ActiveRecord` не указывают все свои атрибуты

напрямую, а выводят их также из схемы таблицы базы данных, с которой они связаны. Добавление, удаление и изменение атрибутов и их типа выполняется непосредственно в базе данных. Любые изменения мгновенно отражаются в объектах ActiveRecord. Отображение, которое привязывает данный класс ActiveRecord к определенной таблице базы данных, будет происходить автоматически в большинстве случаев, но также может быть перезаписано для исключительных случаев. Класс имеет также нижеперечисленные поля.

Поля:

- `id` – поле, хранящее идентификационный номер объекта модели в таблице базы данных;
- `name` – поле, хранящее в себе название объекта модели в таблице базы данных;
- `core` – поле, являющееся объектом класса `Core`;
- `persistence` – поле, являющееся объектом класса `Persistence`, который обеспечивает механизмы неизменности атрибутов объекта;
- `readonly_attributes` – поле, которое является объектом класса `ReadonlyAttributes` и предоставляет механизмы для установки атрибутов объекта в режим «только чтение»;
- `model_schema` – объект класса `ModelSchema`, который обеспечивает привязку атрибутов к схеме таблицы базы данных;
- `inheritance` – поле, являющееся объектом класса `Inheritance`, предоставляющего дополнительные механизмы реализации наследования классов;
- `scoping` – поле, которое является объектом класса `Scoping`, который в свою очередь, предоставляет механизмы группирования объектов в определенной области;
- `sanitization` – поле, которое представлено объектом класса `Sanitization`, предоставляющего механизмы фильтрации входящих аргументов, что предоставляет дополнительную защиту от уязвимостей, связанных с подменой входных параметров;
- `attribute_assignment` – поле, которое является объектом класса `AttributeAssignment`, предоставляющее механизмы расширяющие стандартные для присваивания переменным значений;
- `active_model_conversion` – поле, которое является объектом класса `ActiveModel::Conversion`. Предоставляет механизмы конверсии модели;
- `integration` – поле, которое является объектом класса `Integration`, предоставляющим механизм интеграции модели в кэш;
- `validations` – поле, которое является объектом класса `Validations`, предоставляющим механизмы валидации данных в модели;
- `counter_cache` – объект класса `CounterCache`, который предоставляет доступ к счетчику кэша;

- `attributes` – поле, которое является объектом класса `Attributes`;
- `attribute_decorators` – поле, которое является объектом класса `AttributeDecorators`, обеспечивающий механизм декорации для атрибутов объекта;
- `locking_optimistic` – поле, которое представлено объектом класса `Locking::Optimistic`, которое предоставляет «оптимистичный» механизм блокирования базы данных, который позволяет вносить изменения нескольким объектам одновременно и если они пересекаются, то выбрасывается исключение и все изменения, кроме первого отбрасываются;
- `locking_pessimistic` – поле, которое представлено объектом класса `Locking::Pessimistic`, которое предоставляет «пессимистичный» механизм блокирования базы данных, который позволяет вносить изменения единовременно только одному объекту;
- `define_callbacks` – поле, которое является объектом класса `DefineCallbacks`, предоставляющее механизм объявления «колбэков»;
- `attribute_methods` – поле, которое является объектом класса `Attribute::Methods`, предоставляющее механизм работы с методами атрибутов класса
- `callbacks` – поле, которое является объектом класса `Callbacks`, предоставляющее механизм работы с «колбэками»;
- `timestamp` – поле, которое является объектом класса `Timestamps`. Данное поле позволяет хранить точные дату и время изменения объекта;
- `associations` – поле, которое является объектом класса `Associations`, предоставляющее механизм работы с ассоциациями объекта с другими объектами;
- `active_model_secure_password` – поле, которое является объектом класса `ActiveModel::SecurePassword`, предоставляющее механизм работы с паролями и способы их шифровки;
- `autosave_association` – поле, которое является объектом класса `AutosaveAssociations`, предоставляющее механизм работы с автоматическим сохранением либо удалением объекта при изменении иного объекта, с которым он ассоциирован;
- `nested_attributes` – поле, которое является объектом класса `NestedAttributes`, предоставляющее механизм работы со сложными атрибутами класса, которые в свою очередь представляют из себя другие классы с подклассами;
- `aggregations` – поле, которое является объектом класса `Aggregations`, предоставляющее механизм работы с агрегацией объектом других объектов;
- `transactions` – поле, которое является объектом класса `Transactions`, предоставляющее механизм работы с транзакциями в базу

данных;

- `touch_later` – поле, которое является объектом класса `TouchLater`, который отвечает за выбрасывание исключения, при попытке провести транзакцию в базу данных. Исключение уведомляет, что в данный момент база данных занята и предлагает провести транзакцию позже;

- `no_touching` – поле, которое является объектом класса `NoTouching`, который отвечает за выбрасывание исключения, при попытке провести транзакцию в базу данных. Исключение запрещает проводить данную транзакцию;

- `reflection` – поле, которое является объектом класса `Reflection`, который содержит в себе все метаданные об ассоциациях данного класса;

- `serialization` – поле, которое представлено объектом класса `Serialization`, который отвечает за представление данных модели в различных языках разметки данных, таких как XML и JSON;

- `store` – поле, которое является объектом класса `Store`, который дает тонкую оболочку для сериализации с целью хранения хэшей в одном столбце таблицы базы данных;

- `secure_token` – поле, которое является объектом класса `SecureToken` и реализует интерфейс работы с токеном безопасности модели данных. Этот объект позволяет при необходимости сгенерировать уникальный токен безопасности и провалидировать его;

- `suppressor` – поле, являющееся объектом класса `Suppressor`, который позволяет предотвратить запись изменений модели данных при определенных условиях;

3.1.10 Класс `ApplicationRecord`

Класс представляет из себя абстрактный класс, который является наследником вышеописанного класса `ActiveRecord::Base`. Данный класс выступает родительским классом для всех классов моделей веб-приложения. Он описывает общее поведение для моделей и позволяет расширять их возможности одновременно.

Поля:

- `abstract_class` – поле типа `Boolean`, которое является флагом, определяющим, что класс является абстрактным;

3.1.11 Класс `Ability`

Класс представляет из себя набор полей и методов отвечающих за права доступа различных моделей данных к различным ресурсам веб-приложения. Благодаря этому классу можно реализовать различные права доступа для одной и той же модели в зависимости от ее роли.

Поля:

– `user` – объект класса `User`, являющийся моделью данных, для которой необходимо описать права доступа к различным ресурсам веб-приложения;

Методы:

1. `can(action)` – метод, который определяет права доступа к ресурсу веб приложения в зависимости от действия модели данных, которая представлена в аргументе `action`.

3.1.12 Класс `Image`

Этот класс является моделью данных изображения, приходящего с камеры, а также данных, связанных с ним. Данный класс агрегируется классом `User`, таким образом, при удалении пользователя, все изображения, которые к нему относятся так же удаляются. Класс также предоставляет унаследованные методы для работы с данными;

Поля:

– `id` – поле типа `BigInt`, которое хранит в себе идентификационный номер объекта класса `Image`;

– `title` – поле типа `String`, которое хранит в себе название приходящего изображения, оно зависит от настроек прошивки аппаратной части;

– `photo` – поле типа `String`, которое хранит адрес изображения в директории-хранилище, куда оно загружается. Такой подход является более рациональным, нежели использование типа `BLOB` в таблице базы данных. Благодаря этому транзакции в базу данных проходят быстрее, так как нет необходимости подгружать большой бинарный объект.

– `created_at` – поле типа `DateTime`, которое хранит в себе дату и время создания объекта класса `Image`;

– `updated_at` – поле типа `DateTime`, которое хранит в себе дату и время последнего обновления данных объекта класса `Image`;

– `user_id` – поле типа `BigInt`, которое ссылает на идентификационный номер пользователя с устройства которого пришло изображение;

– `uploader` – поле, которое является объектом класса `PhotoUploader`. Который отвечает за сохранение изображения, пришедшего в запросе, в хранилище;

3.1.13 Класс `ImagesController`

Класс является контроллером, который обрабатывает запросы, направленные на работу с данными модели `Image`.

Поля:

– `image` – объект класса `Image`, представляющий из себя привязку данных к контроллеру, согласно архитектуре MVC;

– `images` – поле типа `Array`, являющееся контейнером для объектов класса `Image`.

Методы:

1. `index()` – метод, который обрабатывает запрос на возврат всех объектов класса `Image`. Таким образом он возвращает объект класса `Array`, который содержит в себе все объекты класса `Image`, принадлежащие пользователю, который произвел запрос.

2. `show()` – метод, который обрабатывает запрос на возврат конкретного объекта класса `Image`. Он возвращает принадлежащий пользователю объект в соответствии с идентификационным номером, указанным в запросе, который произвел пользователь.

3. `new()` – метод, который обрабатывает запрос пользователя на создание объекта класса `Image` с валидацией входных параметров, посредством вызова метода `image_params()`. Для непосредственного сохранения нового объекта, после прохождения валидации, используется метод `create()`.

4. `edit()` – метод, который обрабатывает запрос пользователя на изменение данных объекта класса `Image` с указанным в запросе идентификационным номером. Также, как и вышеописанный метод, проводит валидацию входных параметров посредством вызова метода `image_params()`. Для непосредственного сохранения измененных данных объекта, после прохождения валидации, используется метод `update()`.

5. `create()` – метод, который создает объект класса `Image`. Непосредственно само изображение сохраняется в хранилище в методе `set_image()`, который вызывается в данном методе, перед записью нового объекта класса `Image` в соответствующую таблицу базы данных.

6. `update()` – метод, который изменяет объект класса `Image`. Если изменяется само изображение, то новое сохраняется в хранилище в методе `set_image()`, который вызывается в данном методе, перед записью измененного объекта класса `Image` в соответствующую таблицу базы данных.

7. `destroy()` – метод, который обрабатывает запрос на удаление объекта класса `Image` из соответствующей таблицы базы данных по идентификационному номеру объекта, указанному во входных параметрах запроса, совершенного пользователем.

8. `set_image()` – метод, который непосредственно сохраняет полученное изображение в хранилище из запроса пришедшего от пользователя. Метод делает это с помощью методов предоставленных классом `PhotoUploader`.

9. `image_params()` – метод, который валидирует приходящие в запросе параметры, убирая избыточные и опасные параметры из запроса. Так же он прервет обработку запроса, если в нем не содержатся обязательные параметры, либо их значения недопустимы.