1. 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).

* 1. Классы веб-приложения
     1. Класс User

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.

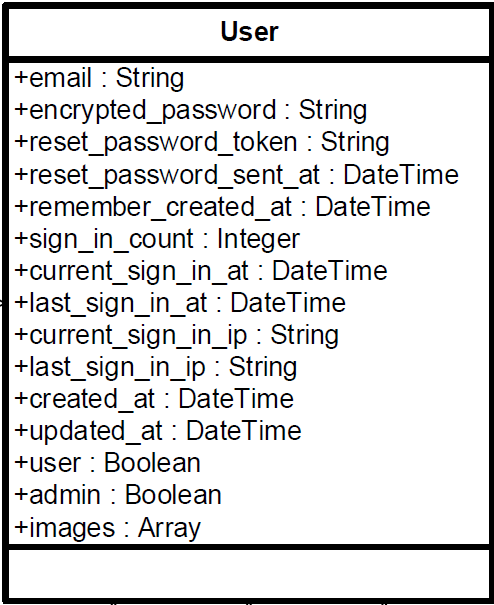


Рисунок 3.1- UML-диаграмма класса User

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;

* + 1. 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. recource=(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 i18n. 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 попытки 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 аттрибутов объекта;

– readonly\_attributes – поле, которое является объектом класса ReadonlyAttributes и предоставляет механизмы для установки аттрибутов объекта в режим «только чтение»;

– model\_schema – объект класса ModelSchema, который обеспечивает привязку аттрибутов 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 приходящих 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 аттрибутов 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 аттрибутов 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 аттрибутами 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 безопасноти 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 поведения 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. Является дочерним классом ApplicationController.

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, которой 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, которой 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, указному 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.

**3.1.14** Класс HomeController

Класс является контроллером, который отвечает за домашнюю страницу веб-приложения. Класс является дочерним классом ApplciationController.

Методы:

1. index() – метод, который отображает домашнюю страницу веб-приложения. Для неавторизованного пользователя при входе в веб-приложение будет выведено приглашение зарегистрироваться или авторизоваться в системе. Для авторизованного пользователя сразу же произойдет перенаправление на страницу, где он сможет посмотреть все изображения, пришедшие с устройства.

**3.1.15** Класс ApplicationController

Класс является контроллером, который является родительским классом для всех контроллеров данного веб-приложения и описывает их общее поведение.

Методы:

1. configure\_permitted\_parameters() – метод, который позволяет дочерним контроллерам использовать собственные методы для валидации входных параметров приходящих запросов.

**3.1.16** Класс ActionController::Base

Класс, являющийся классом-родителем для всех классов-контроллеров в веб-приложении. Является ядром веб-запроса Rails. Они состоят из одного или нескольких действий, которые выполняются по запросу, после чего, либо отображается шаблон, либо перенаправляется на другое действие. Предоставляет один класс для настройки таких функций, как защита от поддельных запросов и фильтрация параметров запроса. Класс имеет также нижеперечисленные поля и методы.

Поля:

– modules – поле типа Array, являющееся массивом автоматически подключенных модулей. Данное поле хранит в себе все модули, подключенные в приложении.

Методы:

1. without\_modules(modules: Array) – метод, проверяет подключены ли модули в приложении и возвращается true, если они не подключены, в противном случае false.

2. make\_response!(request: String) – метод используется для формирования ответа на приходящий запрос в контроллер.

**3.1.17** Класс CarrierWave::Uploader::Base

Класс, который предоставляет возможности легко обрабатывать кэширование и хранение загруженных файлов. Является родительским классом для класса PhotoUploader. Класс имеет поля.

Поля:

– configuration – поле, которое является объектом класса CarrierWave::Uploader::Configuration, который предоставляет механизм для конфигурации загрузчика;

– callbacks – поле, которое является объектом класса CarrierWave::Uploader::Callbacks, который предоставляет механизмы для работы с колбэками;

– proxy – поле, которое является объектом класса CarrierWave::Uploader::Proxy, который предоставляет механизмы для работы с прокси;

– url – поле, которое является объектом класса CarrierWave::Uploader::Url, который предоставляет механизмы для работы с URL;

– mountable – поле, которое является объектом класса CarrierWave::Uploader::Mountable, который позволяет монтировать загрузчик к модели;

– cache – поле, которое является объектом класса CarrierWave::Uploader::Cache, который предоставляет механизмы для работы с хранящимися в кэше файлами;

– url – поле, которое является объектом класса CarrierWave::Uploader::Url, который предоставляет механизмы для работы с URL;

– url – поле, которое является объектом класса CarrierWave::Uploader::Url, который предоставляет механизмы для работы с URL;

Методы:

1. public static ConnectivtyUtils getInstance() – статический метод получения сущности класса ConnectivtyUtils.

2. public ConnectivityReceiver getConnectReceiver(final ConnectivityInterface connectListener) – метод получения широковещательного приемника, отслеживающего состояние соединения. В качестве входного параметра принимает переменную типа ConnectivityInterface, представляющую собой метод обратного вызова, который срабатывает при изменении состояния соединения.

3. public static boolean isConnected(final Context context) – метод возвращает текущее состояние соединения. В качестве входного параметра принимает переменную типа Context, представляющую собой контекст.

**3.1.18** Класс EncodeUtil

Класс содержит в себе статический метод дешифрования из формата Base64.

Методы:

1. public static byte[] decodeBase64(String encoded) – метод преобразования зашифрованного текста в массив байт. В качестве входного параметра принимает переменную типа String, представляющую собой текст для дешифрования.

**3.1.19** Класс CharonVpnService

Класс представляет из себя реализацию сервиса, который управляет VPN соединением.

Поля:

– VPN\_STATE\_NOTIFICATION\_ID – константное статическое поле типа int, являющееся идентификатором уведомления данного сервиса;

– mConnectionHandler – поле типа Thread, являющееся потоком, в котором выполняется вся работа VPN;

– mCurrentProfile – поле типа VpnProfile, являющееся объектом текущего VPN профиля;

– mCurrentCertificateAlias – поле типа String, являющееся именем текущего сертификата;

– mCurrentUserCertificateAlias – поле типа String, являющееся именем текущего сертификата пользователя;

– mNextProfile – поле типа VpnProfile, являющееся объектом следующего VPN профиля, который необходимо запустить;

– mProfileUpdated – поле типа boolean, являющееся флагом того, что профиль обновился;

– mTerminate – поле типа boolean, являющееся флагом того, что соединение завершено;

– mIsDisconnecting – поле типа boolean, являющееся флагом того, что соединение разрывается;

– mShowNorification – поле типа boolean, являющееся флагом того, что уведомление отображается;

– mService – поле типа VpnStateService, являющееся объектом сервиса, отслеживающего состояние VPN соединения;

– mServiceConnection – поле типа ServiceConnection, являющееся сервисом, отслеживающим состояние текущего сервиса;

– mCurrentProfile – поле типа VpnProfile, являющееся объектом текущего VPN профиля.

Методы:

1. public void onCreate() – метод жизненного цикла сервиса. Вызывается при создании сервиса.

2. public void onRevoke() – метод отключение сервиса. Вызывается, когда пользователь отключает VPN через настройки.

3. public void onDestroy() – метод жизненного цикла сервиса. Вызывается при завершении сервиса.

4. public void setNextProfile(VpnProfile profile) – метод устанавливает профиль, который будет инициализирован. Профиль передается в качестве параметра profile.

5. public void run() – метод, в котором происходит основная работа VPN. Вызывается в потоке mConnectionHandler.

6. private void stopCurrentConnection() – метод останавливает текущее VPN соединение.

8. private void addNotification() – метод добавляет уведомление с состоянием VPN соединения.

9. private void removeNotification() – метод удаляет уведомление с состоянием VPN соединения.

10. private Notification buildNotification(boolean publicVersion) – метод создает уведомление с текущим состоянием VPN. В качестве параметра принимает переменную publicVersion, которая является флагом того, необходимо ли отображать содержимое на заблокированном экране.

11. private void startConnection(VpnProfile profile) – метод уведомляет сервис, который отслеживает состояние VPN, о том, что создается новое соединение. В качестве параметра принимает переменную profile, которая является активным VPN профилем.

12. private void setState(State state) – метод обновляет текущее состояние в сервисе, который отслеживает состояние VPN. В качестве параметра принимает переменную state, которая является текущим состоянием.

13. private void setError(ErrorState error) – метод обновляет текущее состояние ошибки в сервисе, который отслеживает состояние VPN. В качестве параметра принимает переменную error, которая является текущим состоянием ошибки.

**3.1.20** Класс TrustedCertificateManager

В данном классе реализован функционал по извлечению всех сертификатов или из конкретного хранилища.

Поля:

– mLock – константное поле типа ReentrantReadWriteLock, являющееся защелкой при чтении и записи;

– mCACerts – поле типа HashTable<String, X509Certificate>, являющееся хэш-таблицей, в которой хранятся сертификаты;

– mReload – поле типа boolean, являющееся флагом того, что была вызвана перезагрузка сертификатов;

– mLoaded – поле типа boolean, являющееся флагом того, что сертификаты были загружены;

– mKeyStores – поле типа ArrayList<KeyStore>, являющееся списком источников для извлечения сертификатов.

Методы:

1. public static TrustedCertificateManager getInstance() – метод возвращает сущность класса TrustedCertificateManager.

2. public TrustedCertificateManager reset() – метод перезагружает кэшированные сертификаты.

3. public TrustedCertificateManager load() – метод проверяет загружены ли сертификаты. Если не загружены, то начинается загрузка сертификатов.

4. private void loadCertificates() – метод открывает хранилище сертификатов и начинает загрузку кэшированных сертификатов.

5. private void fetchCertificates(Hashtable<String, X509Certificate> certs, KeyStore store) – метод загружает все сертификаты из хранилища. Хранилище, из которого необходим выгружать, передается в качестве параметра store. Сертификаты загружаются в хэш-таблицу, которая передается в качестве параметра certs.

6. private X509Certificate getX509CertificateFromAlias(String alias) – метод возвращает сертификат по имени, которое передается в качестве параметра alias.

7. private Hashtable<String, X509Certificate> getAllCertificates() – метод возвращает все сертификаты из всех хранилищ.

Перечисления:

– TrustedCertificateSource – перечисление источников доверенных сертификатов.

**3.1.21** Класс VpnStateService

Класс представляет из себя реализацию сервиса, который отслеживает состояние VPN соединения.

Поля:

– mListeners – поле типа HashSet<VpnStateListener>, являющееся хэш-набором, который хранит всех слушателей, отслеживающих состояние VPN соединения;

– mConnectionID – поле типа long, являющееся идентификатором текущего VPN соединения;

– mHandler – поле типа Handler, являющееся объектом, который будет оповещать всех слушателей о том, что состояние VPN соединения изменилось;

– mProfile – поле типа VpnProfile, являющееся флагом того, что сертификаты были загружены;

– mState – поле типа State, являющееся текущим состоянием VPN соединения;

– mError – поле типа ErrorState, являющееся текущим состоянием ошибки VPN соединения.

Методы:

1. public void onCreate() – метод жизненного цикла сервиса. Вызывается при создании сервиса.

2. public void onDestroy() – метод жизненного цикла сервиса.

Вызывается при завершении сервиса.

3. public void registerListener(VpnStateListener listener) – метод регистрирует нового слушателя, отслеживающего изменения состояния VPN соединения. Новый слушатель передается в качестве параметра listener.

4. public void unRegisterListener(VpnStateListener listener) – метод отменяет регистрацию слушателя, который передается в качестве параметра listener.

5. public VpnProfile getProfile() – метод возвращает текущий VPN профиль.

6. public State getState() – метод возвращает текущее состояние VPN соединения.

7. public ErrorState getErrorState() – метод возвращает текущее состояние ошибки VPN соединения.

8. public void disconnect() – метод останавливает текущее соединение. При этом VPN сервис не останавливается, а сбрасывается. Таким образом новое соединение может быть добавлено.

9. public void notifyListeners(final Callable<Boolean> change) – метод уведомляет всех слушателей о том, что состояние VPN соединения изменилось. Если действие, которое передается в качестве параметра change, не удалось выполнить, то слушатели уведомляться не будут.

10. public void startConnection(final VpnProfile profile) – метод вызывается когда создается новое соединение. Устанавливается новый профиль, который передается в качестве параметра profile. Также сбрасывается состояние ошибки VPN соединения, состояние устанавливается в значение CONNECTING, увеличивается идентификатор текущего VPN соединения и оповещаются все слушатели.

11. public void setState(final State state) – метод устанавливает текущее состояние VPN соединения. Состояние, в которое необходимо установить, передается в качестве параметра state.

12. public void setErrorState(final ErrorState error) – метод устанавливает текущее состояние ошибки VPN соединения. Состояние ошибки, в которое необходимо установить, передается в качестве параметра error.

Перечисления:

– State – перечисление состояний VPN соединения;

– ErrorState – перечисление состояний ошибки VPN соединения.

**3.1.22** Перечисление VpnType

Перечисление представляет из себя набор типов VPN. Каждый VPN типа характеризуется идентификатором и набором функций, которые передаются в качестве параметров для каждого типа.

Поля:

– IKEV2\_EAP;

– IKEV2\_CERT;

– IKEV2\_CERT\_EAP;

– IKEV2\_EAP\_TLS;

– IKEV2\_BYOD\_EAP.

Методы:

1. public String getIdentifier() – метод возвращает идентификатор VPN типа.

2. public boolean has(VpnTypeFeature feature) – метод проверяет имеет ли данный VPN типа функцию. Функция передается в качестве параметра feature.

3. public VpnType fromIdentifier(String identifier) – статический метод получения VPN типа по идентификатору. В качестве параметра передается переменная identifier, по которой необходимо определить VPN тип.

Перечисления:

– VpnTypeFeature – перечисление функций для VPN типа.