

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ПРОГРАММНАЯ БИБЛИОТЕКА ПОДПРОГРАММ ИДЕНТИФИКАЦИИ
ПОЛЬЗОВАТЕЛЕЙ

Руководство системного программиста

Листов 21

Студент гр. ИУ6-84Б

(Подпись, дата)

М.А. Гейне
(И.О.Фамилия)

Руководитель

(Подпись, дата)

П. В. Аргентов
(И.О.Фамилия)

Москва, 2022

Содержание

1	FPauth - библиотека для лёгкой, но настраиваемой аутентификации в веб-приложениях Dream.	3
1.1	Начало работы	3
1.2	Продвинутое использование	4
1.3	Модуль FPauth	5
1.4	Модуль FPauth.Make_Auth	5
1.4.1	Параметры	5
1.4.2	Сигнатура	6
1.5	Модуль Make_Auth.Variables	6
1.6	Модуль Make_Auth.Session_manager	7
1.7	Модуль Make_Auth.Authenticator	8
1.8	Модуль Make_Auth.Router	8
2	FPauth-core - основа системы аутентификации	10
2.1	Модуль FPauth_core	10
2.2	Модуль FPauth_core.Static	10
2.3	Модуль FPauth_core.Auth_sign	12
3	FPauth-strategies - стратегии аутентификации	17
3.1	Модуль FPauth_strategies	17
3.2	Модуль FPauth_strategies.Password	17
3.3	Модуль FPauth_strategies.TOTP	17
4	FPauth-responses - шаблоны ответов	20
4.1	Модуль FPauth_responses	20
5	Полная версия	21

1 FPruth - библиотека для лёгкой, но настраиваемой аутентификации в веб-приложениях Dream.

FPruth - лёгкая система аутентификации для OCaml Dream¹ веб-фреймворка.

Главная идея системы заключается в том, что аутентификация проводится посредством запуска наборов Стратегий, и, когда одна из них завершается успешно, пользователь считается аутентифицированным. Статус аутентификации контролируется middleware, вызываемым после middleware сессии.

Система позволяет:

- Контролировать аутентификацию в веб-сессии;
- Получать статус аутентификации для каждого запроса через `Dream.field`;
- Проверять личность пользователя стратегиями;
- Использовать стратегии в составе библиотеки или сторонние;
- Добавлять все маршруты для аутентификации и стратегий одновременно;
- Добавлять собственные представления событий аутентификации или использовать встроенные в библиотеку;
- Использовать встроенные в библиотеку обработчики запросов или собственные;
- Извлекать параметры для аутентификации из запросов.

1.1 Начало работы

Чтобы начать использовать библиотеку FPruth в своём проекте, необходимо:

- Установить её. К примеру, воспользоваться командой `opam install FPruth`.
- Подключить библиотеку к системе сборки. Например, при использовании dune, необходимо в dune-файле указать:

```
(libraries FPruth)
```

- Инициализировать систему с моделью пользователя, которая удовлетворяет сигнатуре `FPruth.Auth_sign.MODEL`. Эта сигнатура требует функций, которые определяют, как сохранять пользователей в сессии и восстанавливать их из неё (`serialize` и `deserialize`), как находить пользователей по параметрам запросов (`identify`) и какие стратегии могут быть применены к пользователю (`applicable_strats`);

¹<https://github.com/aantron/dream>

```
module Auth = FPauth.Make_Auth (User)
```

- Инициализировать стратегии, которые будут использоваться для верификации личностей пользователей. В `FPauth.Strategies` имеется несколько стратегий. `Password` используется для парольной аутентификации, пароли должны быть предварительно зашифрованы с помощью `Argon2`. `'TOTP'` является стратегия проверки по одноразовым паролям на основе времени. Содержит маршруты для нацепки стратегии для заранее аутентифицированного пользователя. Стратегии могут иметь дополнительные требования к моделям пользователей, а также зависеть от других модулей.

```
module Password = FPauth_strategies.Password.Make (User)
```

- Добавить `'Session_manager'` middleware после middleware сессии;

```
let () = run
  @@ memory_sessions
  @@ Auth.Session_manager.auth_setup
```

- вставить маршруты `FPauth` в `Dream.router middleware`. Здесь задаются стратегии, которые будут использоваться для аутентификации; способ извлечения параметров из запросов; ответы на основные события аутентификации. Также можно определить область видимости маршрутов аутентификации.

```
@@ router [
  Auth.Router.call [(module Password)] ~responses:(module Responses)
    ~extractor:extractor ~scope:"/authentication"
]
```

Стратегии и Ответы подаются в виде модулей первого класса, которые должны удовлетворять требованиям сигнатур `FPauth.Auth_sign.STRATEGY`. * In `'FPauth_responses'` you can find some default responses in JSON and HTML format; * In `'FPauth.Static.Params'` you can find some default extractors from JSON-requests' bodies, forms or from query;

- Готово! Теперь веб-приложение может аутентифицировать пользователей.

1.2 Продвинутое использование

Можно настроить многие аспекты работы системы аутентификации. К примеру:

- Можно установить только необходимые пакеты:
 - `FPauth-core` содержит `Session_manager`, `Authenticator`, `Router`, `Variables`, а также статичный модуль `Static` и сигнатуры. Это всё позволяет выстроить собственную систему аутентификацию почти с нуля;

- FPAuth-strategies содержит Password and OTP стратегии. Если они не нужны - их можно не устанавливать;
 - FPAuth-responses содержит некоторые простые ответы на основные события аутентификации;
- Можно написать собственные стратегии, ответы и экстракторы параметров.

Подробная документация в разделе FPAuth.

1.3 Модуль FPAuth

module Static = FPAuth_core.Static

Static - модуль, содержащий в себе определения статичных типов, которые не зависят от Auth_sign.MODEL.

module Make_Auth (M : FPAuth_core.Auth_sign.MODEL) : **sig** ... **end**

Make_Auth создаёт модуль аутентификации на основе Auth_sign.MODEL. Предоставляет локальные переменные (field), middleware, аутентификатор, запускающий стратегии аутентификации, а также маршрутизатор, добавляющий в приложение маршруты аутентификации.

module Auth_sign = FPAuth_core.Auth_sign

Auth_sign - модуль, содержащий сигнатуры других модулей, которые могут быть реализованы и интегрированы извне библиотеки, а также сигнатуры некоторых внутренних модулей.

module Strategies = FPAuth_strategies

Strategies содержит две стандартные стратегии: FPAuth_strategies.Password.

module Responses = FPAuth_responses

Responses содержит стандартные шаблоны ответов на базовые события аутентификации.

1.4 Модуль FPAuth.Make_Auth

Make_Auth создаёт модуль аутентификации на основе Auth_sign.MODEL. Предоставляет локальные переменные (field), middleware, аутентификатор, запускающий стратегии аутентификации, а также маршрутизатор, добавляющий в приложение маршруты аутентификации.

1.4.1 Параметры

module M : **sig**

type t

Некоторое представление сущности, которая будет аутентифицирована.

```

val serialize : t → Base.string
    serialize ent создаёт string из t.

val deserialize : Base.string → ( t, Base.Error.t ) Base.Result.t
    deserialize создаёт t. Возвращает: Ok t если десериализация была успешна или
    Error string если произошла ошибка.

val identificate : Dream.request → ( t, Base.Error.t ) Base.Result.t Dream.promise
    identificate определяет, какая именно сущность аутентифицируется. Находит
    репрезентацию сущности или возвращает ошибку.

val applicable_strats : t → Base.string Base.list
    applicable_strats возвращает список стратегий, которые могут быть приме-
    нены ко всей MODEL или к определённой сущности t. Строки должны совпадать с
    STRATEGY.name.

end

```

1.4.2 Сигнатура

```

module Variables : FPauth_core.Auth_sign.VARIABLES with type entity
= M.t
    Variables содержит типы, функции и fields, основанные на Auth_sign.MODEL.

module Session_manager : FPauth_core.Auth_sign.SESSIONMANAGER with
type entity = M.t
    SessionManager - модуль, который устанавливает fields из сессии для каждого за-
    проса через Auth_sign.SESSIONMANAGER.auth_setup middleware.

module Authenticator : FPauth_core.Auth_sign.AUTHENTICATOR with type
entity = M.t
    Authenticator содержит функции для исполнения списка Auth_sign.STRATEGY и
    сброса аутентификации.

module Router : FPauth_core.Auth_sign.ROUTER with type entity = M.t
    Router создаёт маршруты, необходимые для аутентификации. Содержит несколько ба-
    зовых handlers и объединяет их с маршрутами из стратегий Auth_sign.STRATEGY.
    routes.

```

1.5 Модуль Make_Auth.Variables

```

    Variables содержит типы, функции и fields, основанные на Auth_sign.MODEL.
type entity = M.t

```

тип `entity` - тип аутентифицируемой сущности, совпадающий с `MODEL.t`.

val `authenticated` : `Base.bool Dream.field`

`authenticated` - переменная, действительная в рамках одного запроса, отражает, была ли пройдена аутентификация ранее. Устанавливается в `SESSIONMANAGER.auth_setup`.

val `current_user` : `entity Dream.field`

`current_user` - переменная, действительная в рамках одного запроса, содержит аутентифицированную сущность (если ранее была пройдена аутентификация). Устанавливается в `SESSIONMANAGER.auth_setup`

val `auth_error` : `Base.Error.t Dream.field`

`auth_error` - `field`-переменная с ошибкой, которая могла произойти на любом этапе аутентификации. Устанавливается в `AUTHENTICATOR.authenticate`.

val `update_current_user` : `entity → Dream.request → Base.unit Dream.promise`

`update_current_user user request` обновляет `current_user` и сессию. Необходимо использовать в том случае, если были внесены изменения, влияющие на сериализацию.

1.6 Модуль `Make_Auth.Session_manager`

`SessionManager` - модуль, который устанавливает `fields` из сессии для каждого запроса через `Auth_sign.SESSIONMANAGER.auth_setup middleware`.

type `entity` = `M.t`

тип `entity` - тип аутентифицируемой сущности, совпадающий с `MODEL.t`

val `auth_setup` : `Dream.middleware`

`auth_setup` - `middleware`, которое контролирует сессию, устанавливает переменные-`field` и вспомогательные функции для последующих `handlers`. `auth_setup` пробует извлечь строку из сессии с ключом `auth` и определить статус аутентификации. Если поле `auth` отсутствует, то аутентификация не была пройдена. Если `auth` имеется, то по строке проверяется и изменяется статус аутентификации:

- Если `auth` содержит пустую строку, то ситуация считается ошибочной;
- Если `M.deserialize` вернула `Error Error.t`, то аутентификация не пройдена и ситуация считается ошибочной;
- Если `M.deserialize` вернула `Ok M.t`, то аутентификация считается успешной и устанавливаются `VARIABLES.current_user` и `VARIABLES.authenticated`. Если с сессией что-то не так, то сессия становится недействительной, ошибка записывается и статус 401 отправляется. Если сессия в порядке, то запрос поступает в следующий обработчик.

1.7 Модуль `Make_Auth.Authenticator`

`Authenticator` содержит функции для исполнения списка `Auth_sign.STRATEGY` и сброса аутентификации.

type `entity` = `M.t`

тип `entity` - тип аутентифицируемой сущности, совпадающий с `MODEL.t`.

type `strategy` = (**module** `FPauth_core.Auth_sign.STRATEGY` **with type** `entity` = `entity`)

`strategy` - модуль первого класса стратегии для `entity`.

val `authenticate` : `strategy Base.list` → `Dream.request` → `FPauth_core.Static.AuthResult.t` `Dream.promise`

`authenticate` запускается множество стратегий для запроса и определяет, была ли аутентификация в целом успешной или нет.

val `logout` : `Dream.request` → `Base.unit` `Lwt.t`

`logout` сбрасывает сессию, что приводит к сбросу статуса аутентификации. В связи с особенностями работы `field` текущий пользователь будет сброшен только в следующем запросе.

1.8 Модуль `Make_Auth.Router`

`Router` создаёт маршруты, необходимые для аутентификации. Содержит несколько базовых `handlers` и объединяет их с маршрутами из стратегий `Auth_sign.STRATEGY.routes`.

type `entity` = `M.t`

тип `entity` - тип аутентифицируемой сущности, совпадающий с `MODEL.t`.

type `strategy` = (**module** `FPauth_core.Auth_sign.STRATEGY` **with type** `entity` = `entity`)

`strategy` - модуль первого класса стратегии для `entity`.

val `login_handler` : `strategy Base.list` → (**module** `FPauth_core.Auth_sign.RESPONSES`) → `Dream.request` → `Dream.response` `Lwt.t`

`login_handler` получается список стратегий и шаблоны ответов, запускает аутентификацию и обрабатывает её результаты.

val `logout_handler` : (**module** `FPauth_core.Auth_sign.RESPONSES`) → `Dream.request` → `Dream.response` `Lwt.t`

`logout_handler` сбрасывает аутентификацию для текущего пользователя.

val `call` : `?root:Base.string` → `responses:(module FPauth_core.Auth_sign.RESPONSES)` → `extractor:FPauth_core.Static.Params.extractor` → `strategy Base.list` → `Dream.route`

`call ?root ~responses ~extractor strat_list` создаёт маршруты для аутентификации, которые добавляются в `Dream.router`. Содержит следующие базовые маршруты:

- `"/auth"` является стартовой точкой для аутентификации. Передаёт `strategies` в `Auth_sign.AUTHENTICATOR.authenticate`.
- `"/logout"` выполняет сброс аутентификации с помощью `Authenticator.logout`.

`extractor` определяет способ извлечения параметров из запросов для всех запросов, связанных с аутентификацией, в том числе поступающих по маршрутам `STRATEGY.routes`. Подробнее в `Static.Params.extractor.responses` определяет, какие ответы отправлять для запросов, поступающих по базовым маршрутам. `?root` определяет корневой путь для всех маршрутов, связанных с аутентификацией. По умолчанию `"/"`.

2 FPauth-core - основа системы аутентификации

В данном пакете содержатся основные функции библиотеки, реализующие систему аутентификации. Подробнее в разделе FPauth_core.

2.1 Модуль FPauth_core

module Static : **sig** ... **end**

Static - модуль, содержащий в себе определения статичных типов, которые не зависят от Auth_sign.MODEL.

module Variables : **sig** ... **end**

VARIABLES - модуль, инициализирующий и содержащий field-переменные для аутентификации.

module Session_manager : **sig** ... **end**

Session_manager содержит middleware для контроля сессии веб-приложения.

module Authenticator : **sig** ... **end**

Authenticator - модуль, предоставляющий функции для аутентификации и её сброса.

module Router : **sig** ... **end**

Router - модуль, который содержит handlers для аутентификации и создаёт маршруты к ним.

module Make_Auth (M : Auth_sign.MODEL) : **sig** ... **end**

Make_Auth создаёт модуль аутентификации на основе Auth_sign.MODEL. Предоставляет локальные переменные (field), middleware, аутентификатор, запускающий стратегии аутентификации, а также маршрутизатор, добавляющий в приложение маршруты аутентификации.

module Auth_sign : **sig** ... **end**

Auth_sign - модуль, содержащий сигнатуры других модулей, которые могут быть реализованы и интегрированы извне библиотеки, а также сигнатуры некоторых внутренних модулей.

2.2 Модуль FPauth_core.Static

Static - модуль, содержащий в себе определения статичных типов, которые не зависят от Auth_sign.MODEL.

Static - содуль, содержащий все возможности библиотеки, которые не зависят от FPauth.Auth_sign.MODEL

module StratResult : **sig**

```

type 'a t =
  | Authenticated of 'a

```

Сущность была успешно аутентифицирована. Также может быть использована внутри стратегий с функцией `bind` аналогично `Ok 'a` результату. Когда возвращается в `FPauth.Auth_sign.AUTHENTICATOR`, завершает процесс аутентификации.

```

  | Rescue of Base.Error.t

```

Аутентификация должна быть немедленно остановлена с ошибкой.

```

  | Redirect of Dream.
    response Lwt.t

```

Пользователь должен быть перенаправлен в соответствии с `response.response promise` создаётся с помощью `Dream.redirect`.

```

  | Next

```

Следующая стратегия из списка в `FPauth.Auth_sign.AUTHENTICATOR` должна быть исполнена.

'a t определяет результаты стратегий.

```

val bind : 'a t → ( 'a → 'b t ) → 'b t

```

`bind r f` возвращает `f r` если `r` является `Authenticated` или `r` в иных случаях.

```

module Infix : sig

```

```

  val (»==) : 'a t → ( 'a → 'b t ) → 'b t

```

Инфиксный оператор для `FPauth.Static.StratResult.bind`

```

end

```

Модуль с операторами в инфиксной форме для `StratResult`

```

end

```

`StratResult` определяет результат стратегий, а также задаёт вспомогательные функции.

```

module AuthResult : sig

```

```

  type t =

```

```

    | Authenticated

```

Сущность была успешно аутентифицирована.

```

    | Rescue

```

Аутентификация завершилась с ошибкой.

```

    | Redirect of Dream.
      response Lwt.t

```

Пользователь должен быть перенаправлен в соответствии с `response.response promise` создаётся с помощью `Dream.redirect`.

```

end

```

`AuthResult` - результат всего процесса аутентификации. Похож на `StratResult`, но не содержит некоторые типы, которые имеют смысл только для стратегий. `Authenticated` и `Rescue` не содержат в себе данных, так как они сохраняются в `Dream.field` к концу аутентификации.

```

module Params : sig

```

```

type t
val params : Dream.request → t option
    params request возвращает t middleware.

type extractor = Dream.request → t Lwt.t
    extractor - тип функции, которая извлекает параметры из запросов.

val get_param : string → t → string option
    get_param key params ищет заданный ключ key в params и возвращает Some
    str, если параметр был найден, или None в ином случае.

val get_param_exn : string → t → string
    get_param_exn key params совпадает с get_param, но возвращает исключе-
    ние в случае, если key отсутствует.

val get_param_req : string → Dream.request → string option
    get_param_req key request является сокращением params request >>=
    get_param key.

val extract_query : extractor
    extract_query request извлекает все query-параметры запроса и возвращает их
    в виде t.

val extract_json : extractor
    extract_json request извлекает все пары ключей-значений из запроса в фор-
    мате JSON. Content-Type запроса должен быть application/json.

val extract_form : ?csrf:bool → extractor
    extract_form request извлекает параметры из форм, отправленных с Dream.
    csrf_tag. Content-Type запроса должен быть application/x-www-form-
    urlencoded.

val of_assoc : (string * string) list → t
    of_assoc lst создаёт t.

val set_params : extractor:extractor → Dream.handler → Dream.
    request → Dream.response Dream.promise
    ser_params ~extractor - middleware, которое устанавливает параметры для за-
    проса, извлекая их с помощью ~extractor.

```

end

Params хранит параметры запроса, все или только требуемые для аутентификации.

2.3 Модуль `FPauth_core.Auth_sign`

Auth_sign - модуль, содержащий сигнатуры других модулей, которые могут быть реали-
зованы и интегрированы извне библиотеки, а также сигнатуры некоторых внутренних модулей.

В этом модуле хранятся сигнатуры модулей системы аутентификации или модулей, которые необходимо реализовать извне.

```
module type MODEL = sig
```

```
  type t
```

Некоторое представление сущности, которая будет аутентифицирована.

```
val serialize : t → Base.string
```

serialize ent создаёт string из t.

```
val deserialize : Base.string → ( t, Base.Error.t ) Base.Result.t
```

deserialize создаёт t. Возвращает: Ok t если десериализация была успешна или Error string если произошла ошибка.

```
val identificate : Dream.request → ( t, Base.Error.t ) Base.Result.t Dream.promise
```

identificate определяет, какая именно сущность аутентифицируется. Находит репрезентацию сущности или возвращает ошибку.

```
val applicable_strats : t → Base.string Base.list
```

applicable_strats возвращает список стратегий, которые могут быть применены ко всей MODEL.

```
end
```

MODEL - сигнатура модулей, обслуживающих аутентифицируемые сущности.

```
module type SESSIONMANAGER = sig
```

```
  type entity
```

тип entity - тип аутентифицируемой сущности, совпадающий с MODEL.t

```
val auth_setup : Dream.middleware
```

auth_setup - middleware, которое контролирует сессию, устанавливает переменные field и вспомогательные функции для последующих handlers. auth_setup пробует извлечь строку из сессии с ключом auth и определить статус аутентификации. Если поле auth отсутствует, то аутентификация не была пройдена. Если auth имеется, то по строке проверяется и изменяется статус аутентификации:

- Если auth содержит пустую строку, то ситуация считается ошибочной;
- Если M.deserialize вернула Error Error.t, то аутентификация не пройдена и ситуация считается ошибочной;
- Если M.deserialize вернула Ok M.t, то аутентификация считается успешной и устанавливаются VARIABLES.current_user. Если с сессией что-то не так, то сессия становится недействительной, ошибка записывается и статус 401 отправляется. Если сессия в порядке, то запрос поступает в следующий обработчик.

```
end
```

SESSIONMANAGER - сигнатура для функтора, создающего модуль контроллера сессий для аутентификации сущностей типа MODEL.t.

```
module type STRATEGY = sig
```

```
  type entity
```

тип entity - тип аутентифицируемой сущности, совпадающий с MODEL.t

```
  val call : Dream.request → entity → entity Static.StratResult.  
  t Lwt.t
```

call request entity является основной функцией стратегии. Является непосредственным методом подтверждения личности.

```
  val routes : Dream.route
```

routes определяет дополнительные маршруты к handlers стратегии, если они необходимы. Может включать несколько маршрутов, если применить Dream.scope.

```
  val name : Base.string
```

name - имя стратегии. Используется для определения, может ли стратегия быть применена к определённой сущности.

```
end
```

STRATEGY - модуль, который содержит функции для аутентификации сущности определённым способом, а также дополнительные маршруты и функции.

```
module type AUTHENTICATOR = sig
```

```
  type entity
```

тип entity - тип аутентифицируемой сущности, совпадающий с MODEL.t.

```
  type strategy = (module STRATEGY with type entity = entity)
```

strategy - модуль первого класса стратегии для entity.

```
  val authenticate : strategy Base.list → Dream.request → Static.  
  AuthResult.t Dream.promise
```

authenticate запускается множество стратегий для запроса и определяет, была ли аутентификация в целом успешной или нет.

```
  val logout : Dream.request → Base.unit Lwt.t
```

logout сбрасывает сессию, что приводит к сбросу статуса аутентификации. В связи с особенностями работы field текущий пользователь будет сброшен только в следующем запросе.

```
end
```

AUTHENTICATOR - сигнатура для функтора, создающего аутентификаторы для различных сущностей.

```
module type VARIABLES = sig
```

```
  type entity
```

тип entity - тип аутентифицируемой сущности, совпадающий с MODEL.t.

```

val authenticated : Base.bool Dream.field
    authenticated - переменная, действительная в рамках одного запроса, отражает, была ли пройдена аутентификация ранее. Устанавливается в SESSIONMANAGER.auth_setup.

val current_user : entity Dream.field
    current_user - переменная, действительная в рамках одного запроса, содержит аутентифицированную сущность (если ранее была пройдена аутентификация). Устанавливается в SESSIONMANAGER.auth_setup

val auth_error : Base.Error.t Dream.field
    auth_error - field-переменная с ошибкой, которая могла произойти на любом этапе аутентификации. Устанавливается в AUTHENTICATOR.authenticate.

val update_current_user : entity → Dream.request → Base.unit
    Dream.promise
    update_current_user user request обновляет current_user и сессию. Необходимо использовать в том случае, если были внесены изменения, влияющие на сериализацию.

```

end

VARIABLES - сигнатура модуля, содержащего field-переменные, основанные на MODEL.

module type RESPONSES = **sig**

```

val login_successful : Dream.request → Dream.response Dream.promise
    login_successful вызывается в случае, если аутентификация была успешна.

val login_error : Dream.request → Dream.response Dream.promise
    login_error вызывается в случае, если в рамках аутентификации произошла ошибка.

val logout : Dream.request → Dream.response Dream.promise
    logout вызывается после того, как аутентификация была сброшена.

```

end

RESPONSES - сигнатура модуля, определяющего способ представления базовых событий библиотеки.

module type ROUTER = **sig**

```

type entity
    тип entity - тип аутентифицируемой сущности, совпадающий с MODEL.t.

type strategy = (module STRATEGY with type entity = entity)
    strategy - модуль первого класса стратегии для entity.

val login_handler : strategy Base.list → (module RESPONSES) →
    Dream.request → Dream.response Lwt.t

```

login_handler получается список стратегий и шаблоны ответов, запускает аутентификацию и обрабатывает её результаты.

```
val logout_handler : (module RESPONSES) → Dream.request → Dream.  
response Lwt.t
```

logout_handler сбрасывает аутентификацию для текущего пользователя.

```
val call : ?root:Base.string → responses:(module RESPONSES) →  
extractor:Static.Params.extractor → strategy Base.list → Dream.  
route
```

call ?root ~responses ~extractor strat_list создаёт маршруты для аутентификации, которые добавляются в Dream.router. Содержит следующие базовые маршруты:

- `"/auth"` является стартовой точкой для аутентификации. Передаёт strategies в Auth_sign.AUTHENTICATOR.authenticate.
- `"/logout"` выполняет сброс аутентификации с помощью Authenticator.logout и отвечает с использованием шаблона Auth_sign.RESPONSES.logout.

extractor определяет способ извлечения параметров из запросов для всех запросов, связанных с аутентификацией, в том числе поступающих по маршрутам STRATEGY.routes.responses определяет, какие ответы отправлять для запросов, поступающих по базовым маршрутам. ?root определяет корневой путь для всех маршрутов, связанных с аутентификацией. По умолчанию `"/"`.

end

ROUTER - сигнатура модуля, который содержит handlers для аутентификации и создаёт для них маршруты.

3 FAuth-strategies - стратегии аутентификации

В данном пакете содержатся 2 стратегии аутентификации: парольная и TOTP. Подробнее в разделе FAuth_strategies.

3.1 Модуль FAuth_strategies

module Password : **sig** ... **end**

Password - простая стратегия аутентификации, которая подтверждает личность по паролю, предоставленному в параметрах запроса.

module TOTP : **sig** ... **end**

TOTP - стратегия проверки по одноразовым паролям на основе времени. Личность пользователя подтверждается паролем, действующим только в ограниченном временном промежутке.

3.2 Модуль FAuth_strategies.Password

Password - простая стратегия аутентификации, которая подтверждает личность по паролю, предоставленному в параметрах запроса.

Требует параметр «password», иначе пропускается.

val name : string

Имя стратегии.

module type MODEL = **sig**

type t

val encrypted_password : t → string option

encrypted_password - строка с хэшем пароля, на соответствие которой предоставленный пароль будет проверяться. Argon2 используется для верификации.

end

MODEL содержит требования к модели пользователя для того, чтобы использовать стратегию.

module Make (M : MODEL) : **sig** ... **end**

Make создаёт стратегию для предоставленной модели.

3.3 Модуль FAuth_strategies.TOTP

TOTP - стратегия проверки по одноразовым паролям на основе времени. Личность пользователя подтверждается паролем, действующим только в ограниченном временном промежутке.

Требуется параметр «totp_code», иначе пропускается. Предоставляет следующие маршруты в области видимости «/totp»:

- GET `"/generate_secret"` является первым шагом для включения TOTP. Генерирует секрет для пользователя. Пользователь должен быть заранее аутентифицирован. Для пользователя стратегия не должна быть предварительно настроена.
- POST `"/finish_setup"` является вторым шагом для включения TOTP. Должен получить `"totp_code"` в качестве параметра, верифицирует его и включает TOTP в случае успешной верификации.

```
val name : string
```

Имя стратегии.

```
module type MODEL = sig
```

```
  type t
```

```
  val otp_secret : t → string
```

Извлекает секрет TOTP для пользователя.

```
  val otp_enabled : t → bool
```

Проверяет, что TOTP уже настроена для пользователя. Возвращает: `true`, если пользователь может использовать TOTP.

```
  val set_otp_secret : Dream.request → t → string → t Lwt.t
```

Устанавливает TOTP секрет во время настройки. Возвращает обновлённого пользователя.

```
  val set_otp_enabled : Dream.request → t → bool → t Lwt.t
```

Включает TOTP. Возвращает обновлённого пользователя.

```
end
```

MODEL содержит требования к модели пользователя для того, чтобы использовать стратегию.

```
module type RESPONSES = sig
```

```
  val response_error : Dream.request → Base.Error.t → Dream.response Lwt.t
```

Этот шаблон используется для демонстрации различных ошибок.

```
  val response_secret : Dream.request → string → Dream.response Lwt.t
```

Этот ответ используется во время настройки TOTP. В рамках этого шага пользователям предоставляется секрет, который им необходимо занести в их генератор OTP.

```
  val response_enabled : Dream.request → Dream.response Lwt.t
```

Этот ответ информирует пользователя об успешном включении TOTP.

```
end
```

RESPONSES содержит представления данных для определённых событий.

```
module Make (R : RESPONSES) (M : MODEL) (V : FPauth_core.Auth_sign.  
VARIABLES with type entity = M.t) : sig ... end
```

Make создаёт стратегию для предоставленной модели с предоставленными ответами.

```
module JSON_Responses : RESPONSES
```

Модуль с ответами для TOTP в формате JSON.

```
module type HTML_settings = sig
```

```
  val app_name : string
```

```
end
```

Этот модуль должен содержать такие настройки, как имя приложения для заголовков.

```
module Make_HTML_Responses (S : HTML_settings) : RESPONSES
```

Этот функтор создаёт модуль, соответствующий RESPONSES, в формате HTML.

```
val make_html_responses : ?app_name:string → unit → (module  
RESPONSES)
```

make_html_responses ~app_name () - функция для удобного создания модуля HTML ответов без HTML_settings. Возвращает модуль первого класса.

4 FAuth-responses - шаблоны ответов

В данном пакете содержатся шаблоны ответов на основные события системы аутентификации в форматах HTML и JSON. Подробнее в разделе FAuth_responses.

4.1 Модуль FAuth_responses

module JSON : **sig** ... **end**

Этот модуль содержит ответы на базовые события FAuth в формате JSON.

module HTML : **sig** ... **end**

Этот модуль содержит ответы на базовые события FAuth в формате HTML.

5 Полная версия

С более подробной версией руководства можно ознакомиться online².

²<https://mikegeine.github.io/FPauth/>