|  |  |  |  |
| --- | --- | --- | --- |
| Министерство образования Республики Беларусь  Учреждение образования  Белорусский Государственный Университет Информатики и Радиоэлектроники | | | |
| Факультет | Компьютерных сетей и систем | | |
| Кафедра | Информатики  Дисциплина: Конструирование те технологии электронных вычислительных средств | | |
|  |  | | |
| **Проект**  по курсу «Современные технологии разработки программного обеспечения»  **СИСТЕМА ОНЛАЙН КРЕДИТОВАНИЯ** | | | |
| Студент:  гр. 7M2632  Ярош Г.И. | |  | Проверил:  Стержанов М.В. |
| Минск, 2017 | | | |

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ 3

1.1 Цель 3

1.2 Задачи 3

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ 4

2.1 Django Framework 4

Django Authentication 4

Django ORM 4

Django REST Framework 5

2.2 AngularJS 5

2.3 Flask 6

2.4 Docker 6

2.5 Nginx 7

3 РАЗРАБОТКА ПРИЛОЖЕНИЯ 8

3.1 Архитектура приложения 8

3.2 Back-end часть 9

3.2 Front-end часть 13

3.3 Сервис скоринга 15

4 ВЫВОД 15

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16

# ПОСТАНОВКА ЗАДАЧИ

## Цель

Разработать приложение с frontend и backend частью предоставляющее функциональность онлайн кредитования, что включает в себя оформление кредита, совершение оплаты по кредиту, закрытие кредита, а также возможность получение значения кредитного скоринга для конкретного пользователя.

## Задачи

1. Реализовать backend часть приложения с использованием фреймворка Django, которое должна предоставлять следующие функции, реализованные в виде RESTful API:
   1. Работа с базой данных;
   2. Аутентификация и авторизация;
   3. Управление пользователями, регистрация новых пользователей;
   4. Управление основными типами данных системы: Кредит, Шаблон кредита, Транзакция, Счет;
   5. Автоматический обновление данных по кредиту (остаток, начисление штрафов).
2. Реализовать frontend часть приложения с использованием библиотеки AngularJS, которая должна предоставлять пользовательский интерфейс для работы с API backend части приложения.
3. Реализовать сервис кредитного скоринга с использованием библиотеки Flask, который будет предсказывать вероятность выплаты кредита для клиента по его кредитной истории.
4. Все вышеперечисленные части приложения должны быть изолированы в отдельные контейнеры, используя технологию Docker.

# ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

## Django Framework

Django — [свободный](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) на языке [Python](https://ru.wikipedia.org/wiki/Python), использующий шаблон проектирования [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller).

Сайт на Django строится из одного или нескольких приложений, которые рекомендуется делать отчуждаемыми и подключаемыми. Один из основных принципов фреймворка — [DRY](https://ru.wikipedia.org/wiki/DRY) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Don't repeat yourself)

Также, в отличие от других фреймворков, обработчики [URL](https://ru.wikipedia.org/wiki/URL) в Django конфигурируются явно при помощи [регулярных выражений](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F), а не выводятся автоматически из структуры моделей контроллеров.

Архитектура Django похожа на «[Модель-Представление-Контроллер](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C-%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5-%D0%9A%D0%BE%D0%BD%D1%82%D1%80%D0%BE%D0%BB%D0%BB%D0%B5%D1%80)» (MVC). Контроллер классической модели MVC примерно соответствует уровню, который в Django называется Представление ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) View), а презентационная логика Представления реализуется в Django уровнем Шаблонов ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Template). Из-за этого уровневую архитектуру Django часто называют «Модель-Шаблон-Представление» (MTV).

## Django Authentication

Для реализации аутентификации и авторизации в приложении был использован встроенный в Django модуль Django Authentication. Он предоставляет возможность начальной аутентификации пользователя с помощью логина и пароля, генерацию токена, который используется для дальнейшей аутентификации пользователя.

Авторизация была реализована с помощью набора пермишенов, каждый из который регулирует возможность действия над типом данных (создание, чтение, обновление, удаление). Для простоты их использования пермишены объединяются в группы по типам пользователей (администратор, пользователь).

## Django ORM

Django ORM - встроенный во фреймворк Django ORM. Используется в приложении как основное средство работы с базой данных для формирования запросов в базу данных. и получения необходимых данных в виде Python-объектов. С помощью Django Models были описаны необходимые типы данных (уровень моделей).

Также было использовано средство Django Migrations для автоматического создания и поддержки миграций базы данных. Миграции позволяют сохранить историю изменений моделей приложения, а также облегчить обновление схемы базы данных до необходимой структуры.

## Django REST Framework

Django REST Framework – библиотека, которая расширяет возможности фреймворка Django и предоставляет набор классов для построения RESTful Web API. Для создания API были использованы классы view-sets, каждый из которых реализует один эндпоинт с набором CRUD действий (create, read, update, delete). Реализация каждого из действий была переопределена в зависимости от логики работы с типом данных.

Также Django REST Framework предоставляет возможность валидации входных данных. Данная возможность предоставляется классами Serializer, которые были использованы для объявления полей типов данных, а также реализации сложной логики валидации при создании или обновлении объектов.

Модуль Django Filters был использован для реализации и применения фильтров при получении списков данных. Фильтрация необходима для удобного отображения и использования объектов для фронт-энд части.

Для тестирования API были использованы встроенные классы APITests, а также библиотека factory-boy для генерации тестовых данных.

## AngularJS

AngularJS — [JavaScript](https://ru.wikipedia.org/wiki/JavaScript)-[фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) с [открытым исходным кодом](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5). Предназначен для разработки [одностраничных приложений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%BD%D0%BE%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5). Его цель — расширение браузерных приложений на основе [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller)-шаблона, а также упрощение тестирования и разработки.

Фреймворк работает с [HTML](https://ru.wikipedia.org/wiki/HTML), содержащим дополнительные пользовательские атрибуты, которые описываются директивами, и связывает ввод или вывод области страницы с моделью, представляющей собой обычные переменные JavaScript. Значения этих переменных задаются вручную или извлекаются из статических или динамических JSON-данных.

### Angular Controllers

Контроллеры Angular были использованы для реализации функциональности фронтенд приложения. Каждый контроллер реализует взаимодействие с API бекенд приложения, управление HTML шаблонами и компонентами пользовательского интерфейса. В них была реализована логика работы с объектами такая, как получение, создание, обновление объектов. Также в них реализовано использование фильтрации списка объектов для удобного их отображения.

Контроллеры объединяются с помощью модулей Angular по их назначению (работа с аутентификацией, контроллеры базовых страниц, контроллеры админки).

### Angular Services

Сервисы Angular были использованы для реализации часто используемых функций, таких как обработка ошибок API, генерация API URL, вспомогательная функциональность для доступа к бекенду.

Также с помощью сервисов была реализована логика аутентификации, авторизации и логика управления пользователями

### Angular Directives

Дерективы Angular были использованы для добавления поведения некоторым HTML элементам, а также для динамической генерации HTML элементов на основе данных, полученных с бекенда.

### Angular Constants

Константы Angular были использованы для отображения кодов операций и кодов статусов объектов бекенда на строковое их представление. Константы также объединяются с помощью модулей по месту их использования.

## Flask

Flask — [фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) для создания [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5) на языке программирования [Python](https://ru.wikipedia.org/wiki/Python), использующий набор инструментов Werkzeug, а также шаблонизатор [Jinja2](https://ru.wikipedia.org/wiki/Jinja). Относится к категории так называемых [микрофреймворков](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA&action=edit&redlink=1) — минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

Фреймворк Flask был использован для реализации сервиса скоринга, который предоставляет только один API эндпоинт, что удобно реализуется с помощью Flask с минимальным количеством исходного кода.

## Docker

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в среде [виртуализации на уровне операционной системы](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82%D1%83%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B). Позволяет «упаковать» приложение со всем его [окружением](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BE%D0%BA%D1%80%D1%83%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5&action=edit&redlink=1) и зависимостями в контейнер, который может быть перенесён на любую [Linux](https://ru.wikipedia.org/wiki/Linux)-систему с поддержкой [cgroups](https://ru.wikipedia.org/wiki/Cgroups) в [ядре](https://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_Linux), а также предоставляет среду по управлению контейнерами. Изначально использовал возможности [LXC](https://ru.wikipedia.org/wiki/LXC), с 2015 года применял собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux — libcontainer. С появлением ​Open Container Initiative начался переход от монолитной к модульной архитектуре.

Используется для разделения приложения, с помощью контейнеров. Каждый контейнер содержит в себе только бекенд, фронтенд или сервис-скоринга, что предоставляет каждому сервису изолированную среду исполнения. Также контейнерная система предоставляет возможность гибкого управления сетями для взаимодействия компонентов приложения (взаимодействие бекенда и фронтенда части, бекенда и сервиса скоринга).

Каждый сервис приложения (бекенд, фронтенд, скоринг) предварительно упаковывается в докер-образ, из которого в дальнейшем создаются и запускаются контейнеры. Также в него упаковываются зависимости сервисов и необходимые файлы. Это позволяет легко переносить приложения с одной хост машины на другую без необходимости заново устанавливать зависимости.

Также в отдельном контейнере запускается сервер базы данных, что освобождает от необходимости скачивать и устанавливать его на хост-машину.

## Nginx

Nginx — это HTTP-сервер и обратный прокси-сервер, почтовый прокси-сервер, а также TCP/UDP прокси-сервер общего назначения, изначально написанный [Игорем Сысоевым](http://sysoev.ru/). Согласно статистике Netcraft nginx обслуживал или проксировал [23.76% самых нагруженных сайтов в марте 2018 года](https://news.netcraft.com/archives/2018/03/27/march-2018-web-server-survey.html).

Используется для обслуживания запросов статических файлов фронтенд приложения, а также для проксирования запросов на бекенд приложение.

# 3 РАЗРАБОТКА ПРИЛОЖЕНИЯ

## 3.1 Архитектура приложения

Приложение построено с использованием архитектуры MVC, где уровни моделей и контроллеров реализованы в бекенд-части приложения, а уровень представления реализован с помощью отдельного фронтенд приложения.

Фронтенд приложение реализует основную логику работы пользовательского интерфейса. Фронтенд взаимодействует с бекенд частью используя RESTful API, реализованный в бекенд приложении. Бекенд API разделен на эндпоинты, каждый из которых предоставляет набор действий над объектами одного типа данных (создание, чтение, изменение, удаление). Некоторые эндпоинты, такие как эндпоинт аутентификации, предоставляют урезанных список действий, что обуславливается спецификой типов данных.

В свою очередь бекенд приложение реализует логику работы с объектами данных, такими как пользователи приложения, кредиты, транзакции и т.д. Оно взаимодействует с базой данных, в которой хранятся все основные данные системы. Также бекенд часть реализует всю логику по обработке данных.

Сервис скоринга предоставляет возможность получения значения кредитного скоринга для конктетного пользователя по его кредитной истории. Он используется бекенд-приложением, которое на основе скоринга определяет возможность выдачи кредита.

На рисунке 1 схематично изображены компоненты системы и взаимодействия между ними.

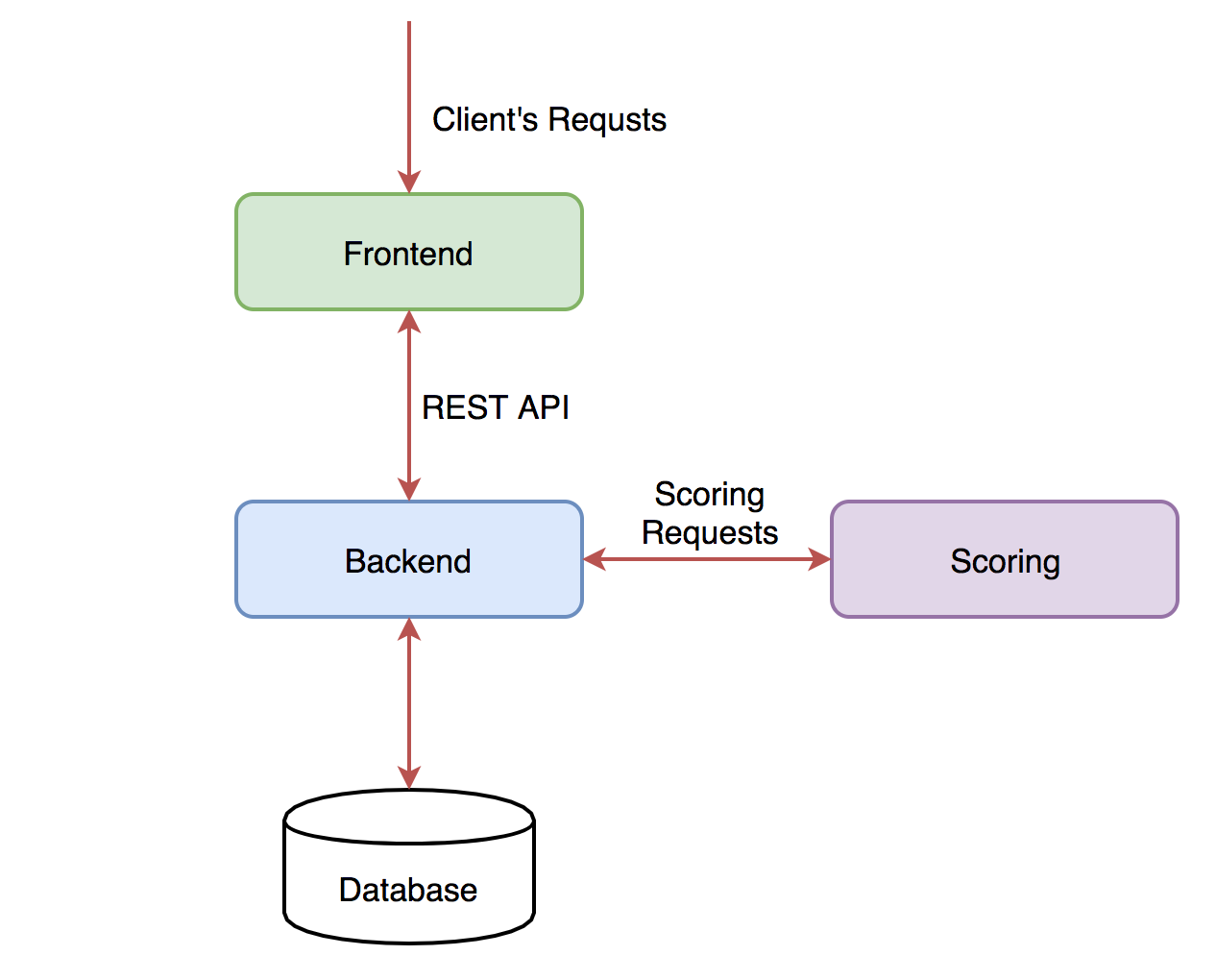


Рис. 1. Общая архитектура приложения

Все части приложения (фронтенд, бекенд, скоринг) запускаются внутри докер контейнеров и работают изолированно друг от друга. Взаимодействие между ними происходит по внутренней сети докера. Во вне открыт только фронтенд сервис, что означает, что только запросы к фронтенд приложению будут обработаны.

Для упрощения создания и запуска контейнеров, для каждого сервиса были созданы докер-образы. Они содержат код приложения, а также зависимости, необходимые для работы приложения.

## 3.2 Back-end часть

Бекенд часть реализована на основе фреймворка Django с использованием библиотеки Django Rest Framework. Она реализует аутентификациию и авторизацию, логику работы с данными, взаимодействие с базой данных.

Схема базы данных приведена на рисунке 2.

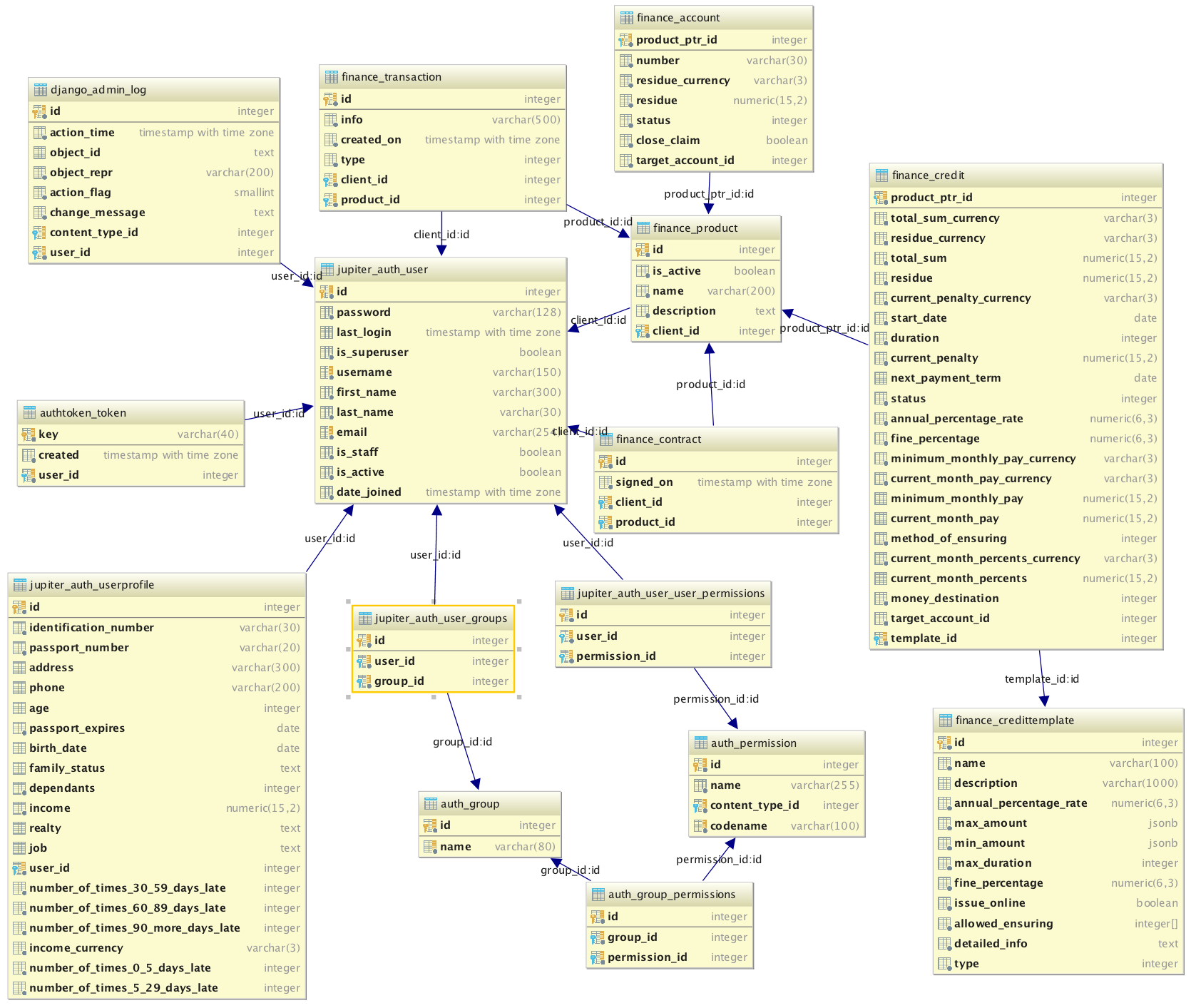


Рис. 2. Схема базы данных

Бекенд разделен на три модуля:

* Core – модуль, содержащий базовые функциональность, такую как авторизация, вспомогательные классы для реализации REST API, фильтрации и модульных тестов.
* Auth – модуль, реализующий логику аутентификации и авторизации в приложении. Также содержит модели пользователя и профайла пользователя и реализует API для работы с ними.
* Finance – модуль, реализующий основную логику работы приложения. Содержит основные модели, реализующие типы данных в системе, а также предоставляет API для работы с ними.

### Модуль Core

Модуль Core содержит базовые классы и функции, которые используются другими модулями для реализации основной функциональности приложения. Диаграмма классов модуля Core представлена на рисунке 3.

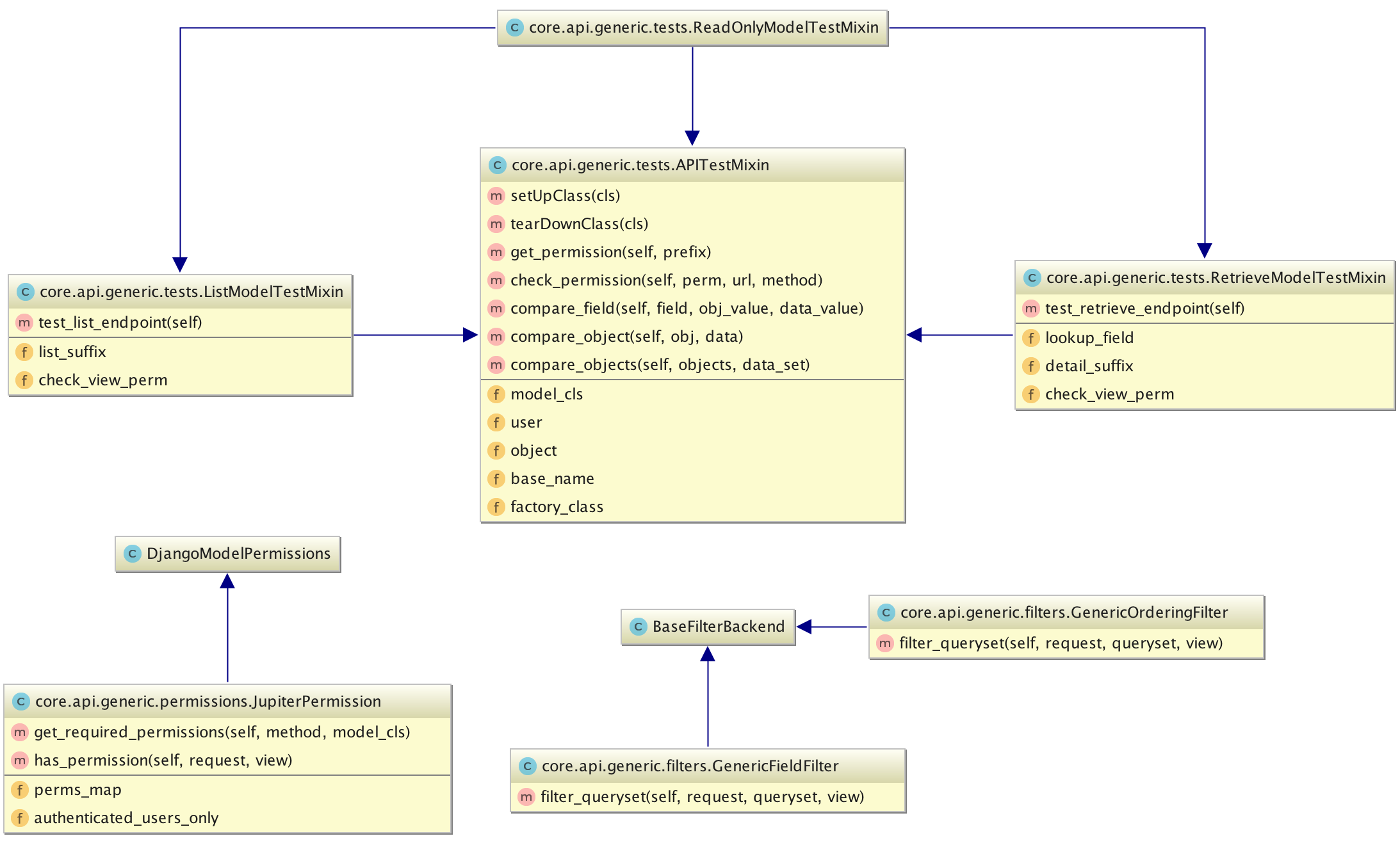


Рис. 3. Диаграмма классов модуля Core

Классы APITestMixin, ReadOnlyModelTestMixin, ListModelTestMixin, RetrieveModelTestMixin служат для разработки модульных тестов API. Они содержат универсальную логику для тестирования чтения объектов, а также APITestMixin содержит вспомогательную функциональность для разработки API тестов для других действий над объектами, такими как создание, изменение и удаление.

Классы GenericOrderingFilter и GenericFieldFilter реализуют поддержку фильтрации и упорядочивание для списка объектов. Они универсальны, то есть пригодны для использования при реализации любого эндпоинта API. GenericOrderingFilter позволяет использовать упорядочивание объектов в ответе по значению поля. А GenericFieldFilter позволяет фильтровать объекты в ответе API, передавая для них DjangoORM фильтры.

Класс JupiterPermission расширяет возможности Django по обработке пермишенов. Он реализует логику работы с пермишенами вида {module}.{action}\_{model}. Он автоматически определяет модель и само совершаемое действие, конструирует соответсвующий пермишен и проверяет его наличие у пользователя.

### Модуль Auth

В модуле Auth реализованы функции аутентификации и авторизации пользователей, а также реализованы модели пользователя и профайла пользователя, содержащего дополнительную информацию о пользователе. Схема классов моделей модуля Auth представлена на рисунке 4.

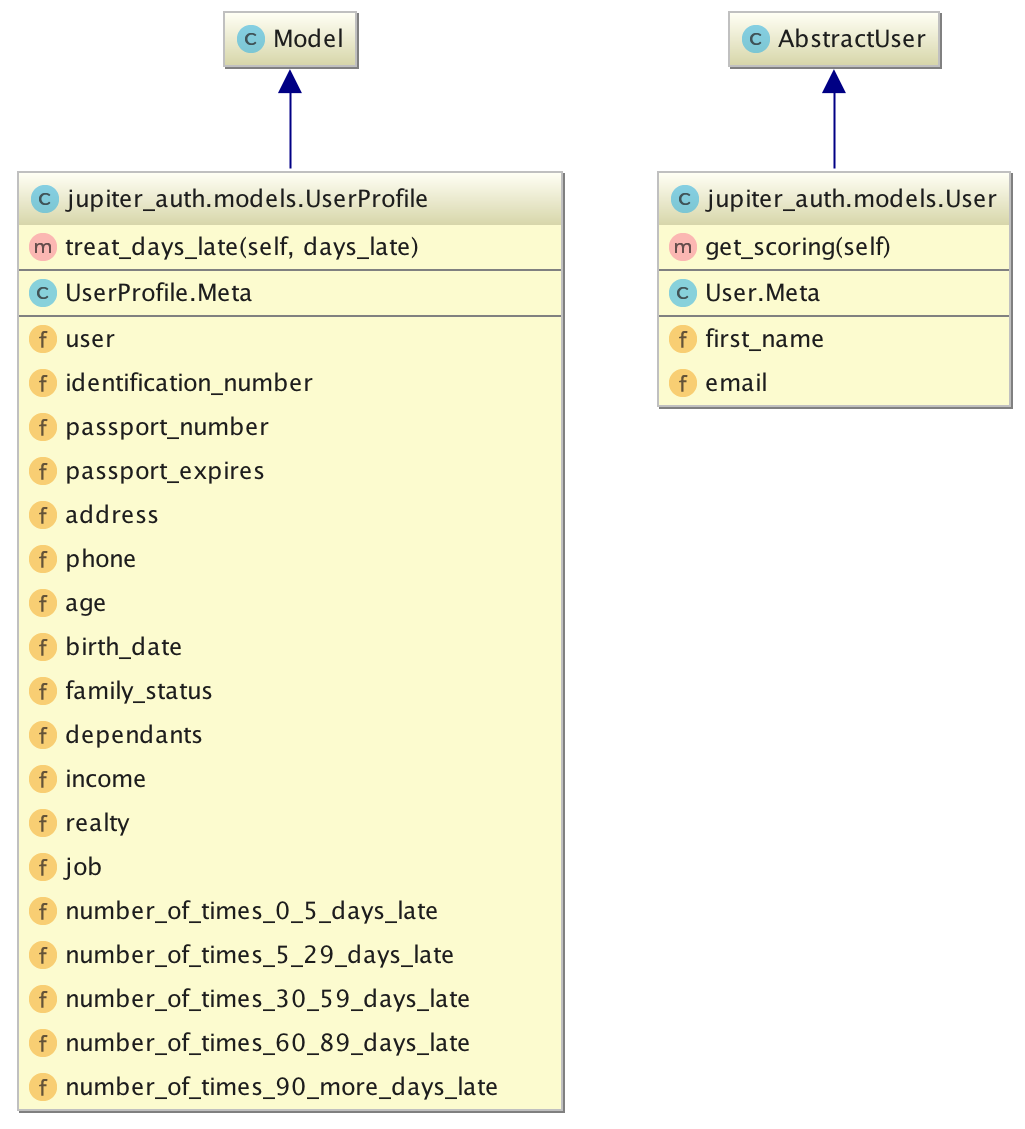


Рис 4. Классы, реализующие модели пользователя и профайла

Класс User расширяет встроенный класс AbstractUser, добавляя дополнительные поля, а также содержит функцию для работы с сервисом скоринга. Класс UserProfile содержит поля, содержащие дополнительную информацию о пользователе, а также информацию о его кредитной истории.

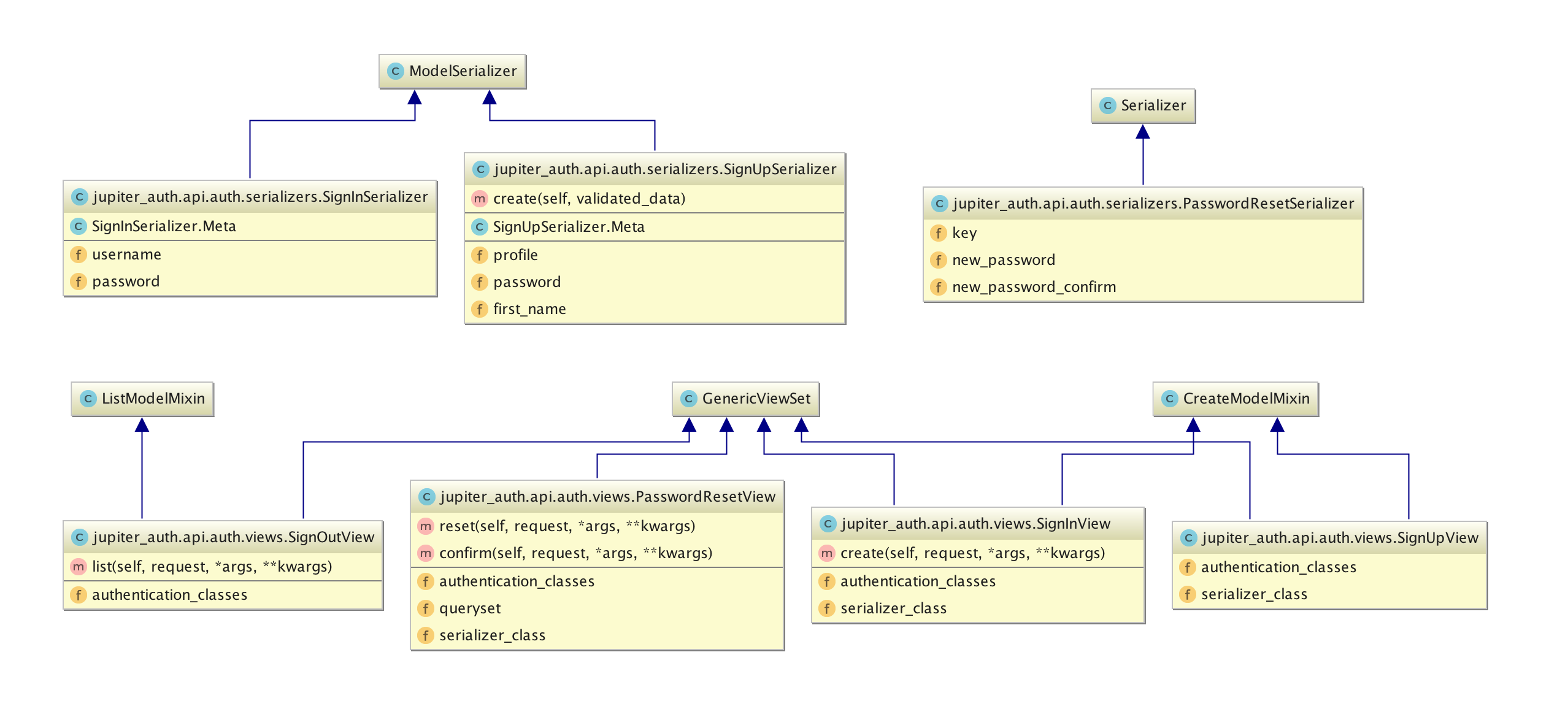


Рис. 5. Классы, реализующие API аутентификации и авторизации

На рисунке 5 отображены классы, реализующие API эндпоинты для аутентификации, авторизации, регистрации, а также смены пароля для пользователя.

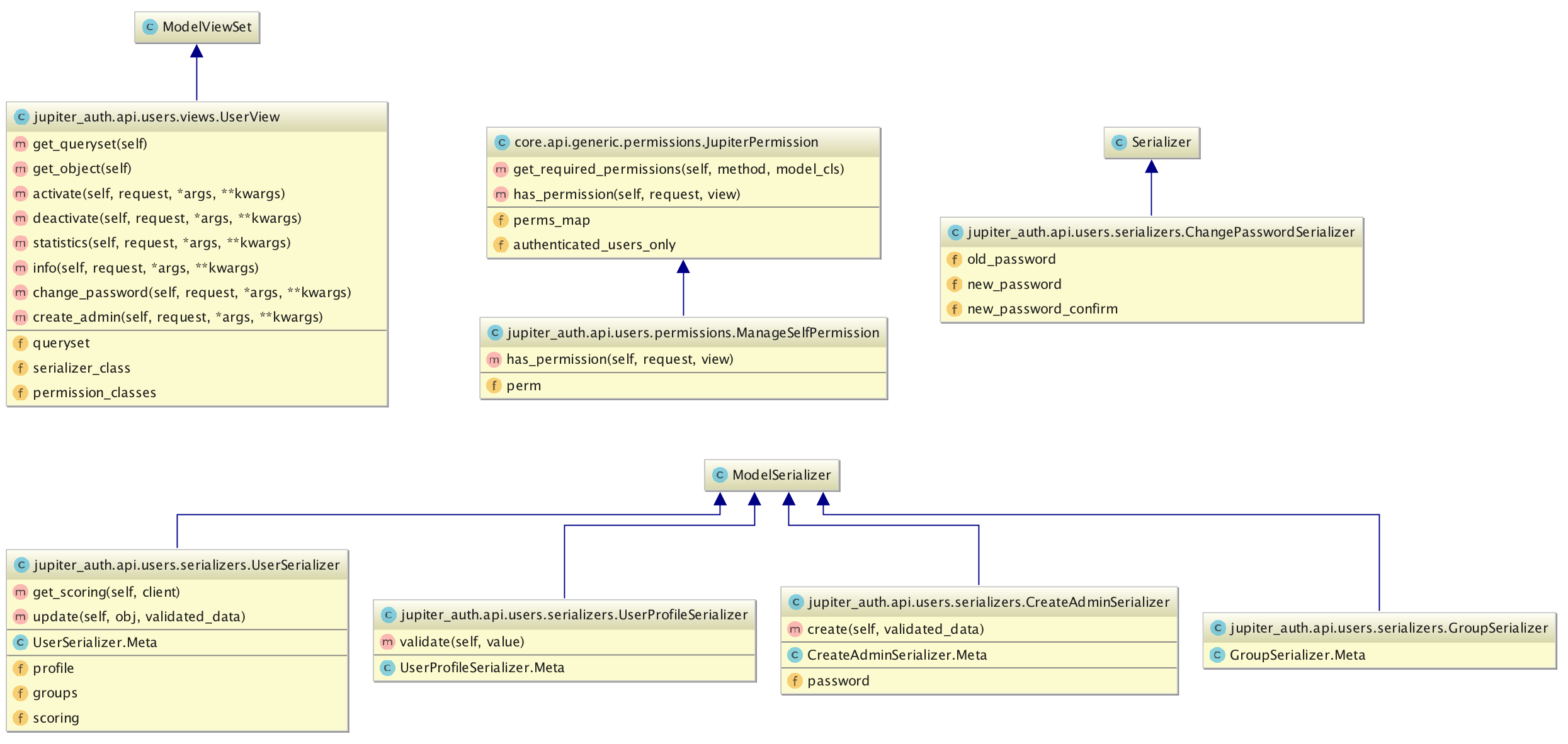


Рис. 6. Классы, реализующие API для управления пользователями

На рисунке 6 приведены классы для реализации API энпоинтов для создания новых пользователей, получения списка пользователей, изменения и удаления объекта пользователя, а также для смены пароля пользователя, активации и деактивации пользователей.

Также в модуле Auth содержаться функции для инициализации групп пользователей: администратор и пользователь. Эти группы содержат пермишенны, которые определяют, какие действия пользователи могут производить над объектами.

### Модуль Finance

Модуль Finance содержит основную логику работы приложения. В нем реализованы модели для работы с основными типами данных приложения: Кредит, Шаблон Кредита, Транзакция, Счет (рис. 7).

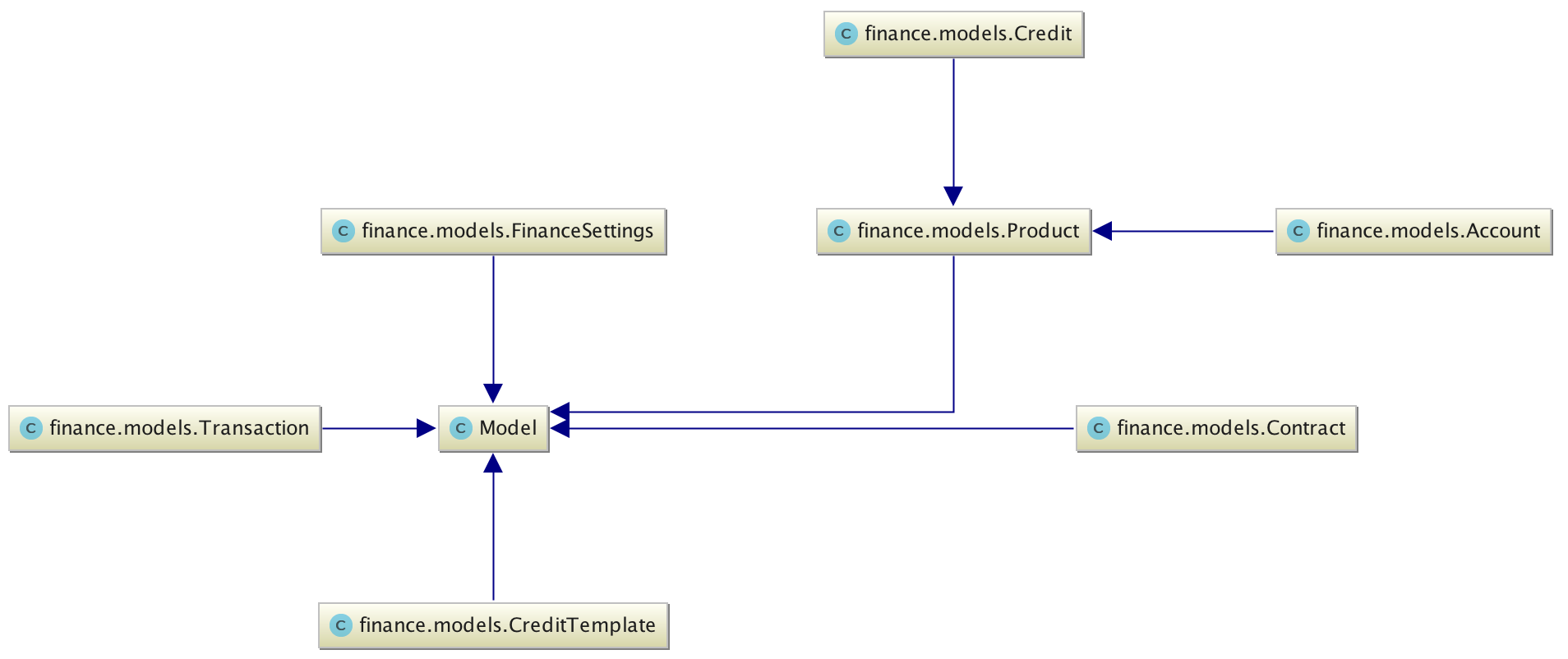


Рис. 7. Модели модуля Finance

Для каждой из приведенных на диаграмме моделей был реализован соответсвтвующий API, поддерживающий набор CRUD действий, а также действий, специфических для типа данных.

Также в данном модуле реализована функциональность по пересчету остатка кредита и начисления штрафов. Пересчет данных происходит раз в день. Это реализовано с помощью утилиты cron, которая запускает соответствующую функцию приложения по расписанию.

Некоторые действия пользователя над объектами, такие как, открытие кредита, перевод средств, закрытие кредита и т.д., отображаются в системе путем создания объекта транзакции. Такой объект содержит тип выполненной операции, пользователя, и время выполнения.

### Тестирование

Для тестирования бекенд приложения были реализованы модульные тесты. Они тестируют каждый эндпоинт API, используя универсальную логику тестирования для стандатрных CRUD действий, а также содержат специфические тест кейсы для отдельных сложных действий над объектами.

## 3.2 Front-end часть

Фронтенд часть реализована в виде отдельного приложения с использованием библиотеки AngularJS. Фронтенд разбит на модули, соответствующие модулям бекенда. Каждый модуль реализует логику работы пользовательского интерфейса для соответствующих функций бекенда.

Для обработки запросов к фронтенд приложению используется веб-сервер nxinx. Он обрабатывает запросы к статическим данным (HTML шаблоны, Java Script файлы), а также проксирует запросы к бекенд приложению на соответствующий порт.

### Модуль Core

Модуль Core содержит HTML шаблон для главной страницы, а также базовые шаблоны отображения других страниц и страницы об ошибке. Также в этом модуле реализованы контроллеры, содержащие базовую логику работы фронтенд приложения.

Модуль Core реализует Angular-сервисы для обработки ошибок бекенда, а также для генерации URL для доступа к бекенд API.

### Модуль Public

Данный модуль содержит логику работы фронтенд приложения, доступную для неавторизованного пользователя. Она включает получения списка и информации о кредитах, а также генерация формы для оформления кредита.

### Модуль Auth

Модуль Auth содержит всю функциональность, связанную с аутентификацией и авторизацией пользователя. Он содержит HTML шаблоны с формами для регистрации, логина и изменения пароля, а также контроллеры, реализующие данные функции.

Также данный модуль реализует сервис для аутентификации по токену, полученному после логина. Данный сервис используется всеми другими контроллерами, чтобы подставить токен в запрос, который требует аутентификации.

В модуле Auth реализован Angular сервис для работы с пользователем. Этот сервис содержит функции для проверки, аутентифицирован ли пользователь, а также к какой группе он принадлежит.

### Модуль Admin

Модуль Admin содержит основную логику по работе с авторизованным пользователем. Он реализует личный кабинет для пользователя, который изменяется в зависимости от типа пользователя: администратор или клиент.

Администратор имеет доступ к широкому набору функций: управление клиентами, управление счетами клиентов, управление кредитами и настройка кредитного скоринга, просмотр списка транзакций всех пользователей. Вместе со списком пользователей, администратору доступен просмотр значения кредитного скоринга по каждому пользователю.

Клиент же может изменять некоторые свои данные, просматривать информацию о своих кредитах, производить оплату по кредиту, просматривать свои транзакции.

Модуль Admin содержит соответствующие контроллеры и HTML шаблоны для реализации вышеперечисленных функций.

## 3.3 Сервис скоринга

В качестве данных для обучения были использованы тестовые данные для скоринга предоставляемые российским банком ТКС:

* <https://static.tcsbank.ru/documents/olymp/SAMPLE_CUSTOMERS.csv>
* https://static.tcsbank.ru/documents/olymp/SAMPLE\_ACCOUNTS.csv

Далее произвел предварительную обработку данных, используя библиотеку pandas. Она включала в себя приведение сумм в российских рублях к белорусским рублям, удаление столбцов, информация в которых не соответствует информации, которой оперирует приложение.  
Далее я произвел корреляционный анализ полученных данных. Определил и удалил столбцы, которые имеют нулевой коэффициент корреляции. Также удалил столбцы, которые имеют коэффициент корреляции близкий к единице.

Следующим шагом было простроение модели и обучение ее. В качестве алгоритма я выбрал градиентный бустинг, который рекомендуется как самый эффективный для поставленной задачи. Для этого использовал библиотеку sklearn. Для определения эффективности модели, данные были разбиты на выборки для обучения и для контроля. В итоге точность предсказания по контрольным данным составиля 0.765.

Сервис скоринга реализован с помощью библиотеки Flask. В начале своей работы, он звгружает модель предсказания из файла. Затем основной бекенд по необходимости запрашивает значения скоринга, передавая необходимую информацию в сервис. Сервис возвращает значение вероятности выплатить кредит для конкретного пользователя.

# 4 ВЫВОД

В процессе работы над проектом были изучены технологии Django, Flask AngularJS. Было спроектировано и реализовано приложение с frontend и backend частью предоставляющее функциональность онлайн кредитования с возможностью оформления кредита, совершение оплаты по кредиту, закрытие кредита, а также с возможностью получения значения кредитного скоринга для конкретного пользователя.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Django Framework Documentation [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://docs.djangoproject.com/en/2.0/.

[2] Django REST Framework Documentation [Электронный ресурс]. – Электронные данные. – ­Режим доступа: http://www.django-rest-framework.org/.

[3] AngularJS Developer Guide [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://docs.angularjs.org/guide.

[4] Docker Documentation [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://docs.docker.com/.

[5] Nginx Documentation [Электронный ресурс]. – Электронные данные. – ­Режим доступа: https://nginx.org/en/docs/.

[6] Flask Documentation [Электронный ресурс]. – Электронные данные. – ­Режим доступа: http://flask.pocoo.org/docs/0.12/.

[6] Sklearn GradientBoostingClassifier Documentation [Электронный ресурс]. – Электронные данные. – ­Режим доступа: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html