ОСНОВНАЯ ЧАСТЬ

1 Исследование предметной области

Предметной областью является деятельность кафедры МОСИТ, однако в рамках предмета исследования выделяются два основных процесса:

1) расчёт индивидуальной нагрузки преподавателей;

2) контроль исполнения поручений.

1.1 Расчёт индивидуальной нагрузки

Данный процесс включает в себя подпроцессы, такие как:

1) создание индивидуального плана;

2) корректировка индивидуального плана.

1.1.1 Участники процесса

Основными участниками процесса являются сотрудники кафедры (Таблица 1).

Таблица 1 – Описание сотрудников

Сотрудник	Описание
Преподаватель	Создаёт индивидуальный план
Методист	Корректирует индивидуальный план

1.1.2 Данные

Основной информацией в рамках процесса является индивидуальный план.

Индивидуальный план представляет собой файл в формате электронной таблицы, состоящий из 6 листов.

На первом листе содержится информация о преподавателе:

1) институт;

2) кафедра;

3) фио;

4) должность;

5) учёная степень;

6) учёное звание.

На втором листе содержится список учебных работ за осенний семестр, более конкретно определяющийся как список дисциплин, которые ведёт преподаватель. Также на листе содержится информация о планируемых и фактических затратах часов на выполнение плана.

На третьем листе содержится список учебных работ за весенний семестр, более конкретно определяющийся как список дисциплин, которые ведёт преподаватель. Также на листе содержится информация о планируемых и фактических затратах часов на выполнение плана.

На четвёртом листе содержится сводная информация по учебной работе, в которой подсчитан итог за два семестра. Также на листе содержится список учебно-методических работ с указанием планируемых и фактических затрат часов на их выполнение.

На пятом листе содержится список научно-исследовательских работ и организационно-методических работ с указанием планируемых и фактических затрат часов на их выполнение. Также на листе содержится сводка по общей годовой нагрузке.

На шестом листе содержится приложение к отчёту о работе преподавателя.

В рамках процесса «Создание индивидуального плана» входными данными являются:

1) информация о преподавателе;

2) информация о дисциплинах;

3) информация о группах;

Основанием для создания индивидуального плана является документ о нормах по планированию и учёту труда профессорско-преподавательского состава.

В качестве выходных данных, результатом процесса является индивидуальный план, представляющий собой файл в формате электронной таблицы (Рисунок 1).

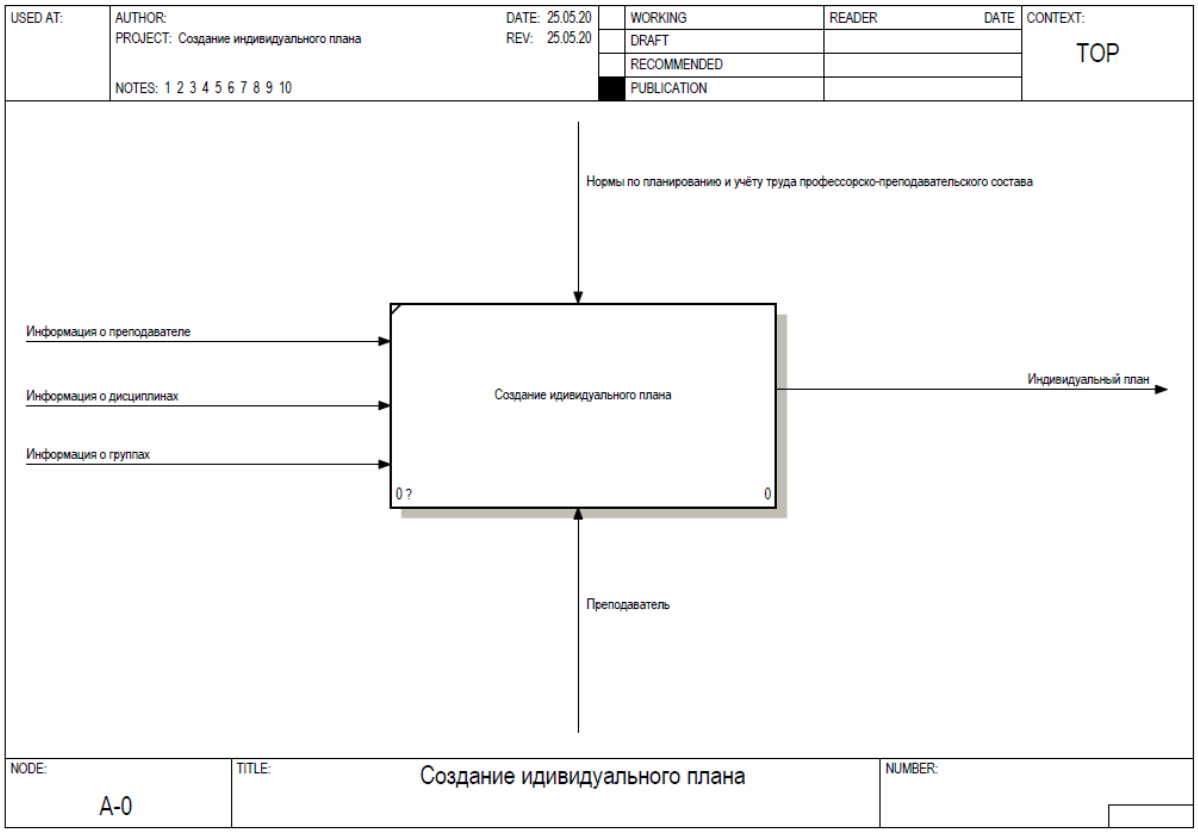

Рисунок 1 – Подпроцесс «Создание индивидуального плана»

В рамках процесса «Корректировка индивидуального плана» основными данными является индивидуальный план преподавателя.

В качестве выходных данных, результатом процесса является скорректированный индивидуальный план (Рисунок 2).

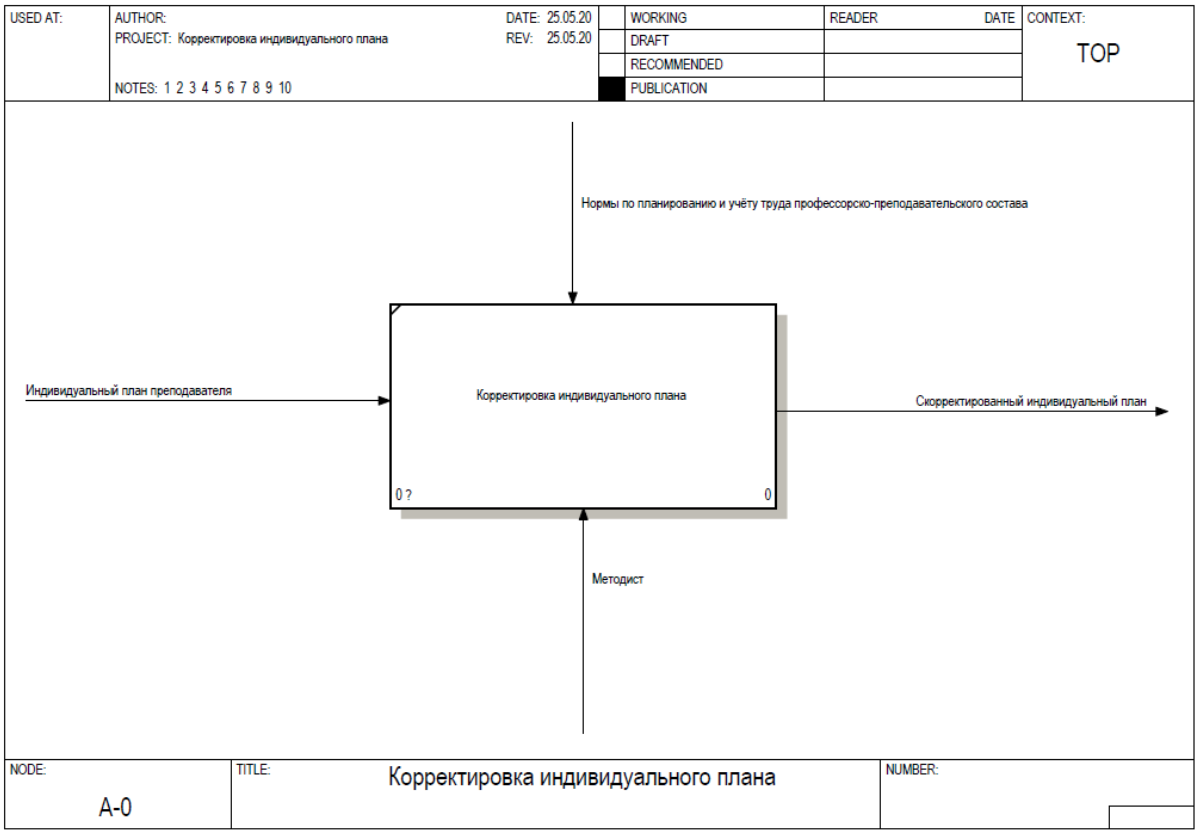

Рисунок 2 – Подпроцесс «Корректировка индивидуального плана»

Стоить отметить, что корректировка индивидуального плана осуществляется вручную, с заполнением 4-го и 5-го листов.

1.2 Контроль исполнения поручений

Данный процесс включает в себя подпроцессы:

1) назначение поручения;

2) исполнение поручения.

1.2.1 Участники процесса

Основными участниками процессов являются сотрудники кафедры (Таблица 2).

Таблица 2 – Описание сотрудников

Сотрудник	Описание
Преподаватель	Исполняет назначенные поручения
Методист	
Зам. по учебной работе	Исполняет назначенные поручения и также назначает поручения своим подчинённым
Зам. по научной работе	
Зам. по учебно-методической работе	
Ответственный за МТО	
Ответственный по работе со студентами	

Учёный секретарь	
Заведующий кафедры	

1.2.2 Данные

Основной информацией в рамках процесса «Контроль поручений» является поручение.

Поручение представляет собой задачу или встречу, на исполнение которой назначаются подчинённые сотрудники.

Также поручение содержит следующие сведения:

1) постановку задачи или цель встречи;
2) описание;
3) документы;
4) срок исполнения.

Стоит отметить, что назначать поручения может только начальник в распоряжении которого находятся подчинённые сотрудники.

В рамках процесса «Создание поручения» входными данными являются:

1) информация о сотруднике или сотрудниках, которому или которым назначается поручение;

2) постановка задачи или цель встречи;

3) описание;

4) прикрепляемые документы;

5) сроки исполнения.

В качестве выходных данных, результатом процесса является назначенное поручение (Рисунок 3).

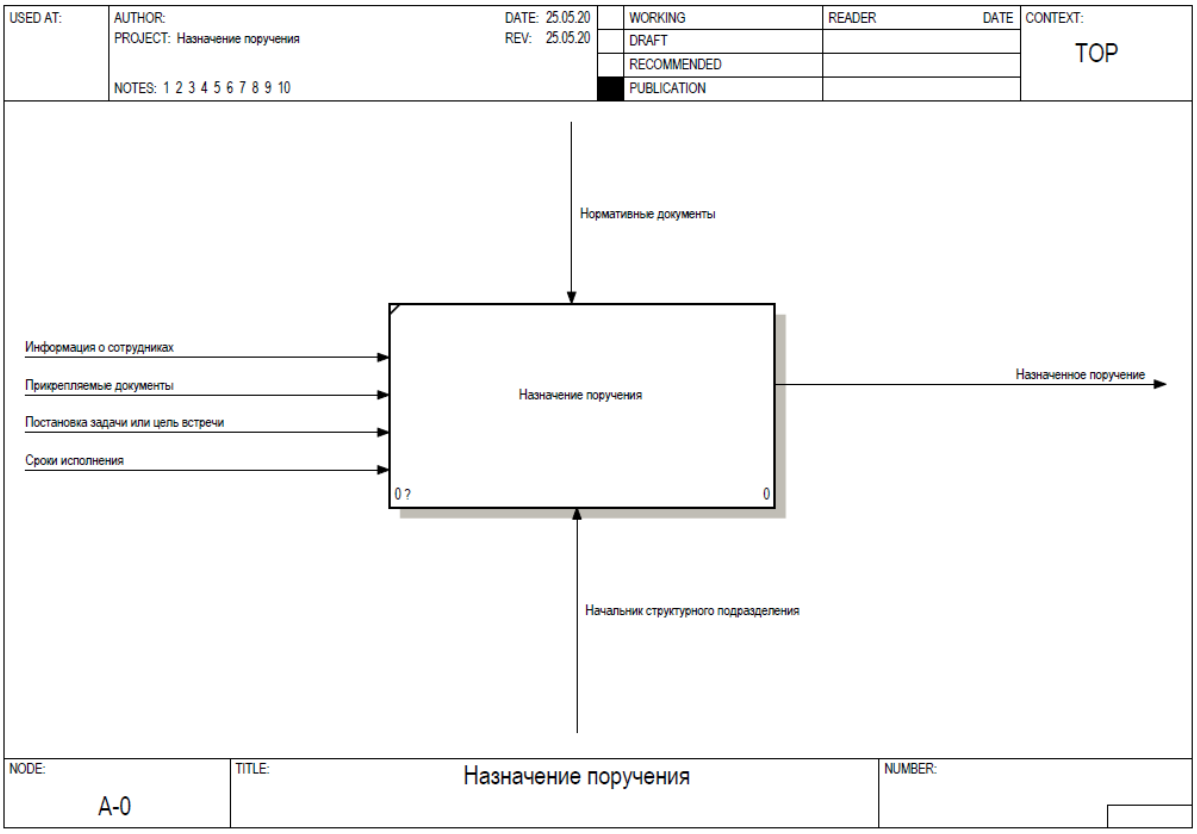

Рисунок 3 – Подпроцесс «Назначение поручения»

В рамках процесса «Исполнение поручения» входными данными является назначенное поручение.

Выходными данными процесса является исполненное поручение (Рисунок 4).

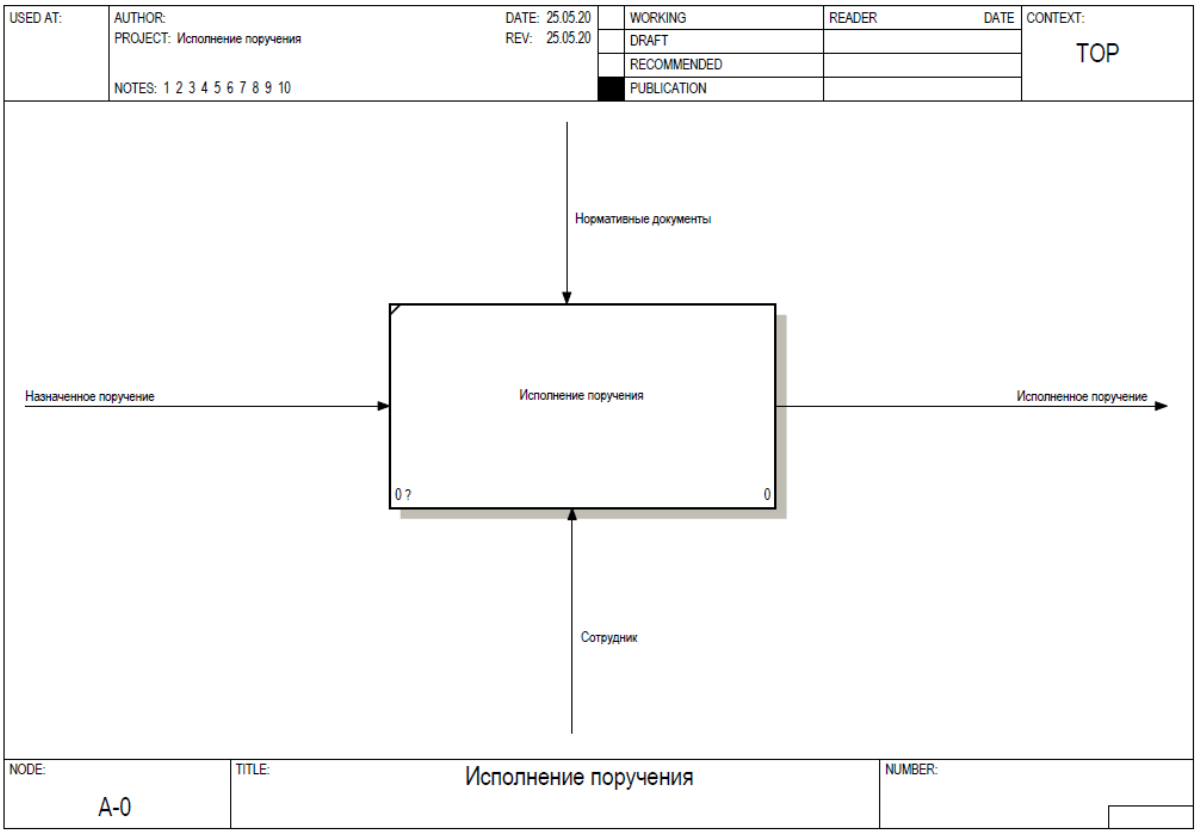

Рисунок 4 – Подпроцесс «Исполнение поручения»

1.3 Анализ существующих решений

1.3.1 Тандем.Университет

На данный момент для автоматизации процесса расчёта индивидуальной нагрузки преподавателей используется программное решение Тандем.Университет – Модуль «Нагрузка» (Далее Тандем).

По представленной на официальном сайте информации, Тандем имеет следующие преимущества:

1) Технические:

— web-ориентированность;

— модульность решения и возможность модульного внедрения;

— возможность выбора опций в рамках модулей;

— открытая J2EE платформа;

— простота обновлений АРМов – не требуются усилия со стороны конечных пользователей;

— надежность и масштабируемость решения;

— кроссплатформенность;

— быстрота развертывания решения;

— встроенный модификатор скриптов и печатных шаблонов документов;

— покрытие данных системы веб-сервисами и view для построения отчетов внешними средствами, либо с помощью интегрированной Pentaho.

2) Экономические:

— нет необходимости приобретения стороннего программного обеспечения, возможность работы пользователей с системой на свободном программном обеспечении (Open source);

— как правило, не требуется приобретение дорогостоящего серверного оборудования;

— лицензируется инсталляция, а не рабочие места.

3) Развитие:

— поставка с открытыми исходными кодами и правом на доработку решения при условии не распространения третьим лицам;

— широкие интеграционные возможности со сторонними программными продуктами.

1.3.1.1 Функциональные возможности

Основными функциями Тандема являются:

1) получение и корректировка сводки планируемого контингента студентов;

2) получение и корректировка планируемых учебных групп (потоков, подгрупп);

3) получение и печать расчета нагрузки на читающих подразделениях (кафедрах);

4) формирование штата, вакансий на кафедрах;

5) распределение строк (часов) нагрузки между преподавателями (вакансиями) на кафедрах, передача часов между кафедрами;

6) контроль объема нагрузки на ставку, должность;

7) коррекция расчета нагрузки, контроль изменений, актуализация расчета с учетом изменений в фактическом контингенте студентов, траекториях обучения, нормах времени, и других факторов;

8) фиксация внеучебной нагрузки в индивидуальных планах преподавателей;

9) получение и печать индивидуальных планов преподавателей, печать учебных поручений;

10) получение фактических учебных групп (групп, подгрупп, потоков) по данным нагрузки и контингента студентов в начале учебного года (семестра);

11) гибкое управление нормами времени и их применением: простые нормы, нормы-формулы, нормы-скрипты;

12) возможность отслеживания показателей (при использовании дополнительных модулей системы) для сравнения планируемой учебной нагрузки с ее фактическим выполнением (по преподавателям, кафедрам, образовательной организации).

1.3.1.2 Архитектура

Тандем имеет следующую архитектуру (Рисунок 5).

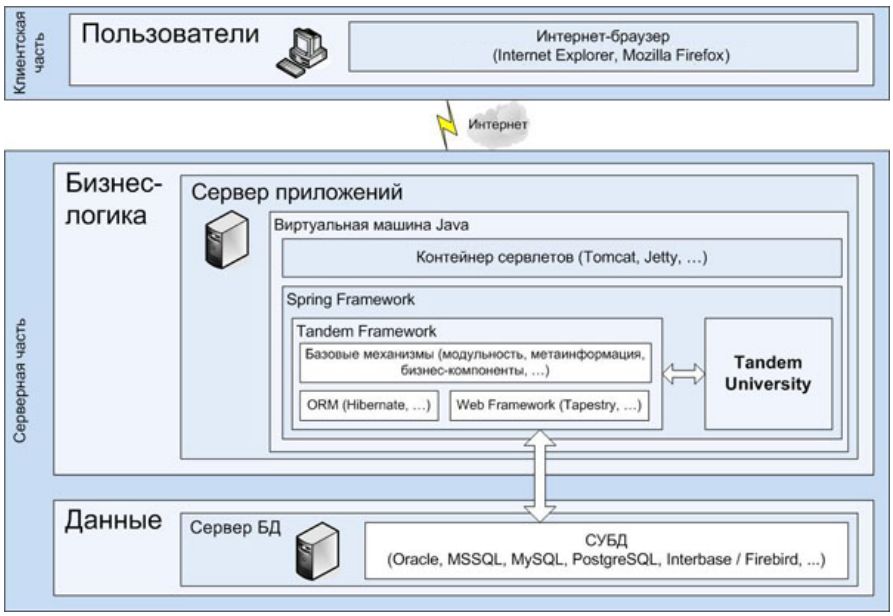

Рисунок 5 – Архитектура Тандема

Архитектура Тандема представляет собой модульную структуру, которая позволяет расширять функционал.

Из рисунка видно, что на базе Spring Framework был разработан Tandem Framework. И на базе него идёт разработана система Tandem University.

Стоит также отметить, что данные могут храниться в различных СУБД, в зависимости от той, которая по умолчанию была предусмотрена в учебном заведении.

1.3.1.3 Заключение

Не смотря на то, что основным преимуществом Тандема является открытость к расширению функционала после её приобретения, в свою очередь для того, чтобы внести некоторые изменения в функционал необходимо будет сначала обучить специалиста или студента. Это может вызывать определённые трудности со стороны учебного заведения.

Также текущая функциональность связанная с расчётом нагрузки не позволяет автоматически рассчитать нагрузку для учебно-методической, научно-исследовательской и организационно-методической работ (которые представлены на 4 и 5 листах индивидуального плана).

В дальнейшем при разработке собственной системы следует учитывать возникающие недостатки в работе Тандема и не допускать их в разрабатываемой системе.

1.3.2 SharePoint

На данный момент для автоматизации процесса «Контроля поручений» используется SharePoint. SharePoint представляет собой набор программных продуктов предназначенные для обеспечения документооборота и организации интранет сети.

1.3.2.1 Функциональные возможности

1.3.2.2 Архитектура

1.3.2.3 Заключение

2 Формирование требований к системе

2.1 Общие требования

Система должна предоставлять функции для хранения материалов кафедры.

Система должна предоставлять функции для создания, контроля и исполнения поручений.

Система должна предоставлять функции для работы с индивидуальными планами.

Система должна представлять собой веб-приложения. Исходя из этого следует учесть поддержку браузеров: Chrome, Firefox и Opera.

2.2 Требования к функциям

Разделим предоставляемые функции по пользователям.

Для всех типов пользователей предоставляются следующие функции:

1) хранение материалов;

2) работа с поручениями;

3) просмотр информации профиля;

4) смена пароля.

В свою очередь, хранение материалов включает в себя:

1) загрузку файлов;

2) удаление загруженных файлов;

3) скачивание загруженных файлов;

4) просмотр файлов и папок;

5) создание папок;

6) удаление папок.

В свою очередь, работа с поручениями включает в себя:

1) просмотр назначенных поручений;

2) комментирование поручения;

3) прикрепление поручения;

4) пометка поручения как выполненного.

Прочие функциональные возможности приведены в таблице .

Таблица 3 – Функциональные возможности пользователей

Пользователь	Функциональная возможность
Преподаватель	Просмотр индивидуальных планов
	Редактирование индивидуальных планов
	Скачивание индивидуальных планов
Методист	Добавление индивидуальных планов
	Изменение индивидуальных планов
	Удаление индивидуальных планов
Зам. по учебной работе	Создание поручений, изменение поручений, удаление поручений
Зам. по научной работе	
Зам. по учебно-методической работе	
Ответственный за МТО	
Ответственный за работу со студентами	
Учёный секретарь	
Зав. кафедрой	

2.3 Требования к надёжности

Система должна восстанавливать свою работоспособность (в случае аппаратных сбоев) после корректного перезапуска аппаратных средств.

Также система должна выдавать сообщения пользователю в случае совершения некорректных действий, таких как: ввод некорректных данных, возникновение исключительных ситуаций в самой системе.

2.4 Требования к безопасности

Прямой доступ к конфиденциальной информации должен быть исключён.

Доступ к базе данных может быть осуществлён только системным администратором.

Наличие встроенной панели администратора не подразумевается.

3 Проектирование системы

3.1 Проектирование архитектуры

Архитектура системы является клиент-серверной.

На сервере располагается база данных и само веб-приложение.

На клиенте осуществляется доступ к веб-приложению посредством браузера.

Следует разделить функциональные возможности на основные и сервисные подсистемы.

К основным подсистемам относятся:

1) подсистема индивидуальных планов;

2) подсистема кадров;

3) подсистема поручений;

4) подсистема хранения материалов.

К сервисным подсистемам относятся:

1) подсистема пользователей;

2) подсистема прав доступа.

3.2 Проектирование базы данных

Для каждой из подсистем представлены следующие таблицы, на рисунках .

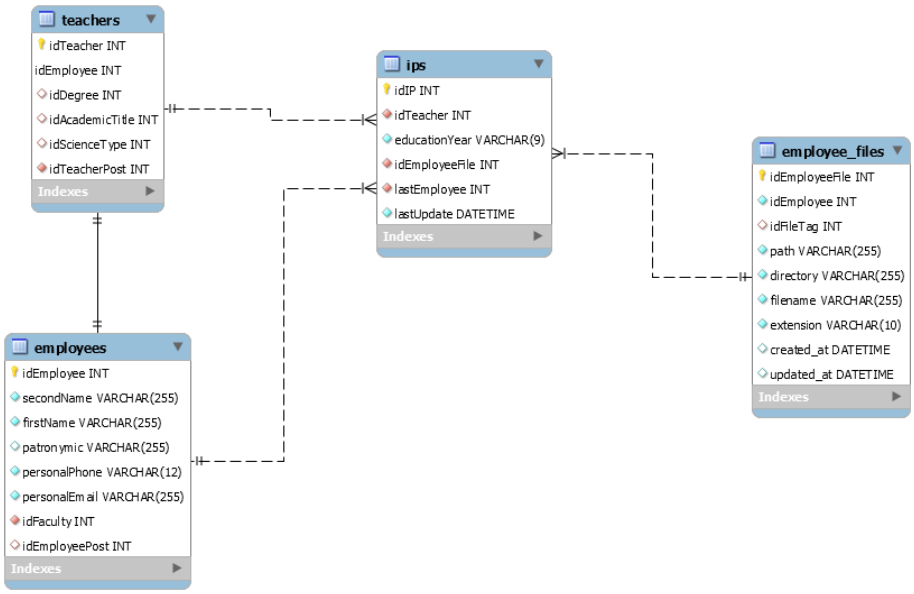

Рисунок 6 – Таблицы подсистемы индивидуальных планов

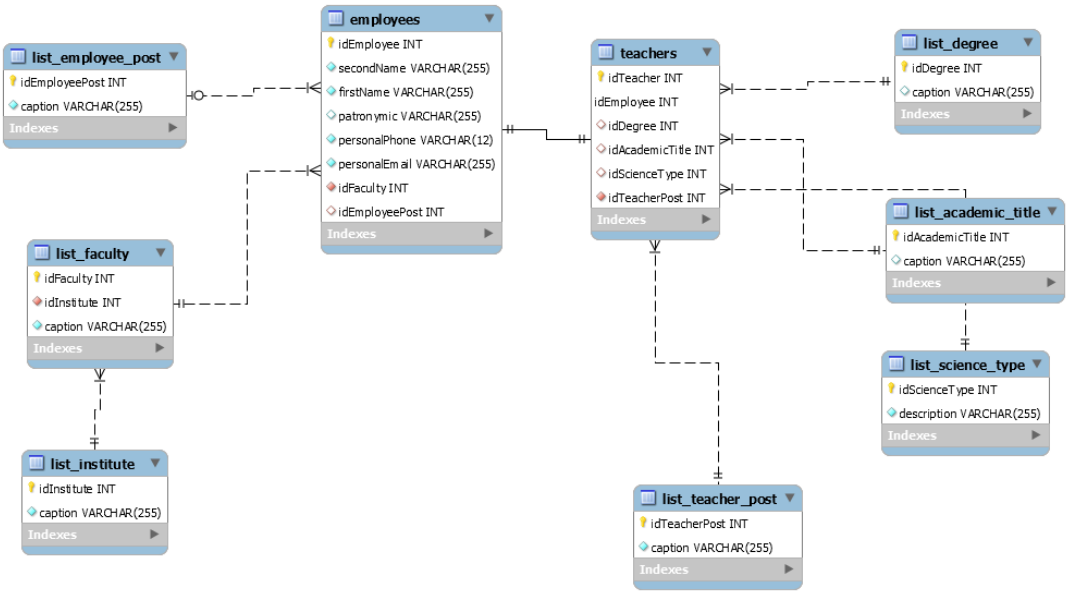

Рисунок 7 – Таблицы подсистемы кадров

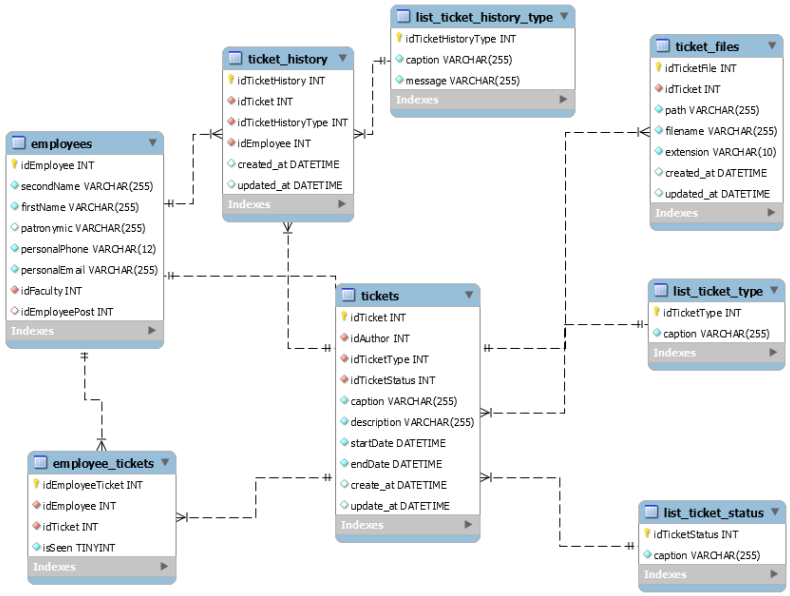

Рисунок 8 – Таблицы подсистемы поручений

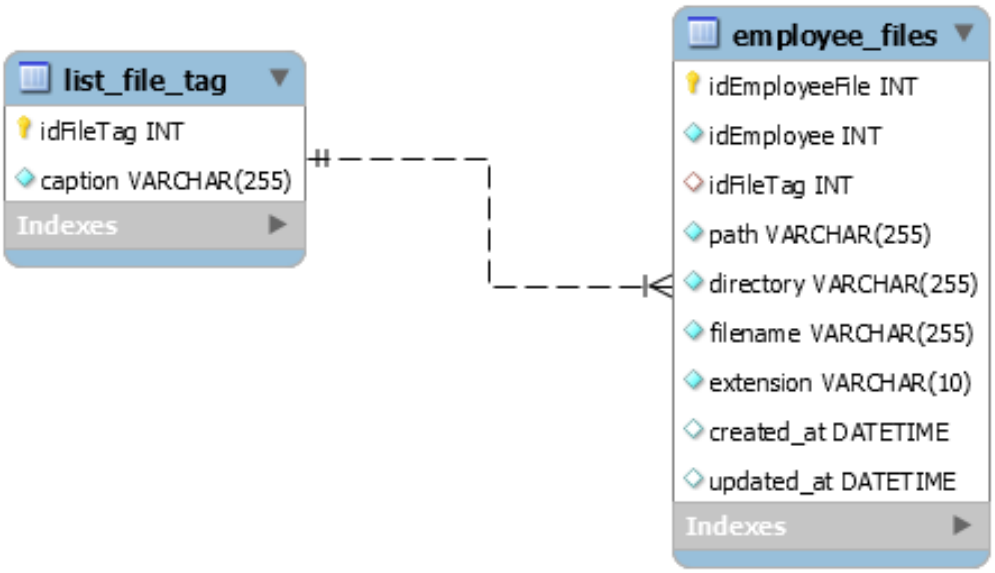

Рисунок 9 – Таблицы подсистемы хранения материалов

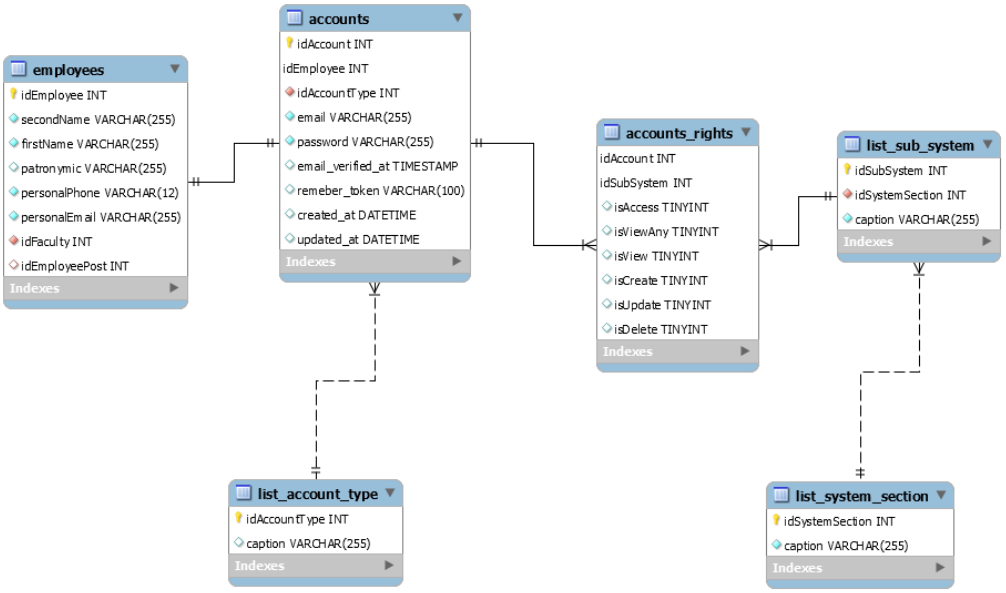

Рисунок 10 – Таблицы подсистемы пользователей и прав доступа

3.2 Проектирование классов

3.2.1 Модели

Для работы с таблицами непосредственно из кода, необходимо спроектировать так называемые модели, которые повсеместно используются во фреймворке.

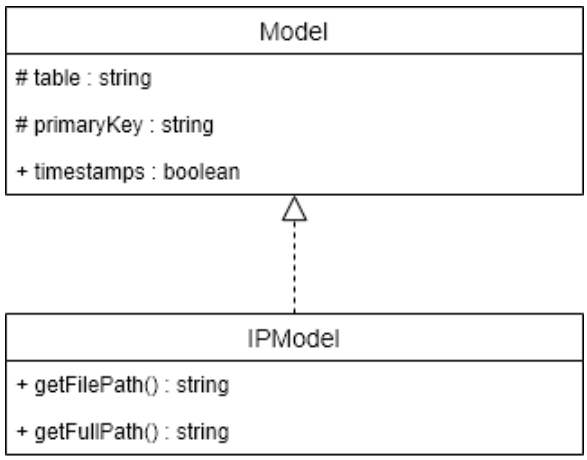

Рисунок 11 – Модели подсистемы индивидуальных планов

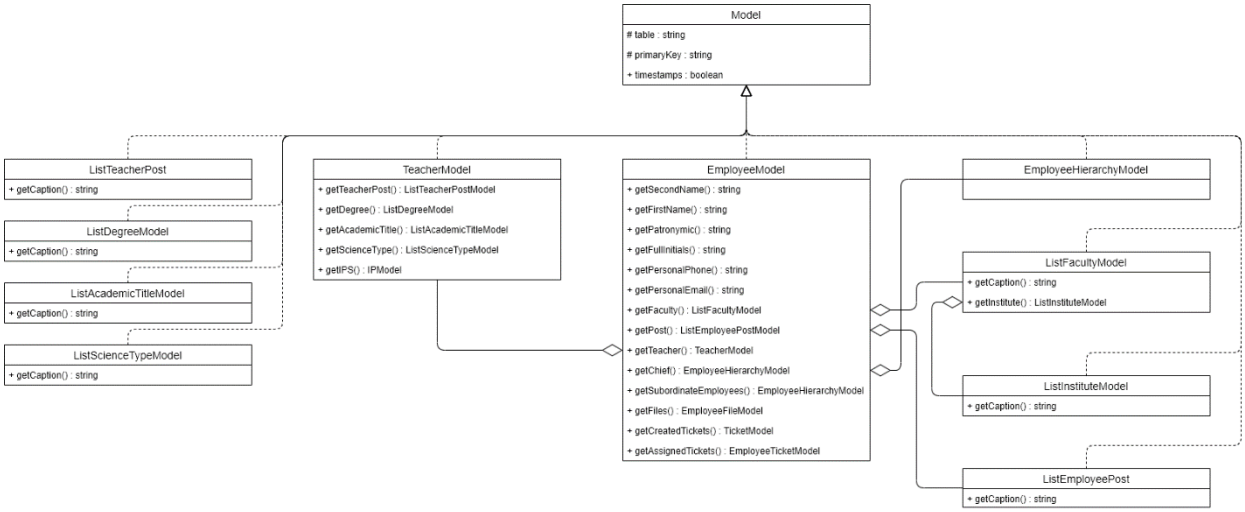

Рисунок 12 – Модели подсистемы кадров

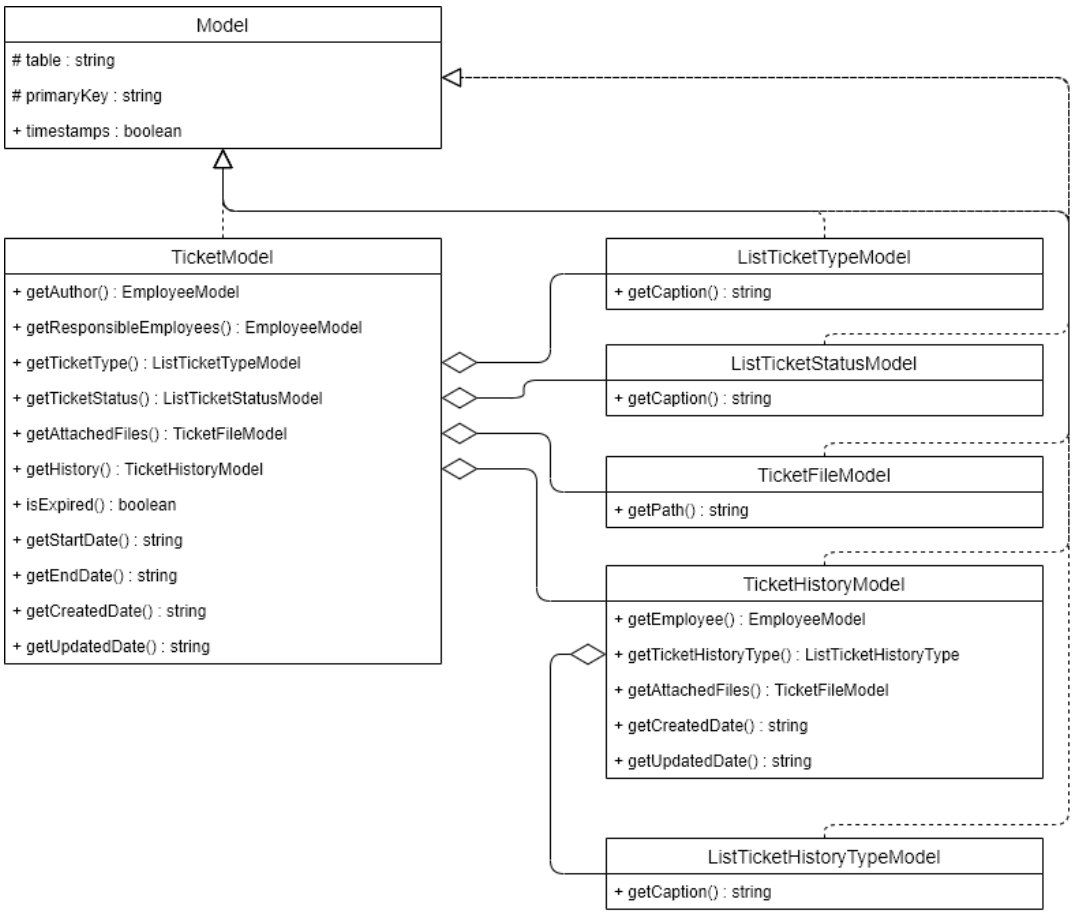

Рисунок 13 – Модели подсистемы поручений

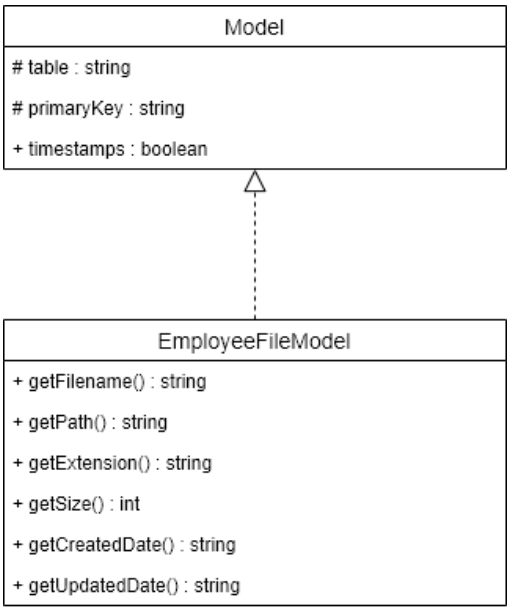

Рисунок 14 – Модели подсистемы хранения материалов

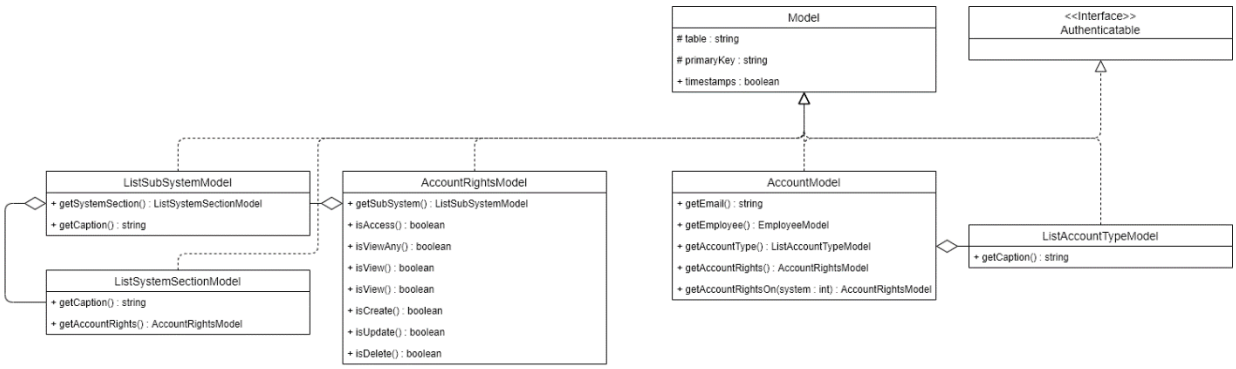

Рисунок 15 – Модели подсистемы пользователей и прав доступа

3.2.2 Контроллеры

Для реализации функционала различных подсистем необходимо спроектировать контроллеры.

Стоит разделить контроллеры на страничные и ресурсные. Страничные контроллеры отвечают за то, чтобы возвращать представления для различных страниц. Ресурсные контроллеры отвечают за реализацию CRUD для конкретной модели.

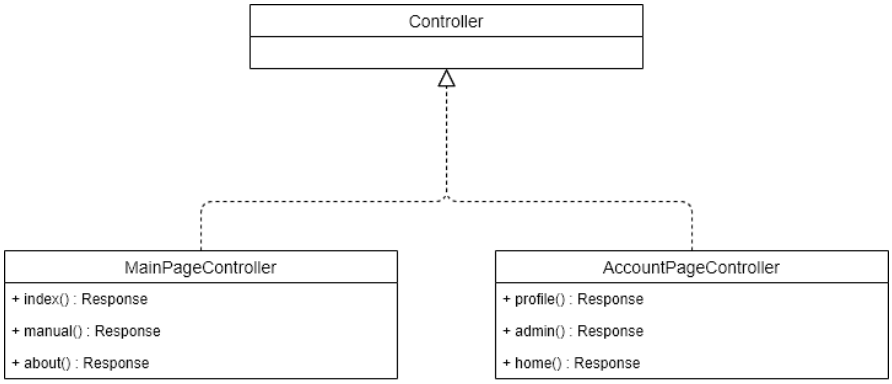

Рисунок 16 – Контроллеры

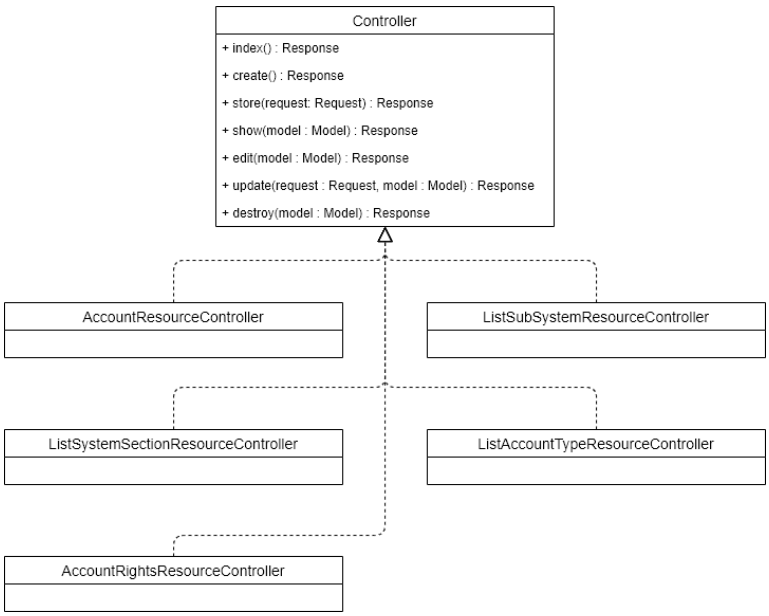

Рисунок 17 – Ресурсные контроллеры подсистемы пользователей

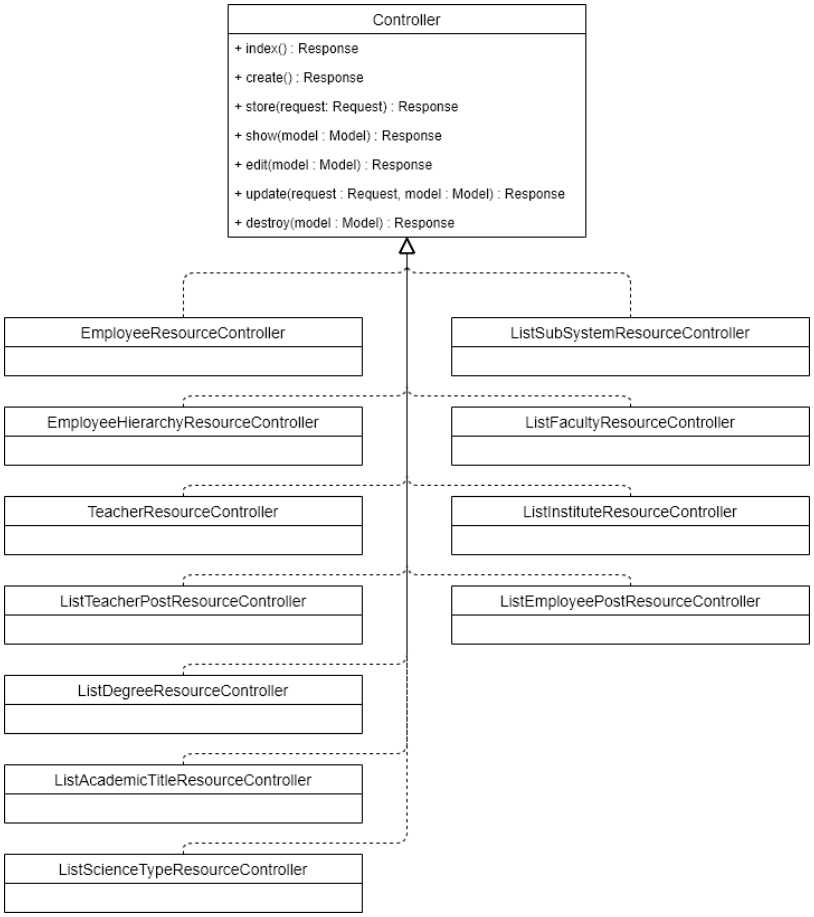

Рисунок 18 – Ресурсные контроллеры подсистемы кадров

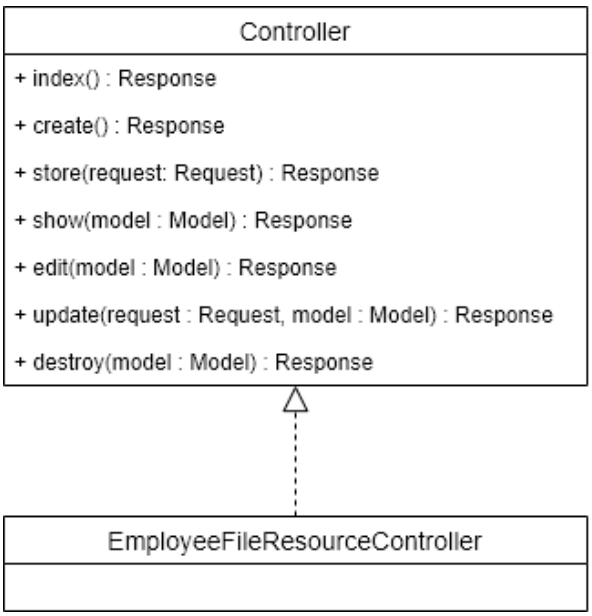

Рисунок 19 – Контроллер подсистемы хранения материалов

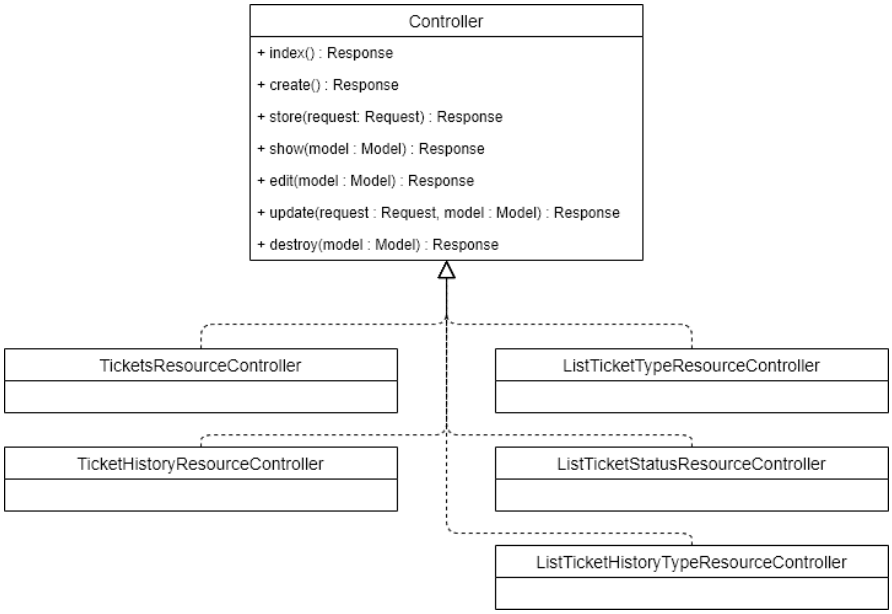

Рисунок 20 – Контроллеры подсистемы поручений

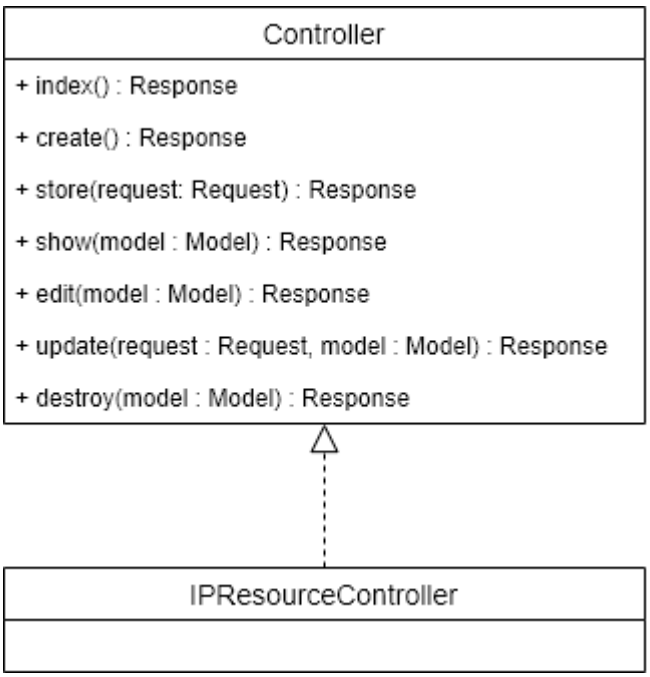

Рисунок 21 – Контроллер подсистемы индивидуальных планов

4 Реализация системы

4.1 Маршруты

Логика доступа к веб-страницам осуществляется посредством HTTP-маршрутов, которые описываются в файле web.php. Маршруты приведены на рисунке 22.

```php
<?php

use Illuminate\Support\Facades\Route;



/**
 * Маргруты главным страниц
 */
Route::get('/', 'MainPageController@index')->name('index');
Route::get('/manual', 'MainPageController@manual')->name('manual');
Route::get('/home', 'Service\Accounts\AccountPageController@home')->name('home');
Route::get('/profile','Service\Accounts\AccountPageController@profile')->name('profile');

Route::get('/login', function () { // #fixme Настроить класс LoginController или иной для автоматического редиректа на главную стра
    return redirect()->route('index');
});

/**
 * Маршруты связанные с подсистемами
 */
Route::resource('/ips', 'Main\IP\IPResourceController');
Route::get('/ips/download/{ip}', 'Main\IP\IPResourceController@downloadIP')->name('ips.download');

Route::resource('/files', 'Main\Storage\EmployeeFileResourceController');
Route::get('/files/download/{file}', 'Main\Storage\EmployeeFileResourceController@downloadFile')->name('files.downloadFile');
Route::post('/files_createDirectory', 'Main\Storage\EmployeeFileResourceController@createDirectory')->name('files.createDirectory')
Route::delete('/files_destroyDirectory', 'Main\Storage\EmployeeFileResourceController@destroyDirectory')->name('files.destroyDirect

Route::resource('/tickets', 'Main\Tickets\TicketResourceController');
Route::get('/tickets_inbox', 'Main\Tickets\TicketResourceController@inbox')->name('tickets.inbox');
Route::get('/tickets_expired', 'Main\Tickets\TicketResourceController@expired')->name('tickets.expired');
Route::get('/tickets/download/{file}', 'Main\Tickets\TicketResourceController@downloadFile')->name('tickets.downloadFile');

/**
 * Маршруты связанные с аутентификацией/авторизацией и выходом
 */
Route::post('/login', '\App\Http\Controllers\Auth\LoginController@login')->name('login');
Route::post('/logout', '\App\Http\Controllers\Auth\LoginController@logout')->name('logout');
```

Рисунок 22 – Маршруты приложения

4.2 Миграции

Laravel позволяет создавать файлы миграций, описывающие таблицы. В свою очередь это позволяет на этапе развёртывания сразу создавать базу данных, а также заполнять её данными.

Пример миграции приведён на рисунке 23.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateAccounts extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('accounts', function (Blueprint $table) {
            $table->bigIncrements('idAccount');
            $table->integer("idEmployee");
            $table->integer("idAccountType");
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('accounts');
    }
}
```

Рисунок 23 – Пример миграции таблицы «Accounts»

4.3 Сеялки

Laravel позволяет создавать, так называемые, сеялки, которые отвечают за наполнение базу данных информацией.

Пример сеялки приведён на рисунке 24.

```php
<?php

use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class AccountsSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $accounts = [];

        foreach (DataSeeder::$employees as $employee) {
            $accounts[] = [
                'idAccount' => $employee['idEmployee'],
                'idEmployee' => $employee['idEmployee'],
                'idAccountType' => $employee['idEmployeePost'],
                'email' => $employee['personalEmail'],
                'email_verified_at' => NULL,
                'password' => \Illuminate\Support\Facades\Hash::make('qwerty'),
                'remember_token' => NULL,
                'created_at' => '2020-01-01',
                'updated_at' => '2020-01-01'
            ];
        }

        DB::table('accounts')->insert($accounts);
    }
}
```

Рисунок 24 – Пример сеялки

4.4 Модели

Laravel активно использует паттерн MVC, в связи с этим создание моделей необходимо.

Пример модели представлен на рисунке 25.

```php

class AccountModel extends Authenticatable
{
    use Notifiable;

    protected $table = "Accounts";
    protected $primaryKey = "idAccount";

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'email', 'password', 'idEmployee', 'idAccountType'
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

    public function getEmail() {
        return $this->email;
    }

    public function getEmployee() {
        return $this->hasOne('App\Models\Main\Staff\EmployeeModel', 'idEmployee', 'idEmployee')->first();
    }

    public function getAccountType() {
        return $this->hasOne('App\Models\Service\Accounts\ListAccountTypeModel', 'idAccountType', 'idAccountType')->first();
    }

    public function getIdAccountType() {
        return $this->idAccountType;
    }

    public function getAccountRights() {
        $rights = $this->hasMany(AccountRightsModel::class, 'idAccount', 'idAccount')
            ->where('isAccess', '=', 1)
            ->whereNotIn('idSubSystem', [
                ListSubSystemConstants::Storage,
                ListSubSystemConstants::Tickets
            ])
            ->get();

        $groupRights = [];
        foreach ($rights as $right) {
            $groupRights[$right->getSubSystem()->getSystemSection()->getCaption()][] = $right;
        }

        return $groupRights;
    }

    public function getAccountRightsOn(int $systemId) {
        return $this->hasOne(AccountRightsModel::class, 'idAccount', 'idAccount')
            ->where('idSubSystem', '=', $systemId)
            ->first();
    }

}
```

Рисунок 25 – Пример модели «AccountModel»

4.5 Контроллеры

Создание контроллеров необходимо для поддержки MVC-паттерна. В связи с этим для каждой модели создаётся ресурсы контроллер, реализующий для неё CRUD.

Пример контроллера представлен на рисунке 26.

```php
class AccountResourceController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        return view ('systems.service.accounts.account_index', [
            'accounts' => AccountModel::all()
        ]);
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Contracts\View\Factory|View
     */
    public function create()
    {
        return view('systems.service.accounts.account_add', [
            'employees' => DB::table('accounts as a')
                            ->select('a.idAccount', 'e.idEmployee', 'e.secondName', 'e.firstName', 'e.patronymic')
                            ->rightJoin('employees as e', 'e.idEmployee', '=', 'a.idEmployee')
                            ->whereNull('a.idAccount')->get(),
            "accountTypes" => \App\Models\Service\Accounts\ListAccountTypeModel::all()
        ]);
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $data = $request->only(['accountType', 'employeeId', 'email', 'password']);

        $validator = Validator::make($data, [
            'email' => ['required', 'string', 'email', 'max:255', 'unique:accounts'],
            'password' => ['required', 'string', 'min:8'],
        ]);

        if (!$validator->fails()) {
            $user = new AccountModel();
            $user->idAccountType = $data['accountType'];
            $user->idEmployee = $data['employeeId'];
            $user->email = $data['email'];
            $user->password = Hash::make($data['password']);
            $user->save();

            Session::flash('message', 'Аккаунт успешно добавлен');
            return Redirect::route('accounts.create');
        } else {
            return Redirect::to('accounts.create')
                ->withErrors($validator)
                ->withInput(Input::except('password'));
        }
    }
```

Рисунок 26 – Пример контроллера «AccountResourceController»

4.6 Пользовательский интерфейс

Laravel использует для создания представлений или попросту страниц, шаблонизатор Blade.

В соответствии с этим, все страницы создаются как файлы с расширением blade.php и содержат HTML-вёрстку, а также специальные директивы для шаблонизатора.

Пример кода в шаблона представлен на рисунке 27.

```blade
<html>
    <head>
        <title>@yield('title')</title>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" type="text/css" href="{{ asset('css/semantic/semantic.css') }}">
        <link rel="stylesheet" type="text/css" href="{{ asset('css/main.css') }}">
        <script type="text/javascript" src="{{ asset("js/jquery.js")  }}"></script>
        <script type="text/javascript" src="{{ asset("js/semantic/semantic.js")  }}"></script>
    </head>
    <body>
        <div id="grid" class="ui stackable grid">
            <div class="two column row">
                <div id="menu" class="three wide column">
                    @include('layout.menu.menu_default')
                </div>
                <div id="content" class="thirteen wide column">

                    @if(\Illuminate\Support\Facades\Session::has('successMessage'))
                        <div class="ui icon success message">
                            <i class="info circle icon"></i>
                            <div class="content">
                                <div class="header">
                                    {{ \Illuminate\Support\Facades\Session::get('successMessage') }}
                                </div>
                            </div>
                        </div>
                    @endif

                    @if(\Illuminate\Support\Facades\Session::has('errorMessage'))
                        <div class="ui icon red message">
                            <i class="exclamation circle icon"></i>
                            <div class="content">
                                <div class="header">
                                    {{ \Illuminate\Support\Facades\Session::get('errorMessage') }}
                                </div>
                            </div>
                        </div>
                    @endif

                    @if(\Illuminate\Support\Facades\Session::has('message'))
                        <div class="ui icon success message">
                            <i class="check icon"></i>
                            <div class="content">
                                <div class="header">
                                    {{ \Illuminate\Support\Facades\Session::get('message') }}
                                </div>
                            </div>
                        </div>
                    @endif
                    @if(\Illuminate\Support\Facades\Session::has('error'))
                        <div class="ui icon red message">
                            <i class="close icon"></i>
                            <div class="content">
                                <div class="header">
                                    {{ \Illuminate\Support\Facades\Session::get('error') }}
                                </div>
                            </div>
                        </div>
                    @endif
                    @yield('content')
                </div>
            </div>
        </div>

        <script type="text/javascript">
            $('.ui.dropdown').dropdown();
            $('.ui.accordion').accordion();
            $('.menu .item').tab();
        </script>
    </body>
</html>
```

Рисунок 27 – Пример кода blade-шаблона

Пользовательский интерфейс представлен на рисунках 28-35.

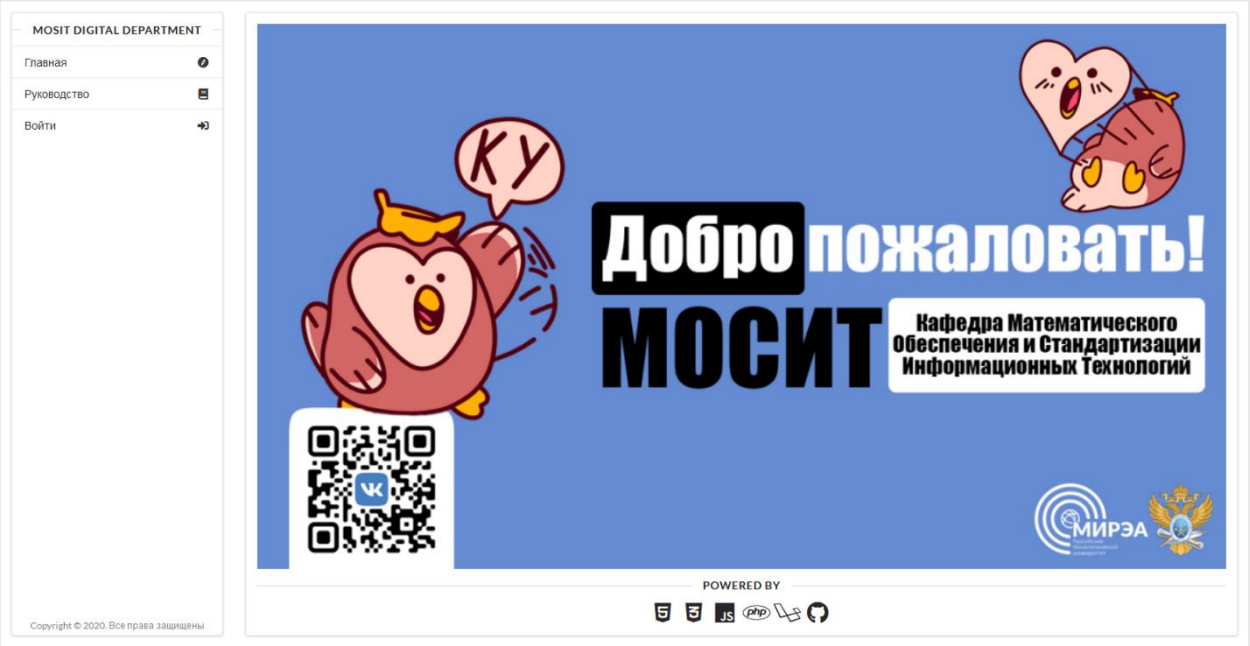

Рисунок 28 – Главная страница

Рисунок 29 – Рабочая область

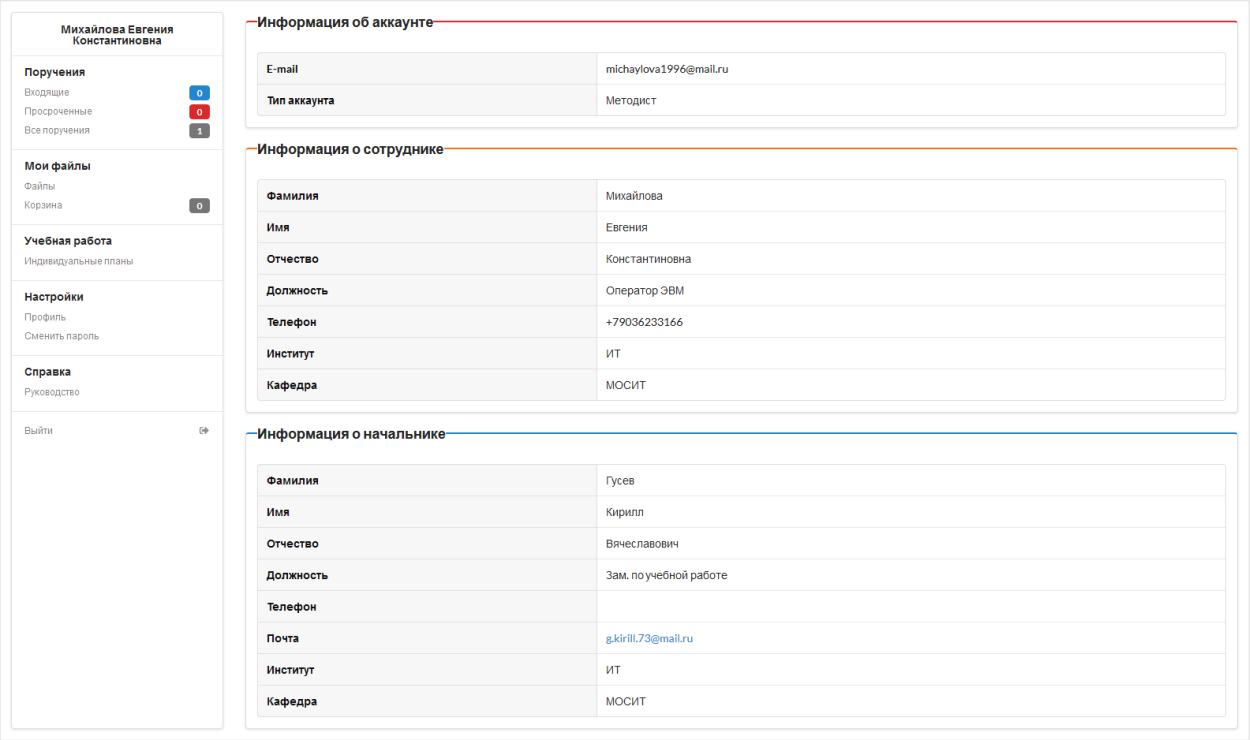

Рисунок 30 – Профиль пользователя

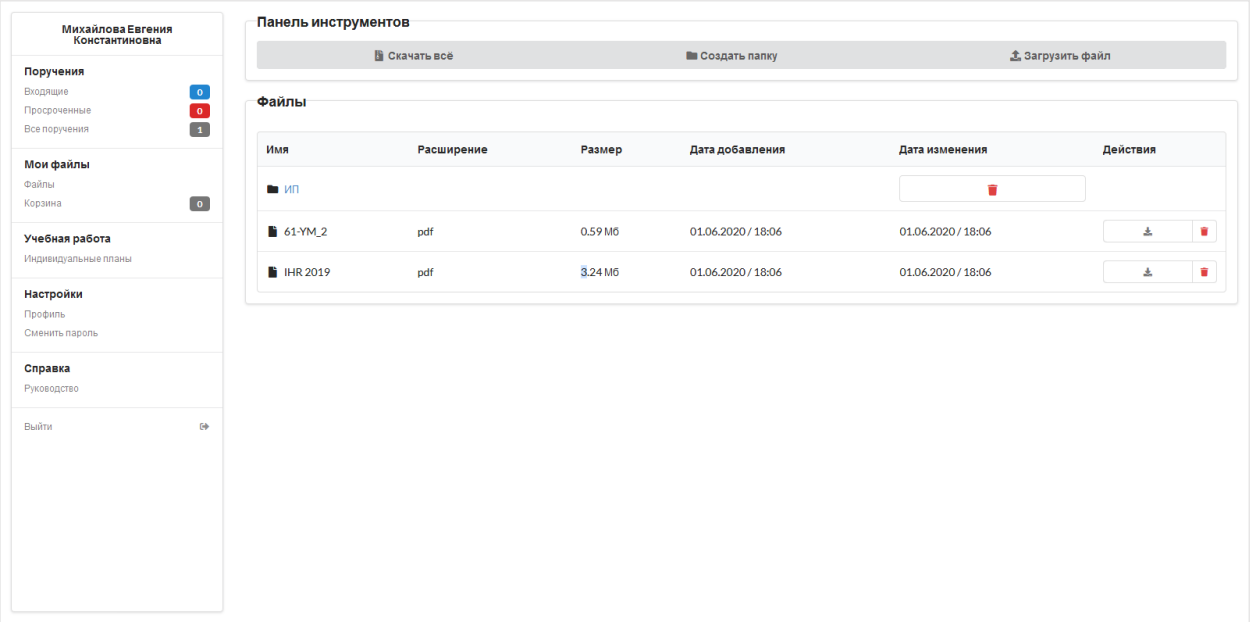

Рисунок 31 – Мои файлы

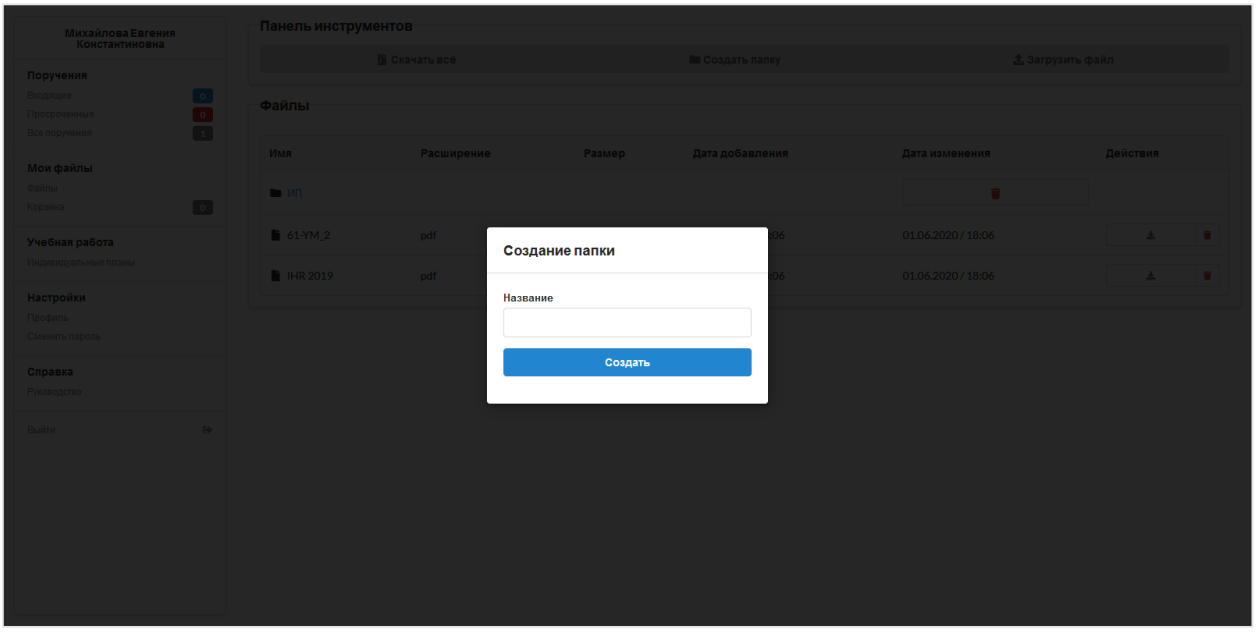

Рисунок 32 – Модальное окно «Создание папки»

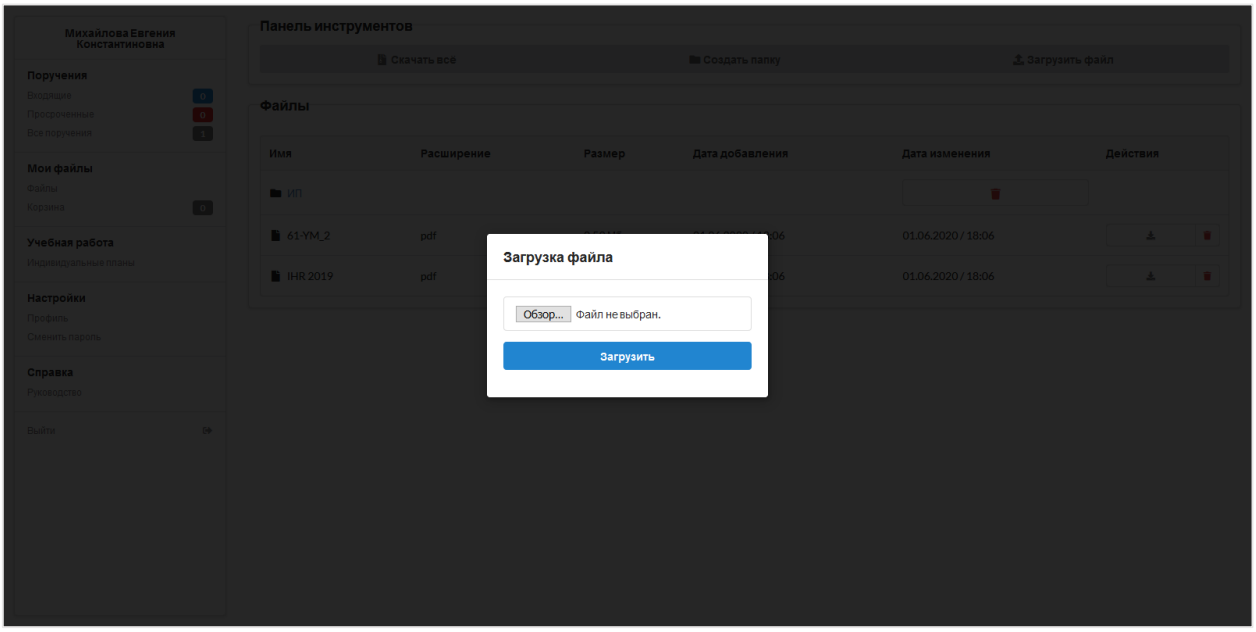

Рисунок 33 – Модальное окно «Загрузка файлов»

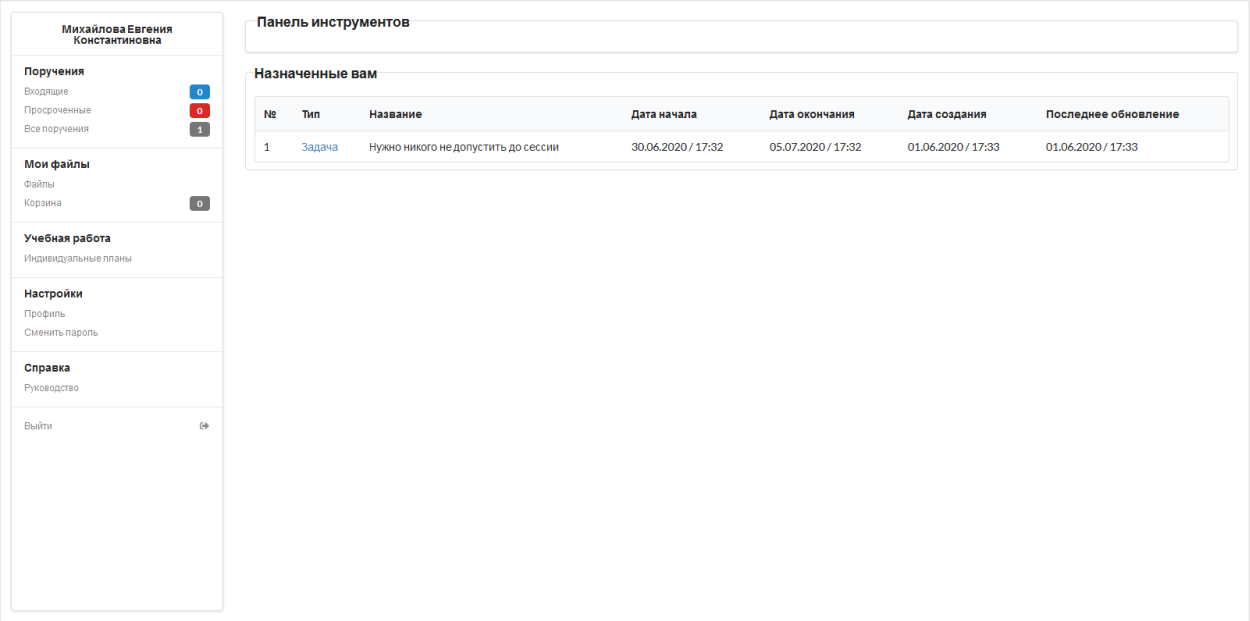

Рисунок 34 – Список всех поручений

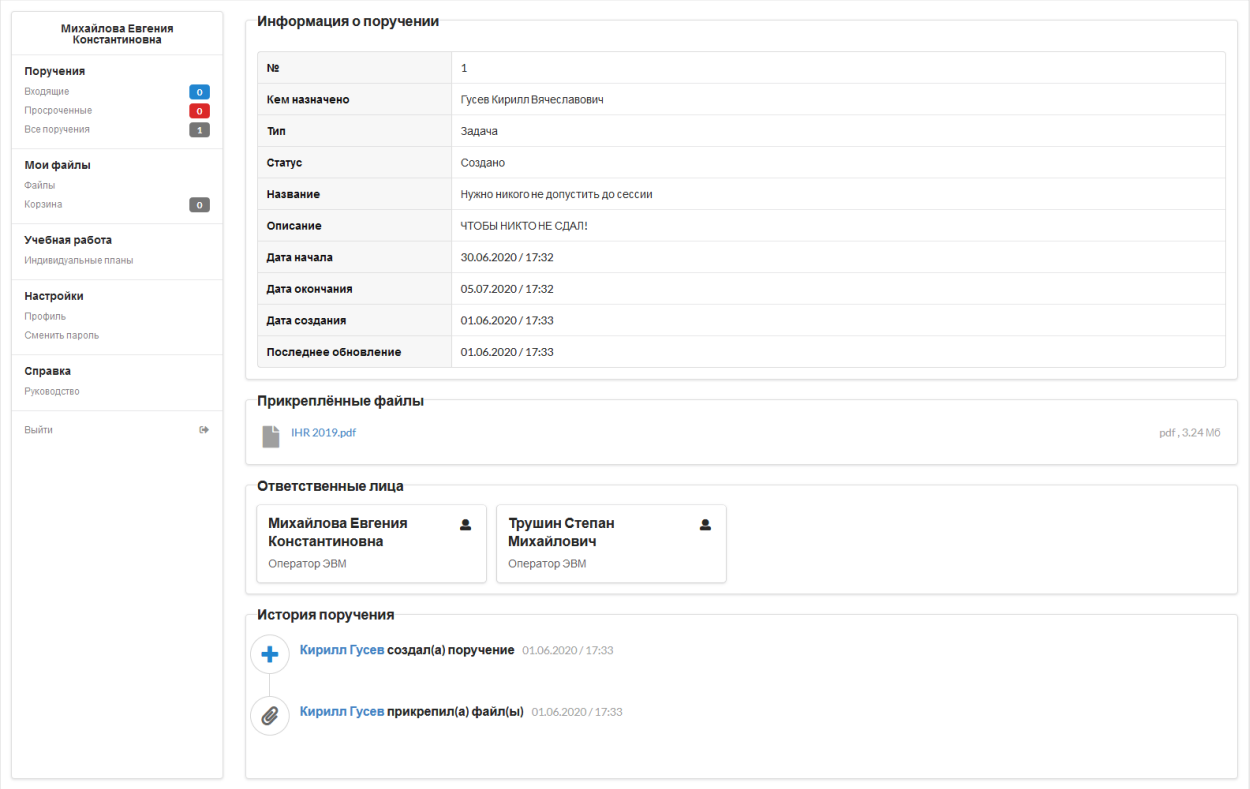

Рисунок 35 – Информация о поручении

4.3 Документация

Документация на исходный код была создана с помощью doxygen (Рисунок 36-37).

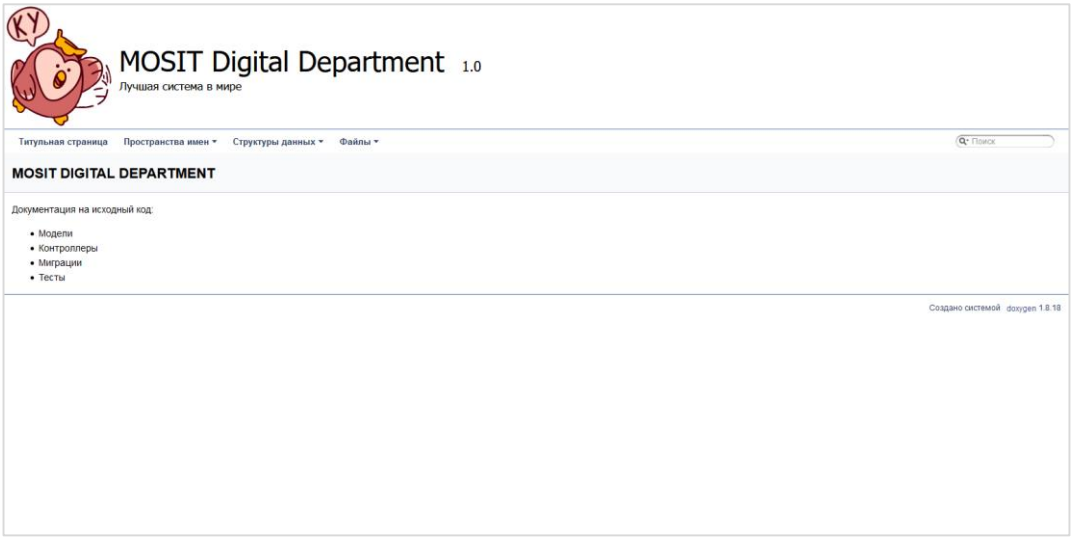

Рисунок 36 – Документация на исходный код

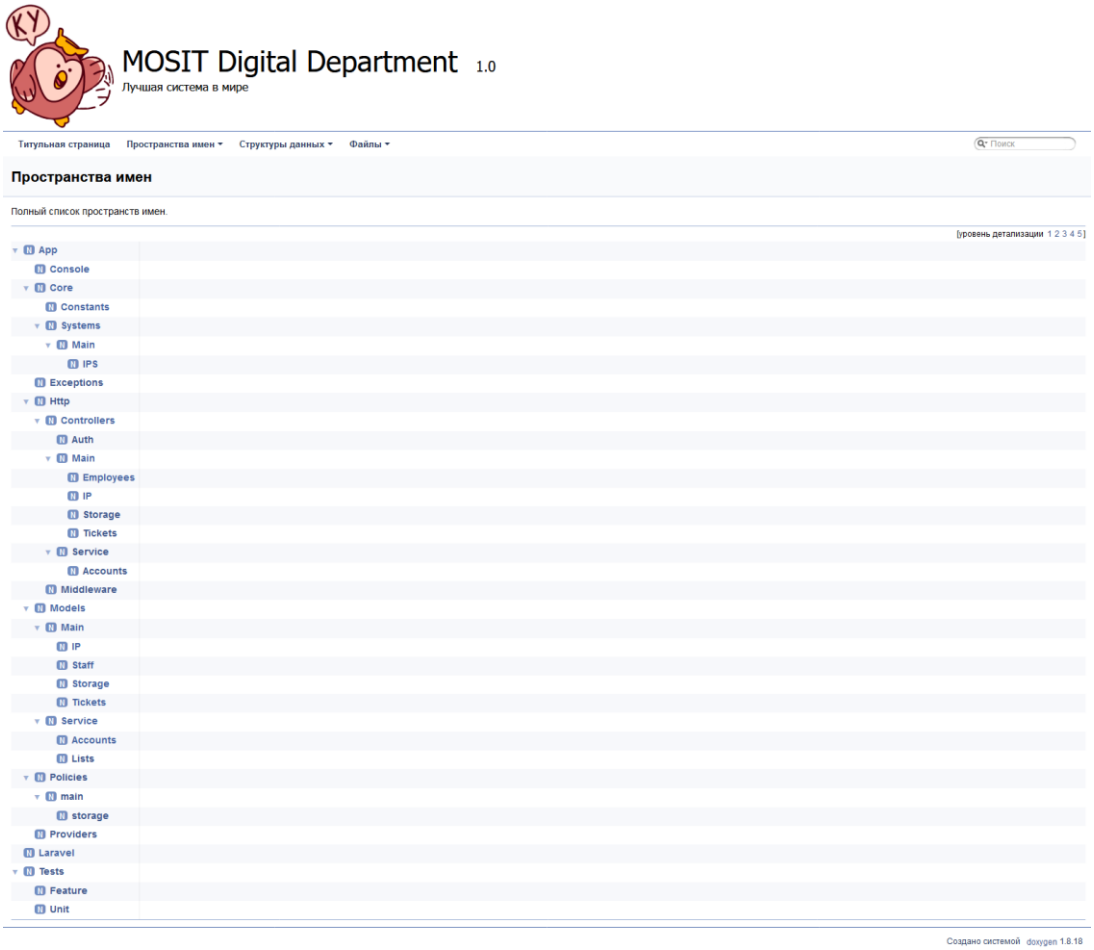

Рисунок 37 – Пространства имён