**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

|  |
| --- |
| Институт информационных технологий и управления в технических системах |
| (полное название института) |

|  |
| --- |
| кафедра «Информационные системы» |
| (полное название кафедры) |

**Пояснительная записка**

к курсовому проекту

по дисциплине «Проектный практикум»

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| на тему | **Веб-ориентирования система для поступления** | | | | | | | | | | | | | | | |
| **в российские университеты** | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Выполнил: студент | | | IV | | | курса, группы: | | | | | **ИС/б-17-2-о** | | | | | |
|  | | | | | | | | | | | | | | | | |
| Направления подготовки (специальности) | | | | | | | | | | | 09.03.02 | | | | | |
|  | | | | | | | | | | | | | | | | |
| Информационные системы и технологии | | | | | | | | | | | | | | | | |
| (код и наименование направления подготовки (специальности)) | | | | | | | | | | | | | | | | |
| профиль (специализация) | | | | | |  | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| **Черняев Никита Георгиевич** | | | | | | | | | | | | | | | | |
| (фамилия, имя, отчество студента) | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Руководитель | | | Кузнецов С.А. | | | | | | | | | | | | | |
|  | | | (фамилия, инициалы, степень, звание, должность) | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
| Защита « | |  | | » |  | | | | 20 | | 21 | г. | | Оценка | |  |
|  | | | | | | | | | | | | | | | | |
| Руководитель | | | | | | |  | | | | | |  | |  | |
|  | | | | | | | (подпись) | | | | | |  | | (инициалы, фамилия) | |
|  | | | | | | | | | | | | | | | | |
| Ведущий преподаватель | | | | | | |  | | | | | |  | | Кузнецов С.А. | |
|  | | | | | | | (подпись) | | | | | |  | | (инициалы, фамилия) | |
|  | | | | | | | | | | | | | | | | |
| 20 | | | | | | | | 21 | | г. | | | | | | |

**АННОТАЦИЯ**

В данной пояснительной записке представлено описание основных этапов выполнения курсового проекта по дисциплине «Проектный практикум». Курсовой проект посвящен разработке WEB-сервиса для поступления в российские университеты. Приведено техническое задание, по которому реализован данный проект.

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ**

**1 Цели создания и целевая аудитория**

Цели создания сайта:

* Предоставление услуги подбора вуза по направлению, уровню образования и т.д.;
* Предоставить последние новости и материалы связанные с поступлением в университет
* Проведение кандидата по жизненному циклу поступления.

Целевой аудиторией сайта являются клиенты, заинтересованные в получении образования в российских университетах, путем подачи заявления онлайн.

**2 Структура сайта**

В разрабатываемом Web-приложении предполагается создание трех независимых интерфейсов пользователей: общий интерфейс, личный кабинет менеджера.

Общий интерфейс содержит:

* главную страницу с описанием основной информации о сервисе, новостями и основными материалами;
* форма входа для менеджера;
* всплывающее окно с информацией об университете;

Личный кабинет менеджера содержит:

* страницу для просмотра кандидатов;
* страницу с университетами, где отображается информация о текущем приема;
* страницу со странами, где будет отображаться сколько желающих на обучения из каких стран;
* страницу администрирования;
* страницу просмотра личного кабинета кандидата с информацией о нем.

**3. Пожелания по сайту**

Все страницы разрабатываемого сайта должны быть выдержаны в одном стиле. Корпоративные цвета: белый, синий, серый. Ширина: 1400px.

**4. Технические требования к сайту**

Сайт должен работать на основных современных браузерах (Google Chrome, Safari, Opera, Mozilla Firefox). Кроме того, сайт должен быть адаптивным: работать на устройствах с любым разрешением экрана (компьютерах, смартфонах, планшетах).

Инструменты для разработки:

Scala, Play framework, PostgreSQL, Squeryl, Liquibase.

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ ……………………………………………………………………….6

1. ПОСТАНОВКА ЗАДАЧИ ……………………………………………………....7

2. АНАЛИЗ ПРОБЛЕМНОЙ СИТУАЦИИ И ОБЗОР АНАЛОГОВ …………....8

2.1. Описание ситуации ………………………………………………………..8

2.2. Обзор аналогов ……………………………………………………………..8

3. ПЛАН РЕАЛИЗАЦИИ ПРОЕКТА …………………………………………….9

4. ВЫБОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ ……………………………....10

5. ОПИСАНИЕ РЕЗУЛЬТАТА …………………………………………………..12

5.1. Реализация интерфейса …………………………………………………..12

5.2. Разработка программных модулей ……………………………………....22

6. ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО САЙТА …………………………...36

ЗАКЛЮЧЕНИЕ ………………………………………………………………….39

СПИСОК ЛИТЕРАТУРЫ И ИНФОРМАЦИОННЫХ РЕСУРСОВ …………..40

ПРИЛОЖЕНИЕ А. КОД ПРИЛОЖЕНИЯ ……………………………………...41

**ВВЕДЕНИЕ**

Данная работа состоит из пояснительной записки, включающей в себя аннотацию, содержание, введение, шесть разделов, заключение, список использованных источников и приложения.

В первом разделе подробно описывается постановка задачи.

Во втором разделе был проведен анализ проблемной ситуации и обзор аналогов. Были выявлены проблемы, которые нас сервис может решить.

В третьем разделе описаны этапы работы на протяжении семестра.

В четвертом разделе были выбраны и описаны инструментальные средства, используемые при разработке сервиса.

В пятом разделе находится описание продуктового результата. Здесь описано состояние проекта на момент завершения семестра.

В шестом разделе было произведено тестирование разработанного сервиса на наличие валидации и правильной работы таких функций, как: авторизация/регистрация, заполнение анкеты пациента и заявки на врача, поиск врача, запись на прием.

**1 ПОСТАНОВКА ЗАДАЧИ**

Определим основные функциональные требования для приложения:

* разрабатываемый WEB-сервис для поступления в российские университеты состоять из основных частей: публичные страницы и личные кабинеты менеджера/кандидата.
* необходимо реализовать авторизацию и регистрацию нового менеджера;
* необходимо предоставить возможность менеджеру добавлять кандидатов;
* необходимо реализовать админ панель;
* для управления версиями исходного кода проекта, а также для упрощения групповой разработки, необходимо использовать систему контроля версий Git;
* разработанный интерфейс сервиса должен проходить тест на кроссбраузерность и адаптивность на разных устройствах;
* структура публичной части сайта: главная страница, личный кабинет менеджера, личный кабинет кандидата, страницы просмотра университетов, страницы для администратора;

**2 АНАЛИЗ ПРОБЛЕМНОЙ СИТУАЦИИ И ОБЗОР АНАЛОГОВ**

## 2.1 Описание ситуации

Во всем мире имеется много людей, которые хотят обучаться в России. Также имеется много компаний и организаций, помогающих в поступлении кандидатам. Разрабатываемый сервис будет служить помощником для таких организаций, чтобы они могли вести кандидата через жизненный цикл поступления онлайн.

## 2.2 Обзор аналогов

В процессе анализа проблемной ситуации был проведен обзор существующих аналогов, среди которых выделены сервисы, представленные ниже.

Study in Russia (https://studyinrussia.ru/):

1. Очень медленная работа сайта.

2. Интуитивно непонятный интерфейс.

Ассоциация Гуманитарного Сотрудничества (https://rustudent.org/):

1. Маленький подбор университетов.

2. Отсутствие адаптивности

**3 ПЛАН РЕАЛИЗАЦИИ ПРОЕКТА**

Для определения времени осуществления мероприятий, направленных на достижение целей проекта, и для установления взаимосвязей между ними по временному параметру с учётом наиболее рисковых событий, составляется календарный план проекта. Календарное планирование заключается в создании и последующем уточнении расписания, которое учитывает состав работ, риски, ограничения. Поскольку календарный план в виде перечня исключительно плановых параметров работ без сравнения с фактическими сроками выполнения утрачивает свой смысл, нередко, вместо календарного плана, применяют название календарного графика.

В таблице 3.1 показан план реализации проекта, где описаны этапы проекта, сроки выполнения, результаты, участники проекта и задействованные материально-технические ресурсы.

Таблица 3.1 – План реализации проекта

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Этапы проекта / конкретные мероприятия, детализирующие этапы | Cрок выпол-нения | Результат | Участники проекта | Задействован-ные материально-технические  ресурсы |
| Этап 1 | Проектирование БД | 10.02.21 – 25.02.21 | draw.io-файл со схемой БД | Черняев Н.Г. | Draw.io  PostgreSQL,  Squeryl |
| Этап 2 | Разработка дизайна | 25.02.21 – 03.03.21 | Макет в Figma | Черняев Н.Г. | Figma,  креатив |
| Этап 3 | Верстка публичной части сайта | 03.03.21 – 23.03.21 | Html, CSS, JS-файлы | Черняев Н.Г. | Время и психическое здоровье |
| Этап 4 | Разработка функционала авторизации | 23.03.21 – 25.03.21 | Функционал для сервиса | Черняев Н.Г. | Время и психическое здоровье |
| Этап 5 | Разработка функционала личных кабинетов | 26.03.21-04.04.21 | Функционал ЛК | Черняев Н.Г. | Время и психическое здоровье |
| Этап 6 | Разработка административной части | 04.04.21-05.04.21 | Админ. часть сайта | Черняев Н.Г. | Время и психическое здоровье |

**4 ВЫБОР ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ**

В ходе работы на стороне клиента было решено использовать язык JavaScript, CSS, для верстки использовать встроенный scala-шаблоны во фреймворк Play. На серверной части в качестве СУБД был выбран PostgreSQL, в качестве ORM - Squeryl, в качестве СУВБД - Liquibase, фреймворк - Play, язык - Scala.

**Play -** каркас разработки с открытым кодом, написанный на Scala и Java, использует паттерн проектирования Model-View-Controller (MVC). Нацелен на повышение производительности, используя договорённости перед конфигурацией, горячую перегрузку кода и отображения ошибок в браузере. Разработку Play вдохновили такие каркасы как Ruby on Rails и Django.

**Scala** - это современный мультипарадигмальный язык программирования, разработанный для выражения общих концепций программирования в простой, удобной и типобезопасной манере. Элегантно объединяя особенности объектно-ориентированных и функциональных языков.

Scala - это чистый объектно-ориентированный язык в том смысле, что [каждое значение - это объект](https://docs.scala-lang.org/ru/tour/unified-types.html). Типы и поведение объектов описаны в [классах](https://docs.scala-lang.org/ru/tour/classes.html) и [трейтах](https://docs.scala-lang.org/ru/tour/traits.html)(характеристиках объектов). Классы расширяются за счет механизма наследования и гибкого [смешивания классов](https://docs.scala-lang.org/ru/tour/mixin-class-composition.html), который используется для замены множественного наследования.

Scala также является функциональным языком в том смысле, что [каждая функция - это значение](https://docs.scala-lang.org/ru/tour/unified-types.html). Scala предоставляет [легкий синтаксис](https://docs.scala-lang.org/ru/tour/basics.html) для определения анонимных функций, поддерживает [функции высшего порядка](https://docs.scala-lang.org/ru/tour/higher-order-functions.html), поддерживает [вложенные функции](https://docs.scala-lang.org/ru/tour/nested-functions.html), а также [каррирование](https://docs.scala-lang.org/ru/tour/multiple-parameter-lists.html). Scala имеют встроенную поддержку алгебраических типов данных, которые используются в большинстве функциональных языках программирования (эта поддержка базируется на механизме [сопоставления с примером](https://docs.scala-lang.org/ru/tour/pattern-matching.html), где в качестве примера выступают [классы образцы](https://docs.scala-lang.org/ru/tour/case-classes.html) ). [Объекты](https://docs.scala-lang.org/ru/tour/singleton-objects.html) предоставляют удобный способ группировки функций, не входящих в класс.

**Squeryl -** Scala ORM и DSL для взаимодействия с базами данных с минимальной детализацией и максимальной безопасностью типов.

**Liquibase** - открытая (open source) система для управления миграциями БД. Liquibase помогает организовать процесс внесения изменений в схему БД, каждая миграция будет содержать описание изменений, необходимых для перехода от старой ревизии к новой.

При использовании Liquibase изменения структуры базы данных будут храниться в отдельных файлах (changelogs), поддерживаются форматы XML, YAML, JSON или SQL, что очень удобно. Изменения можно хранить в одном или множестве файлов с последующем включением в основной файл. Второй вариант предпочтительнее, т.к. позволяет гибко организовать применение и хранение чейнджлогов.

**PostgreSQL** — это популярная свободная объектно-реляционная система управления базами данных. PostgreSQL базируется на языке SQL и поддерживает многочисленные возможности.

СУБД отличается высокой надежностью и хорошей производительностью. PostgreSQL поддерживает транзакции (ACID), репликация реализована встроенными механизмами. При этом система расширяемая — можно создавать свои типы данных и индексов, а также расширять поведение при помощи языков программирования.

**5 ОПИСАНИЕ РЕЗУЛЬТАТА**

**5.1 Реализация интерфейса**

При запуске web-приложения открывается главная страница, на которой представлена возможность отслеживания заявления (Рисунок 2), имеется секция с новостями (Рисунок 3) и раздел с материалами (Рисунок 4).

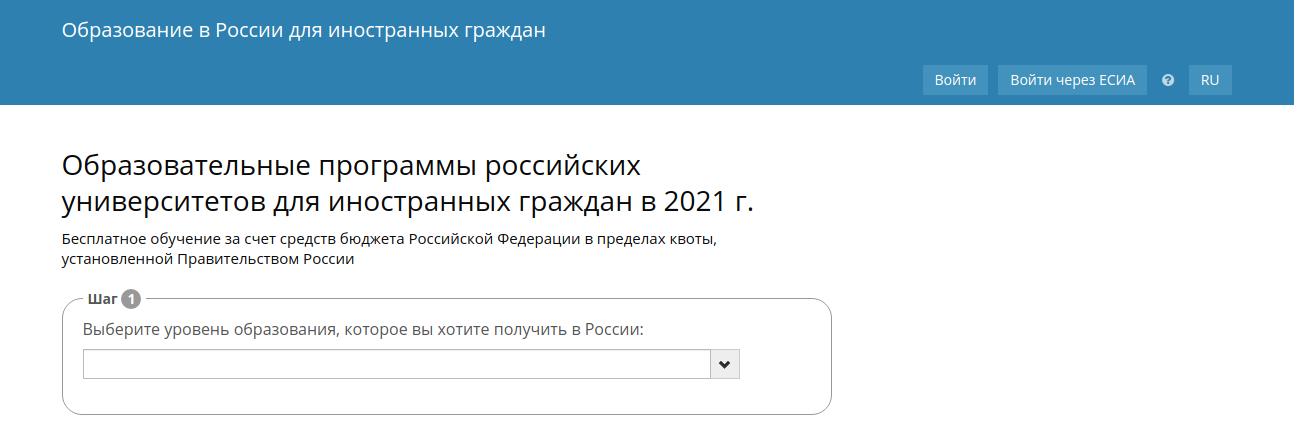


Рисунок 1 – Главный экран

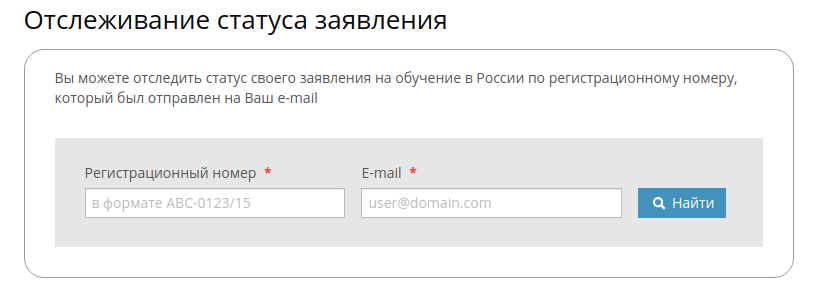


Рисунок 2 – Секция с отслеживанием заявления

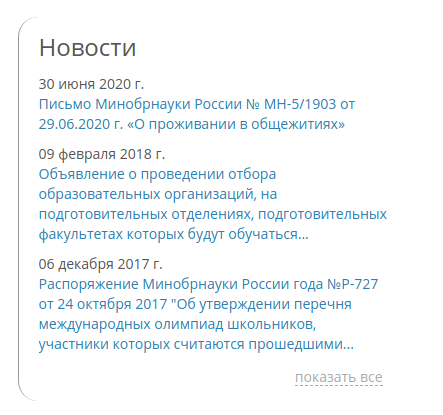


Рисунок 3 – Блок новостей

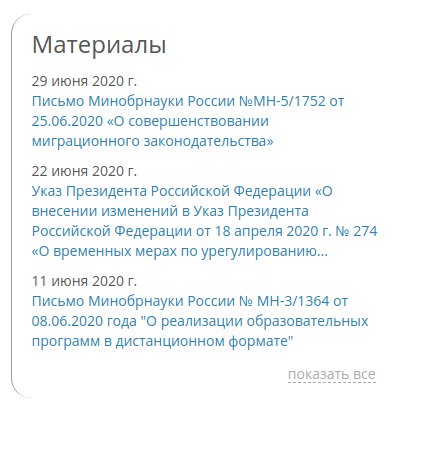


Рисунок 4 – Блок материалов

При нажатии на кнопку «Войти» открывается страница с формой входа состоящие из полей «Email» и «Пароль». Так же на этом окне находится кнопка восстановления пароля (Рисунок 6).

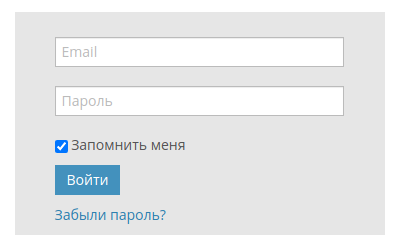


Рисунок 5 – Страница входа

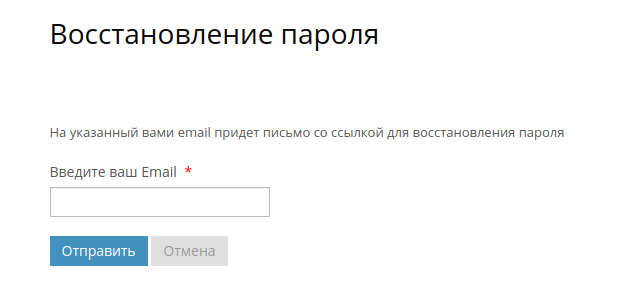


Рисунок 6 – Страница восстановления пароля

После авторизации менеджера, открывается его личный кабинет (Рисунок 7), а также появляется меню администратора (Рисунок 8)



Рисунок 7 – Личный кабинет

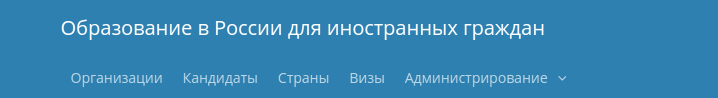


Рисунок 8 – Меню администрирования

Имеется возможность в личном кабинете отредактировать личную информацию (Рисунок 9) и контактные данные (Рисунок 10).

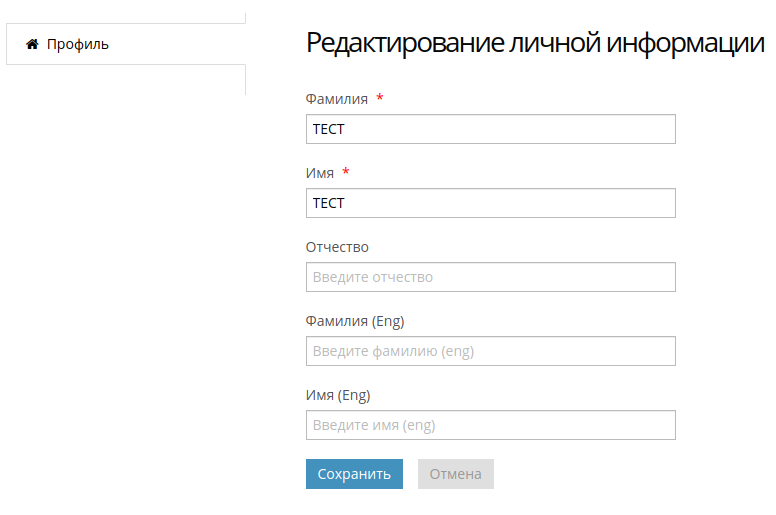


Рисунок 9 - Редактирование личной информации

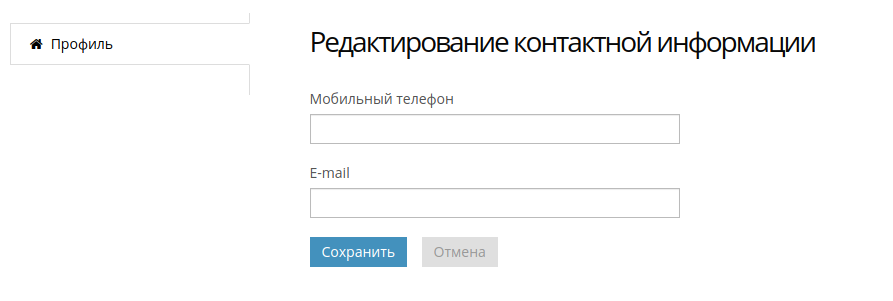


Рисунок 10 – Редактирование контактных данных

При открытии страницы Организации отображается список всех сотрудничающих организаций (Рисунок 11), а также фильтр по ним (Рисунок 12).

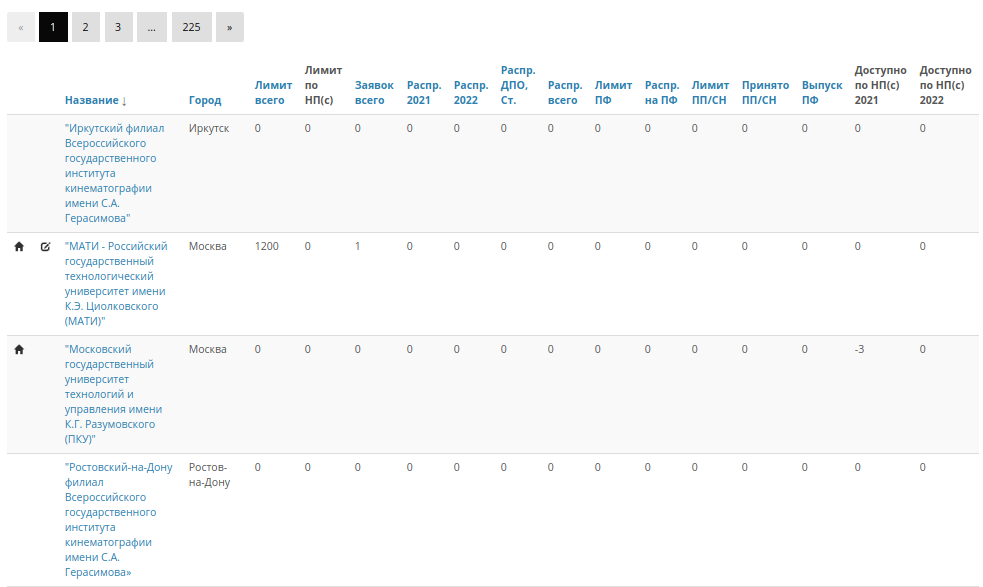


Рисунок 12 – Список организаций

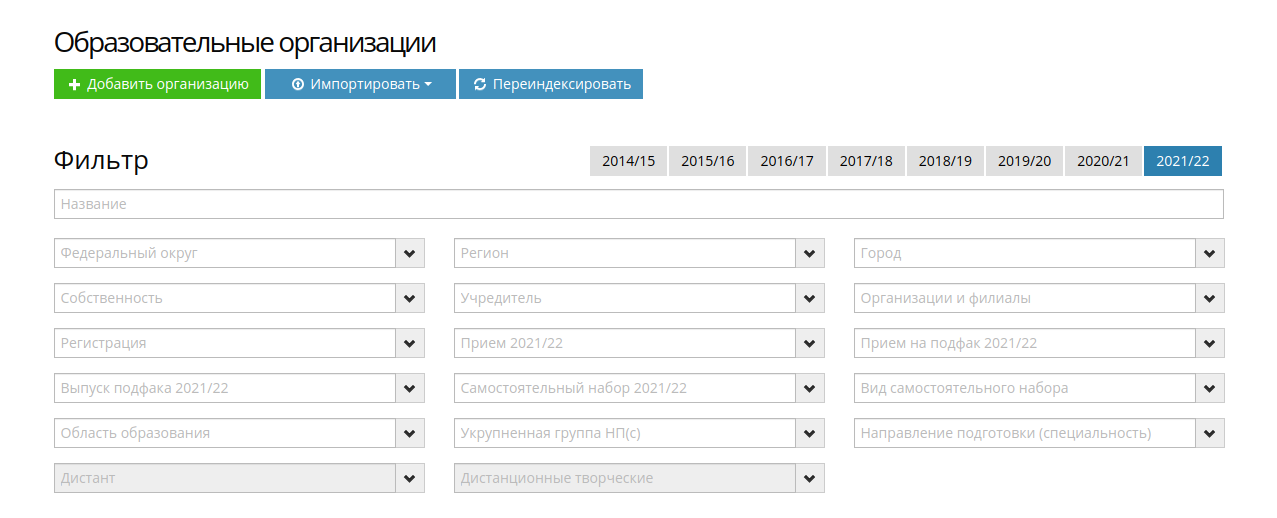


Рисунок 13 – Фильтры по организациям

Имеется возможность добавить организацию (Рисунок 14) и перейти в карточку организации (Рисунок 15).

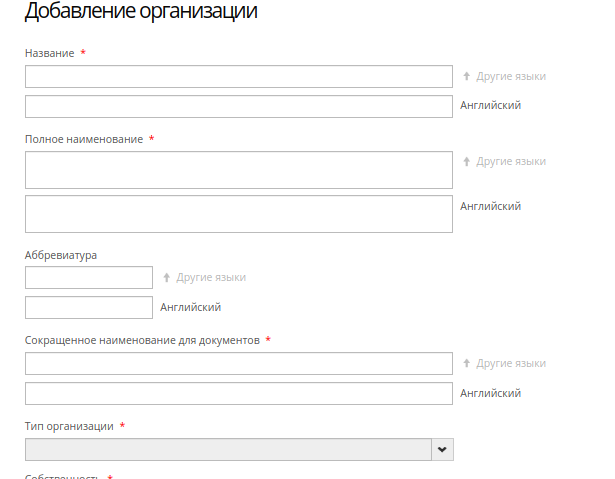


Рисунок 14 – Добавление организации

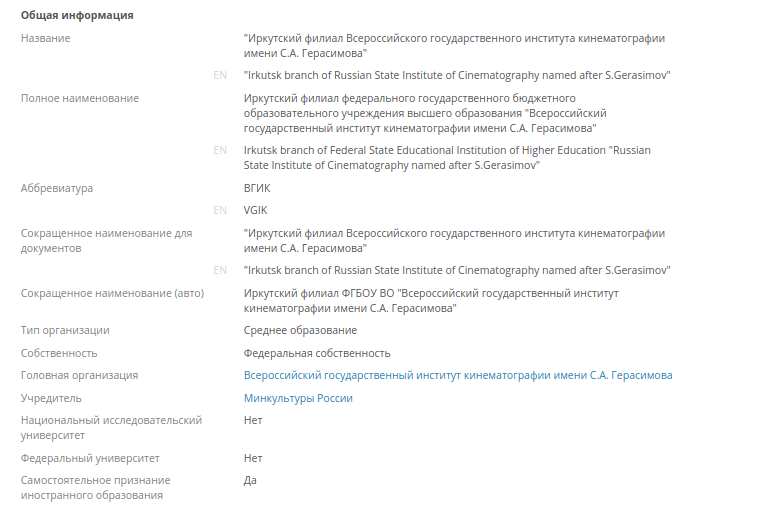


Рисунок 15 –Карточка организации

На рисунках 16-17 представлена страница с кандидатами и фильтрами по ним.

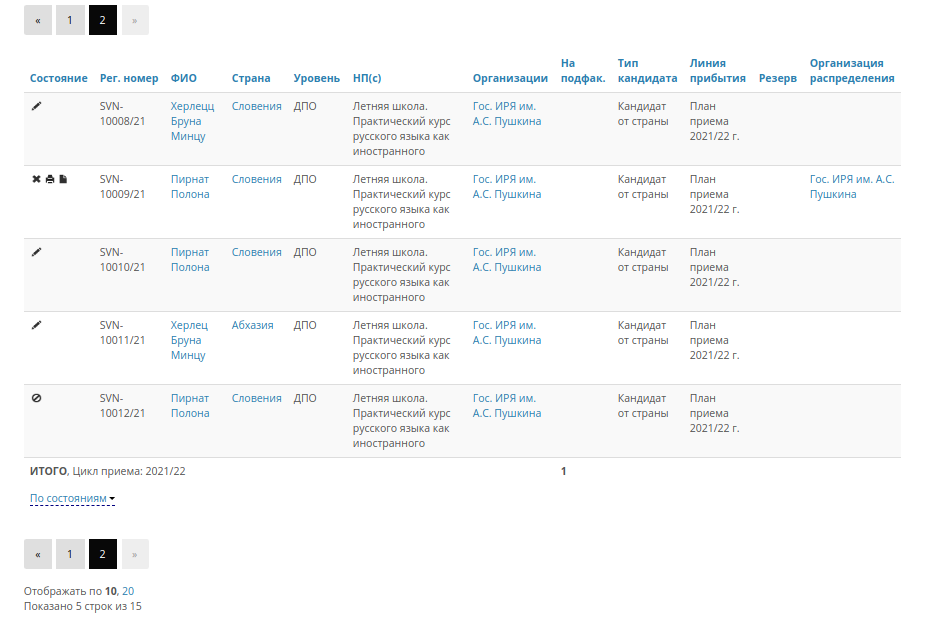


Рисунок 16 – Список кандидатов

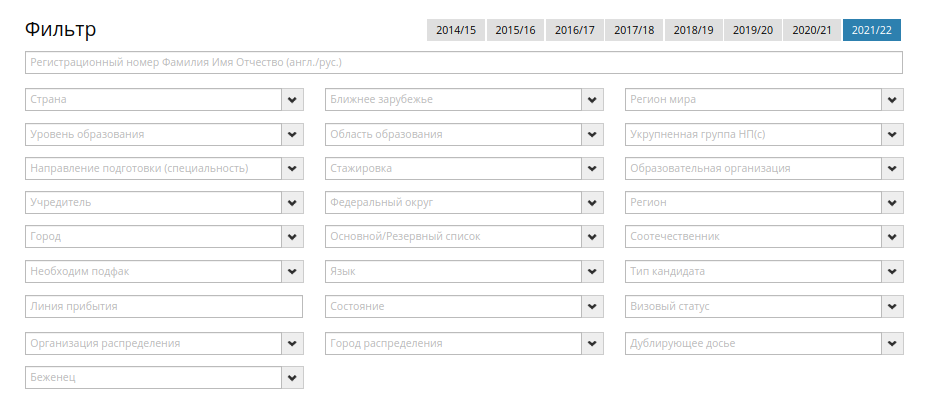


Рисунок 17 – Фильтр по кандидатам

На Рисунках 18-19 отображена карточка кандидата.

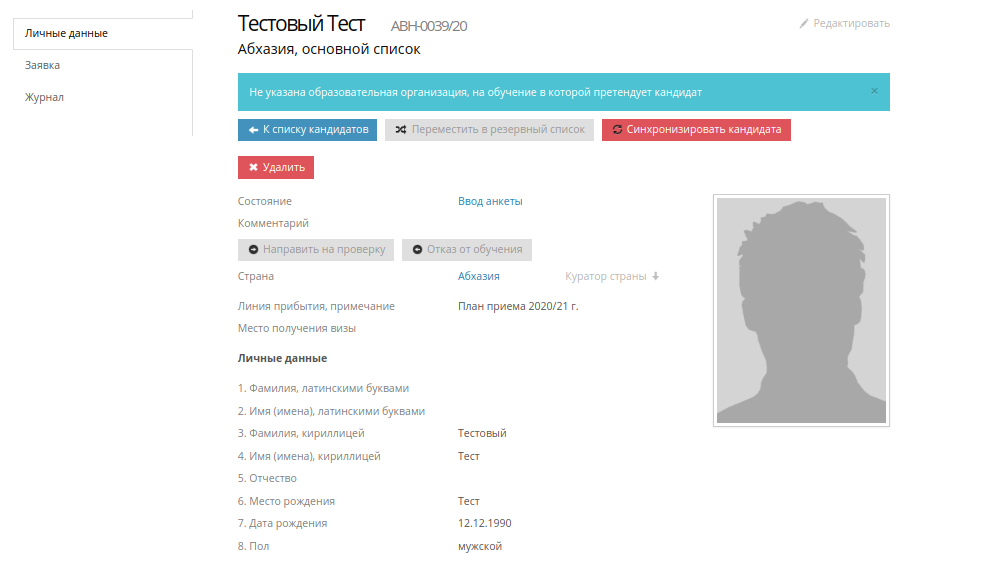


Рисунок 18 – Блок личный данные на странице кандидата

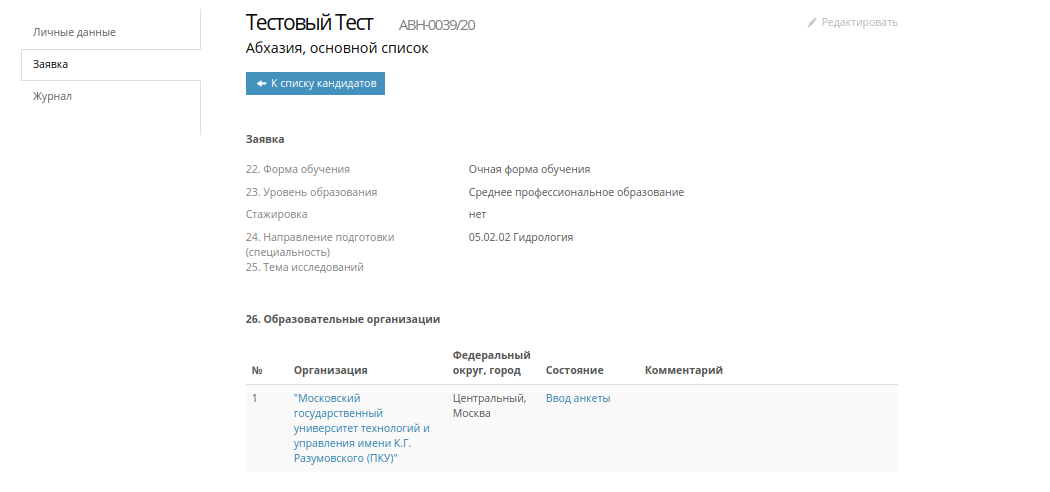


Рисунок 19 – Блок заявка на странице кандидата

На вкладке Страны реализовано добавление кандидата (Рисунок 20).

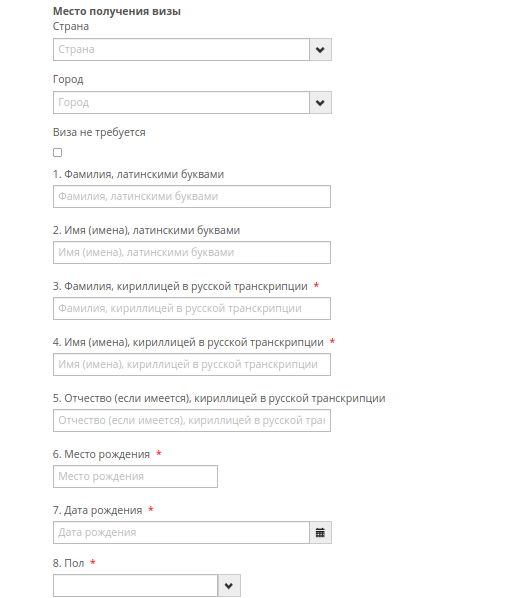


Рисунок 20 – Форма добавления кандидата

**5.2 Разработка программных модулей**

При разработки клиентской части использовались scala-шаблоны фреймворка Play. Разработка серверной части велась на языке Scala с использованием того же фреймворка Play.

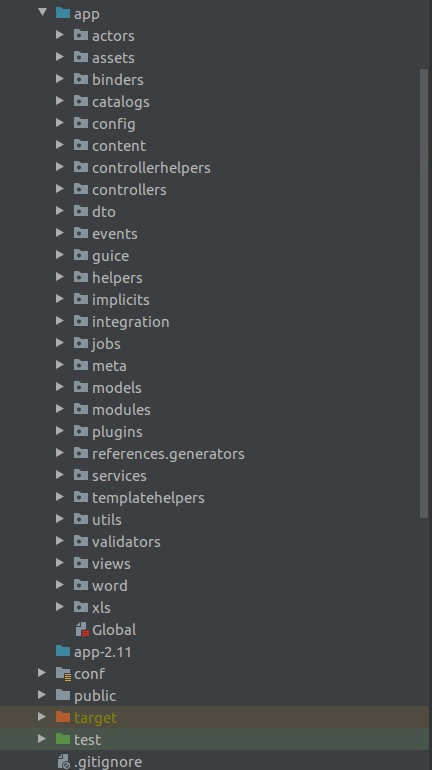


Рисунок 21 – Структура приложения

В папке public хранятся все css и js файлы приложения, а также все картинки.

Папка utils содержит вспомогательные функции, участвующие в дальнейшей разработке, такие как описание api методов, функции для работы со строками и массивами, константы.

В папке conf лежат файлы конфигурации приложения.

В папке templatehelpers находятся шаблоны для отображения основных частей интерфейса.

В папке catalogs находится сервис с условной мини базой данных, которую можно менять не выключая приложение.

В папке validators располагаются классы и файлы для валидации приложения.

В папке models располагаются модели объектов.

В папке controllers располагаются контроллеры.

В папке view располагаются scala-шаблоны для отображения страниц сайта.

Старт приложения начинается с файла Build.scala и Global.scala. Build.scala отвечает за сборку приложения и подтягивает недостающие библиотеки и зависимости. Global.scala запускает само приложение, в этом файле можно описать, что приложение должно сделать перед стартом, для старта, во время старта и т.д.

Файл Global.scala

object Global

extends WithFilters(PjaxNoCacheFilter, HtmlNoCacheFilter)

with GlobalSettings

with DebugHelper

with InjectHelper

with AkkaHelper {

override def getControllerInstance[A](controllerClass: Class[A]) = getInstance(controllerClass)

override def beforeStart(app: Application) {

System.setProperty("user.timezone", EIAppConfig.defaultTimeZone)

TimeZone.setDefault(TimeZone.getTimeZone(EIAppConfig.defaultTimeZone))

}

override def onStart(app: Application) {

initializeSqueryl(app)

startJob()

deleteTempFiles()

sendOnApprovalDataAfterCrash()

}

private def startJob() {

Future {

try {

def exec(needExec: Boolean, op: () => Unit, msg: String) {

if (needExec)

TimeLogger(msg, {

try { op() } catch { case e: Exception => Logger.error(e.getMessage, e)}

})

}

exec(EIAppConfig.importCatalogsOnStart, () => inject[CatalogProcessor].importCatalogs(), "catalogs imported")

} catch {

case e: Exception => Logger.error(e.getMessage, e)

}

}

}

private def initializeSqueryl(app: Application) {

SessionFactory.concreteFactory = Some(() =>

{

val s = getSession(SquerylConfig.dbDefaultAdapter, app)

if(SquerylConfig.logSql)

s.setLogger( s => Logger.warn(s))

s

})

}

private def sendOnApprovalDataAfterCrash() = {

import models.service.MailQueue

Future {

try {

def exec(op: () => Unit) = op()

exec(() => MailQueue.sendFromDb())

} catch {

case e: Exception => Logger.error(e.getMessage, e)

}

}

}

def getSession(adapter:DatabaseAdapter, app: Application) =

Session.create(MyBoneCPPlugin.getConnection(name = "default", autocommit = false)(app), adapter)

override def onRouteRequest(request: RequestHeader): Option[Handler] = Play.maybeApplication.flatMap(\_.routes.flatMap { router =>

//Fix \_pjax request parameter => must ignored

router.handlerFor(PjaxRequestHeader(request))

})

private def deleteTempFiles() {

akkaSchedule(0.seconds, 10.minutes) {

inject[TempFileRepositoryService].deleteOlderThan(180)

}

}

override def onBadRequest(request: RequestHeader, error: String) = {

GlobalErrorHandler.handleBadRequest(error)(getRequestContext(request)) match {

case Some(r) => Future.successful(r)

case None => super.onBadRequest(request, error)

}

}

override def onError(request: RequestHeader, ex: Throwable) = {

Future.successful(GlobalErrorHandler.handleError(ex)(getRequestContext(request)))

}

override def onHandlerNotFound(request: RequestHeader) = {

Future.successful(GlobalErrorHandler.handleNotFound(getRequestContext(request)))

}

private def getRequestContext(request: RequestHeader): RequestContext =

RequestContext(request = Request.apply(request, AnyContentAsEmpty),

principal = SessionHelper.principalFromSession(request.session),

lang = CookieLanguageHelper.languageFromCookie(request.cookies)

)

}

Основные зависимости и библиотеки приложения - Файл Build.scala

object ApplicationBuild extends Build{

val AppName = "fsm"  
val AppVersion = "2.3"  
val PlayVersion = "2.3.7"  
val ScalaVersion = "2.11.11"  
val Guice = "net.codingwell" %% "scala-guice" % "4.0.0"  
val Liftjson = "net.liftweb" %% "lift-json-ext" % "2.6.2"  
val SquerylLib = "org.squeryl" %% "squeryl" % "0.9.5-7"  
val Mustache = "com.github.spullara.mustache.java" % "compiler" % "0.8.15"  
// для templates  
val Findbugs = "com.google.code.findbugs" % "jsr305" % "1.3.9"  
val Quartz = "org.quartz-scheduler" % "quartz" % "2.2.1"  
// для scheduler  
val Liquibase = "org.liquibase" % "liquibase-core" % "3.6.2"  
val CommonsIO = "commons-io" % "commons-io" % "2.4"  
val SolrJ = "org.apache.solr" % "solr-solrj" % "4.2.0"  
val PlayApi = "com.typesafe.play" %% "play" % PlayVersion  
val CommonsValidator = "commons-validator" % "commons-validator" % "1.5.0"  
val Jodatime = "joda-time" % "joda-time" % "2.9.2"  
val Poi = "org.apache.poi" % "poi" % "3.15"  
val PoiXml = "org.apache.poi" % "poi-ooxml" % "3.15"  
val PoiStratchpad = "org.apache.poi" % "poi-scratchpad" % "3.15"  
val Xstream = "com.thoughtworks.xstream" % "xstream" % "1.3.1"  
val Zxing = Seq("com.google.zxing" % "core" % "3.2.1", "com.google.zxing" % "javase" % "3.2.1")  
val Ehcache = "net.sf.ehcache" % "ehcache-core" % "2.6.6"  
val ClosureCompiler = "com.google.javascript" % "closure-compiler" % "v20160208" exclude("com.google.guava", "guava")  
val Scalatest = "org.scalatest" % "scalatest\_2.11" % "3.0.1" % "test"  
val Postgresql = "org.postgresql" % "postgresql" % "9.3-1102-jdbc4"  
val CommonsLang3 = "org.apache.commons" % "commons-lang3" % "3.4"  
val Paranamer = "com.thoughtworks.paranamer" % "paranamer" % "2.5.6"  
val ImgScalr = "org.imgscalr" % "imgscalr-lib" % "4.2"  
val RxScala = "io.reactivex" %% "rxscala" % "0.23.0"  
val UaDetector = "net.sf.uadetector" % "distribution" % "2013.07"  
val CommonsEmail = "org.apache.commons" % "commons-email" % "1.3.1"  
val Ant = "org.apache.ant" % "ant" % "1.7.0"  
val PdfBox = "org.apache.pdfbox" % "pdfbox" % "1.8.6"  
val Jsoup = "org.jsoup" % "jsoup" % "1.7.1"  
val BoneCP = "com.jolbox" % "bonecp" % "0.8.0.RELEASE" exclude("com.google.guava", "guava")  
val ScalaCsv = "com.github.tototoshi" %% "scala-csv" % "1.0.0"  
val JavaDbf = "com.linuxense" % "javadbf" % "0.4.0"  
val PlayMailer = "com.typesafe.play.plugins" % "play-plugins-mailer\_2.11" % "2.3.1" exclude("com.cedarsoft", "guice-annotations")  
val Javassist = "org.javassist" % "javassist" % "3.16.1-GA"  
val shapeless = "com.chuusai" %% "shapeless" % "2.3.2"  
val cats = "org.typelevel" %% "cats-core" % "1.4.0"  
val zio = "dev.zio" %% "zio" % "1.0.0-RC14"  
val AsyncHttpClient = "com.ning" % "async-http-client" % "1.8.13"  
val BouncyCastle = Seq(  
"org.bouncycastle" % "bcprov-jdk15on" % "1.66",  
"org.bouncycastle" % "bcpkix-jdk15on" % "1.66",  
"org.bouncycastle" % "bcprov-ext-jdk15on" % "1.66"  
  
// "org.bouncycastle" % "bcprov-jdk15on" % "1.50"  
)  
  
val eiappDeps = {  
Seq(jdbc, Postgresql, PlayMailer, SquerylLib, Liftjson, Liquibase, CommonsValidator, SolrJ, ws, shapeless,  
ClosureCompiler,  
JavaDbf,  
Scalatest,  
BoneCP,  
ScalaCsv,  
Guice) ++ Zxing ++ BouncyCastle  
}  
  
val sharedSettings = Seq(  
version := AppVersion,  
offline := true,  
scalaVersion := ScalaVersion,  
incOptions := incOptions.value.withNameHashing(nameHashing = true),  
updateOptions := updateOptions.value  
.withCachedResolution(cachedResoluton = true)  
.withLatestSnapshots(latestSnapshots = false),  
transitiveClassifiers in updateClassifiers := List("sources")  
)  
...

…

...  
val eiapp = Project(id = "eiapp", base = file("modules/eiapp"))  
.settings(sharedSettings: \_\*)  
.settings(libraryDependencies ++= (eiappDeps :+ Guice))  
.settings(routesImport ++= Seq("binders.EIAppRoutesBinders" + ".\_"))  
.settings(  
libraryDependencies ++= Seq(ClosureCompiler),  
requireJs += "main.js",  
requireJsShim := "main.js",  
requireJs += "modernizr.js",  
resourceGenerators in Compile «= JavascriptCompiler(Some(closureCompilerOptions))(Seq(\_)),  
LessKeys.compress in Assets := true,  
includeFilter in (Assets, LessKeys.less) := new SimpleFileFilter(file => [file.getParentFile.name](https://vk.com/away.php?utf=1&to=http%3A%2F%2Ffile.getParentFile.name) == "stylesheets" && [file.name](https://vk.com/away.php?utf=1&to=http%3A%2F%2Ffile.name) == "main.less")  
)  
  
val main = Project(AppName, base = file("."))  
.settings(sharedSettings: \_\*)  
.dependsOn(commonutil, auth, eiutil, language, eicore, eiuser, squeryl, catalogs, eimailtemplates, eimailtemplatesview, eijournal, eiapp, mail, timezone)  
.settings(libraryDependencies ++= Seq(jdbc, Guice))  
.aggregate(eiapp)  
.enablePlugins(PlayScala)  
}

…

…

…

Основные классы для взаимодействия с серверной части: CandidateController, RegistrationController, Autharization, UserController представлены ниже:

Файл CandidateController.scala

package controllers.candidate  
  
class CandidateController [)(  
val authorizationService: AuthorizationService,  
val candidateService: CandidateService,  
candidateStateService: CandidateStateService,  
userService: UserService,  
val journalService: JournalRecordService,  
catalogItemService: CatalogItemService,  
val tfaService: TrainingFacultyAcceptanceService,  
tfpService: TrainingFacultyProfileService,  
val fileService: FileRepositoryService,  
val tempFileRepositoryService: TempFileRepositoryService,  
organizationService: OrganizationService,  
candFormCardTplService: CandidateFormAndCardTplService,  
val admissionCycleService: AdmissionCycleService,  
val nationalSelectionService: NationalSelectionService,  
val organizationToCycleService: OrganizationToCycleService,  
val independentSelectionTypeService: IndependentSelectionTypeService,  
val independentSelectionService: IndependentSelectionService](https://vk.com/id9466739)extends Controller with Authorization with SquerylSession with UuidHelper with DateHelper with PagingHelper  
with CandidateForms with ChangeStateWithCommentHelper with StateHistoryHelper[Candidate]  
with UploadFileControllerHelper with FileControllerHelper with StringHelper with CandidateUniquenessControllerHelper  
with EducationRecognitionController with CandidateBackActionHelper  
{  
implicit val timeout = Timeout(1 minute)  
import candidateStateService.\_  
  
def changeVisaStateToRequired(candidate: Candidate) = authorizedAdmin  
{ implicit rc =>  
val updatedCand = candidateService.changeVisaStateToRequired(candidate)  
Ok(views.html.candidate.visaDesignation(updatedCand, catalogItemService))  
}  
  
def changeVisaStateToNotRequired(candidate: Candidate) = authorizedForAsync(adminAttempt)  
{ implicit rc =>  
candidateService.changeVisaStateToNotRequired(candidate)  
.map(cand => Ok(views.html.candidate.visaDesignation(cand, catalogItemService)))  
.recover  
{ case error: Throwable =>  
Logger.error(error.getMessage, error)  
InternalServerError(error.getMessage)  
}  
}  
…

…

...  
def updateArrivalCountry() = authorizedAdmin

{ implicit rc =>  
candidateService.updateArrivalCountry()  
Redirect(controllers.candidate.routes.CandidateListController.list())  
}  
  
val addDirectionsScanCopiesStep1Form = FormBuilderImpl[AddDirectionsScanCopiesStep1Data](Nil)  
.seq(\_.fileIds, ConvertFuns[String]())(\_.label("Файлы").required()).build  
  
val addDirectionsScanCopiesStep2Form = FormBuilderImpl[AddDirectionsScanCopiesStep2Data](Nil)  
.seq(\_.fileAndCandidateIdPairs, ConvertFuns[FileAndCandidateIdConverter]())(\_.required()).build  
  
def addDirectionsScanCopiesStep1 = authorizedFor(ActionAttempt(DirectionsScanCopiesMassAdd))  
{ implicit rc =>  
Ok(views.html.candidate.addDirectionsScanCopies.step1\_UploadFiles(addDirectionsScanCopiesStep1Form,  
controllers.candidate.routes.CandidateController.addDirectionsScanCopiesStep2,  
controllers.candidate.routes.CandidateController.uploadDirectionsScanCopiesTemporary))  
}  
  
def addTFDirectionsScanCopiesStep1 = authorizedFor(ActionAttempt(DirectionsScanCopiesMassAdd))  
{ implicit rc =>  
Ok(views.html.candidate.addDirectionsScanCopies.step1\_UploadFiles(addDirectionsScanCopiesStep1Form,  
controllers.candidate.routes.CandidateController.addTFDirectionsScanCopiesStep2,  
controllers.candidate.routes.CandidateController.uploadDirectionsScanCopiesTemporary))  
}  
  
def addTFDirectionsScanCopiesStep2 = authorizedFormProcessing(addDirectionsScanCopiesStep1Form)(ActionAttempt(DirectionsScanCopiesMassAdd))  
{ errors => implicit rc => BadRequest(views.html.candidate.addDirectionsScanCopies.step1\_UploadFiles(errors,  
controllers.candidate.routes.CandidateController.addTFDirectionsScanCopiesStep2,  
controllers.candidate.routes.CandidateController.uploadDirectionsScanCopiesTemporary)) }  
{ data => implicit rc =>  
val result: Seq[(TempEiFile, Option[Candidate])] = processFiles(data)  
  
Ok(views.html.candidate.addDirectionsScanCopies.step2\_tf\_confirmation(  
result.filter(\_.\_2.exists(candidateSatisfy)).map(p => (p.\_1, p.\_2.get)),  
result.filter(\_.\_2.exists(!candidateSatisfy(\_))).flatMap(\_.\_2),  
result.filter(\_.\_2.isEmpty).map(\_.\_1)  
))  
}  
  
def addDirectionsScanCopiesStep2 = authorizedFormProcessing(addDirectionsScanCopiesStep1Form)(ActionAttempt(DirectionsScanCopiesMassAdd))  
{ errors => implicit rc => BadRequest(views.html.candidate.addDirectionsScanCopies.step1\_UploadFiles(errors,  
controllers.candidate.routes.CandidateController.addDirectionsScanCopiesStep2,  
controllers.candidate.routes.CandidateController.uploadDirectionsScanCopiesTemporary)) }  
{ data => implicit rc =>  
val result: Seq[(TempEiFile, Option[Candidate])] = processFiles(data)  
val transition = CandidateTransitions.commonTransitions.Distributed\_Directed  
  
Ok(views.html.candidate.addDirectionsScanCopies.step2\_Confirmation(  
result.filter(\_.\_2.exists(c => transition.allowedFor(c, c.selectedOrganizations))).map(p => (p.\_1,p.\_2.get)),  
result.filter(\_.\_2.exists(c => !transition.allowedFor(c, c.selectedOrganizations))).flatMap(\_.\_2),  
result.filter(\_.\_2.isEmpty).map(\_.\_1)  
))  
}  
…

…

...  
def createTFDirection(candidate: Candidate) = authorizedFor(ActionAttempt(CreateTFDirection, candidate), predicate = rc => canCreateDirection(candidate))  
{ implicit rc =>  
val file = candidateService.forceCreateDirectionPrintform(candidate, candidate.tfDirectionPrintformFile, false)((c, file) => c.copy(tfDirectionPrintFormFileId = Some([file.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2Ffile.id))))  
sendFile(file.jFile, false, file.filename, rc)  
}  
  
def checkHasDirectionFile(candidate: Candidate): Action[AnyContent] = authorizedFor(ActionAttempt(CreateDirection, candidate))  
{ implicit rc =>  
Ok("{\"result\":" + candidate.directionPrintformFile.nonEmpty + "}").as("application/json")  
}  
  
def checkHasTFDirectionFile(candidate: Candidate): Action[AnyContent] = authorizedFor(ActionAttempt(CreateTFDirection, candidate))  
{ implicit rc =>  
Ok("{\"result\":" + candidate.tfDirectionPrintformFile.nonEmpty + "}").as("application/json")  
}  
  
def printDirection(candidate: Candidate) = authorizedFor(ActionAttempt(PrintDirection, candidate))  
{ implicit rc =>  
candidate.directionPrintformFile match  
{  
case None => NotFound;  
case Some(file) => sendFile(file.jFile, false, file.filename, rc)  
}  
}  
…

…

...  
  
def allowTransition(candidate: Candidate, newStateCode: String)(action: RequestContext => SimpleResult) =  
{  
val transition = findManualTransition(candidate, getState(newStateCode))  
authorizedFor(ActionAttempt(transition.fold(PermissionId(null))(t => t.permissionId), candidate))(action)  
}  
  
def allowTransitionAsync(candidate: Candidate, newStateCode: String)(action: RequestContext => Future[Result]) =  
{  
val transition = findManualTransition(candidate, getState(newStateCode))  
authorizedForAsync(ActionAttempt(transition.fold(PermissionId(null))(t => t.permissionId), candidate))(action)  
}  
  
lazy val allowedMimeTypes = Seq(  
"application/vnd.ms-excel",  
"application/excel",  
"application/msword",  
"application/x-rar",  
"application/x-rar-compressed",  
"application/zip",  
"application/pdf",  
"image/jpeg",  
"image/png",  
"image/gif",  
"application/vnd.openxmlformats-officedocument.wordprocessingml.document",  
"application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"  
)  
  
def downloadTFDirectionScanCopy(candidate: Candidate) = authorizedFor(ActionAttempt(ViewTFDirectionScanCopy, candidate))  
{ implicit rc =>  
val file = candidate.tfDirectionScanCopy.get  
sendFile(file.jFile, fileName = file.filename, rc = rc)  
}

…

…

...  
def changeStateSubmit(candidate: Candidate, newStateCode: String) = allowTransitionAsync(candidate, newStateCode)  
{ implicit rc =>  
findManualTransition(candidate, getState(newStateCode)).get.resolveAsync(candidate)  
}  
  
def showRequest(candidate: Candidate, requestTotalNumberAlert: Boolean = false) = authorizedFor(ActionAttempt(ViewRequestTab, candidate))  
{ implicit rc =>  
val requestBlocks = candidate.tpl.getTemplate().requestBlocks(candidate.independentSelection)  
Ok(views.html.candidate.request(candidate, backAction(), requestBlocks, requestTotalNumberAlert))  
}  
  
def checkCandidate() = checkCandidateUniqueness(ActionAttempt(CreateCandidate))  
  
def deleteSubmit(candidate: Candidate) = authorizedDelete(candidate)  
{ implicit rc =>  
val actor = CandidateCountryAdmissionDispatcherActor.ref  
actor ! DeleteCandidateFromIndexMessage([candidate.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fcandidate.id), candidate.arrivalCountryId, [candidate.admissionCycle.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fcandidate.admissionCycle.id))  
candidateService.deleteCandidateCascade(candidate)  
candidate.nationalSelectionOpt.map(ns =>  
actor ! UpdateNationalSelectionListDataMessage(ns.countryId, ns.admissionCycleId))  
candidate.independentSelection.map(is =>  
actor ! UpdateNationalSelectionListDataMessage(candidate.arrivalCountryId, [is.admissionCycle.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2Fis.admissionCycle.id)))  
Ok  
}  
  
def editVisaDesignation(candidate: Candidate) = authorizedFor(ActionAttempt(EditVisaDesignation, candidate))  
{ implicit rc =>  
Ok(views.html.candidate.editVisaDesignation(visaDesignationForm.fill(VisaDesignationFormData(candidate)), candidate))  
}  
  
def editVisaDesignationSubmit(candidate: Candidate) = authorizedFor(ActionAttempt(EditVisaDesignation, candidate))  
{ implicit rc =>  
visaDesignationForm.bindFromRequest().resolve[VisaDesignationFormData](  
errors => BadRequest(views.html.candidate.editVisaDesignation(errors, candidate)),  
{ data =>  
val newCandidate = candidateService.updateVisaDesignation(candidate, data)  
Redirect(controllers.candidate.routes.CandidateAddEditController.showPersonalData(newCandidate))  
})  
}  
…

…

...  
private def transferCandidate(candidate: Candidate, nationalSelection: NationalSelection, orgToCycle: OrganizationToCycle)  
(implicit rc: RequestContext, timeout: Timeout): Future[Candidate] = {  
val nextAdmCycle = nationalSelection.admissionCycle  
val independentSelectionType = independentSelectionTypeService.getByCode(IndependentSelectionTypeCodes.Transfer)  
val independentSelectionOpt = independentSelectionService.findByOrganizationToCycleAndType(orgToCycle.typedId,  
independentSelectionType.typedId)  
val selection = independentSelectionOpt.fold {  
val selectionToInsert = IndependentSelection(UuidHelper.randomUuid, [orgToCycle.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2ForgToCycle.id),  
[independentSelectionType.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2FindependentSelectionType.id), 1)  
val selectionId = [independentSelectionService.save](https://vk.com/away.php?utf=1&to=http%3A%2F%2FindependentSelectionService.save)(selectionToInsert)  
independentSelectionService.getBy(selectionId)  
} { selection =>  
independentSelectionService.updateEntity(selection.copy(limit = selection.limit + 1))  
selection  
}  
(TransferCandidateDispatcherActor.ref ? TransferCandidate(candidate, candidate.arrivalCountryId,  
[nextAdmCycle.id](https://vk.com/away.php?utf=1&to=http%3A%2F%2FnextAdmCycle.id), selection, rc)).mapTo[Candidate]  
}  
}  
  
case class AddDirectionsScanCopiesStep1Data(fileIds: Seq[String])  
case class AddDirectionsScanCopiesStep2Data(fileAndCandidateIdPairs: Seq[(String, String)])  
  
case class LanguageWithDegreeValidator() extends PreValidator with StringHelper  
{  
def localeKey: String = "validator.required"  
  
def validate(valuesOpt: Option[Seq[String]], form: FormDescription[\_]) = valuesOpt match  
{  
case Some(values: Seq[String]) =>  
{  
values.map(LanguageWithDegreeConverter().convert)  
.find{case (languageId, degreeId) => nonEmptySafe(languageId) && isEmptySafe(degreeId)}  
.map(\_ => RestorationFailure(localeKey))  
}  
case \_ => None  
}  
}  
…

…

...

Остальные основные классы описаны подобно файлу CandidateController.scala и будут приведены в Приложении А. Таким образом была описана основная структура клиентского приложения и приведены фрагменты кода основных компонентов и классов приложения.

**6 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО САЙТА**

Для начала протестируем работу валидации на основных формах - добавление организации и кандидата. Переходим на форму добавления организации (Рисунок 14). Введём неверный формат в полях ввода для проверки валидации (Рисунок 22).

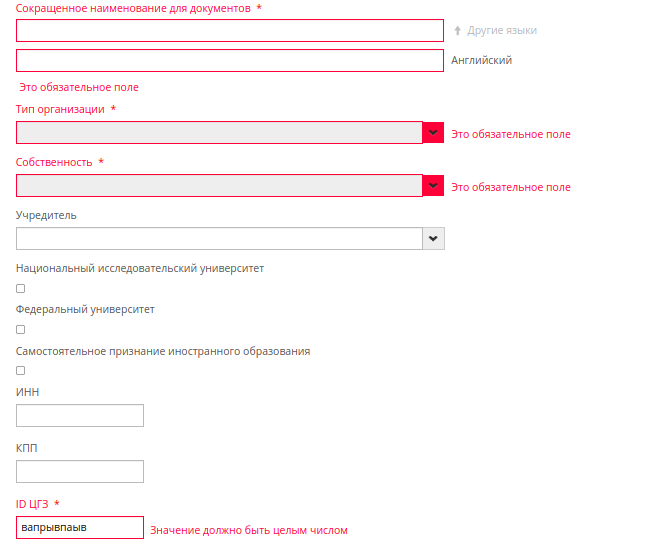


Рисунок 22 – Неверный формат ввода в полях формы добавления организации

Переходим на форму добавления кандидата (Рисунок 20). Введём неверный формат в полях ввода для проверки валидации (Рисунок 23).

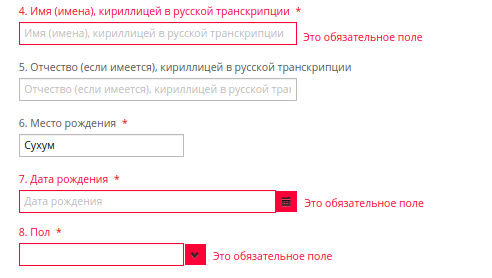


Рисунок 23 – Неверный формат ввода в полях формы добавления кандидата

Проверим правильность работы сервиса подбора университета (навигатора по программам обучения).

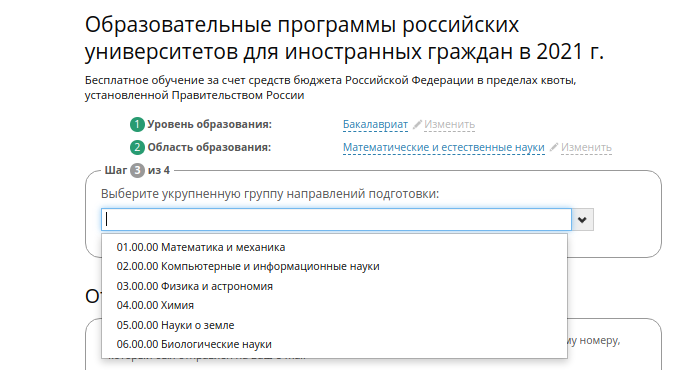


Рисунок 24 – Процесс поиска по навигатору

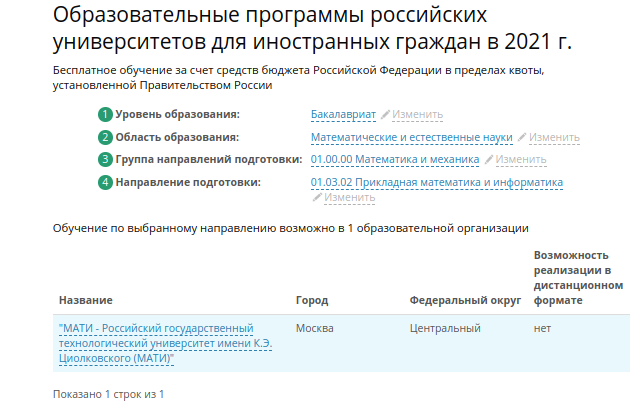


Рисунок 25 – Результат поиска по навигатору

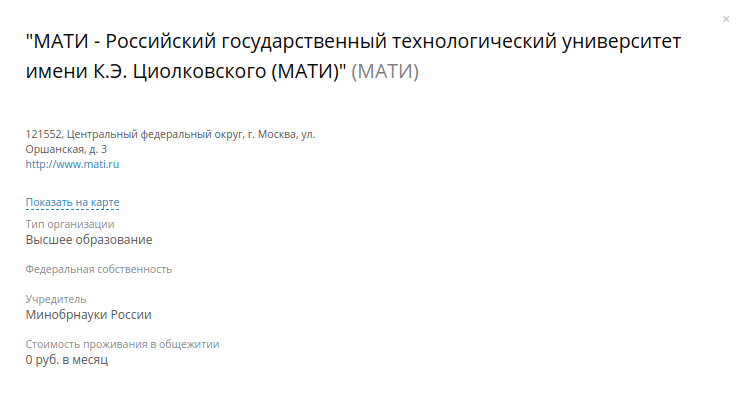
****

Рисунок 26 – Краткая информация про университет

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте была разработана WEB-ориентированная система для поступления в российские университеты.

Для интерфейса менеджера был организован авторизированный доступ. Интерфейс менеджера обеспечивает возможность создания кандидатов, проведение их жизненного цикла, создание организаций и взаимодействие с ними.

Для публичного интерфейса организован свободный доступ. Публичный интерфейс содержит как статическую страницу, с выбором университетов и программ, новостным блоком и блоком основных материалов.

Разработанный WEB-сервис был протестирован на наличие корректной валидации форм, правильной работы авторизации, создания кандидата и поиска по системе.

**СПИСОК ЛИТЕРАТУРЫ И ИНФОРМАЦИОННЫХ РЕСУРСОВ**

1. Play (фреймворк[)](https://ru.wikipedia.org/wiki/Play_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)) // [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Play\_(%D1%84%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA) (дата обращения: 04.05.2021).

2. Scala Documentation // [Электронный ресурс]. URL: https://docs.scala-lang.org (дата обращения: 04.05.2021).

3. Версионирование структуры БД с помощью Liquibase // [Электронный ресурс]. URL: http://easy-code.ru/lesson/database-versioning-liquibase (дата обращения: 25.04.2021).

4. Squeryl Documentation // [Электронный ресурс]. URL: https://www.squeryl.org (дата обращения: 04.05.2021).

5. PostgreSQL — объектно-реляционная система управления базами данных // [Электронный ресурс]. URL: https://web-creator.ru/articles/postgresql (дата обращения: 04.05.2021).

**ПРИЛОЖЕНИЕ А**

КОД ПРИЛОЖЕНИЯ

Файл UserController.scala:

class UserController @Inject()(val referenceService: ReferenceService,

val contactInfoService: ContactInformationService,

val educationService: EducationService,

val careerService: CareerService,

val specializationToUserService: SpecializationToUserService,

val roleToUserService: ActivityKindToUserService,

val academicDegreeService: AcademicDegreeService,

val academicStatusService: AcademicStatusService,

val tokenService: TokenService,

val mailService: MailService,

val authorizationService: AuthorizationService,

val authorizationRoleService: AuthorizationRoleService,

val saltService: SaltService,

val jsonProcessor: JsonProcessor,

val mailTemplateService: MailTemplateService,

val eventBus: EventBus,

val tempFileRepositoryService: TempFileRepositoryService,

val employeePostService: EmployeePostService,

val employeeService: EmployeeService)

extends Controller

with Authorization

with UserForms

with UuidHelper

with ControllerHelper

with UserControllerHelper

with SpecializationController

with RegistrationController

with UserListController

with UserActivityKindController

with AcademicDegreeController

with AcademicStatusController

with UserAccountController

with UserRegistrationByAdminController {

def remove(*user*: User) = authorizedUpdate(*user*) { implicit *rc* =>

*userService*.deleteUserCascade(*user*)

if (*user*.typedId == *rc*.authUser.typedId) gotoLogoutSucceeded

else Redirect(routes.*UserController*.users())

}

def forceUserLogout(*user*: User) = authorizedAdmin { implicit *rc* =>

resolver.removeByUserId(*user*.id)

*rememberMeTokenService*.removeAllTokensForUser(*user*.id)

*Ok*(s"User **$**{*user*.displayableFullTitle} with id: **$**{*user*.id} - logged out")

}

def showUser(*user*: User) = authorizedRead(*user*) { implicit *rc* =>

eventBus.fireEvent(*UserActionStatEvent*(*user*, *rc*))

if (*user*.removed) {

*Ok*(views.html.application.deletedObject(*user*, Constant.menuItems.*contacts*))

} else {

val *jobs* = (employeeService.listOrgEmployeeBy(*user*.typedId) ++

employeeService.listGovEmployeeBy(*user*.typedId)) ++

employeeService.listAgentEmployeeBy(*user*.typedId)

*Ok*(views.html.user.showUser(*user*, *jobs*))

}

}

def editPersonalInfo(*user*: User) = authorizedUpdate(*user*) { implicit *rc* =>

val *form* = personalInfoForm.fill(PersonalInfoDto(lastName = *user*.lastName,

firstName = *user*.firstName,

middleName = *user*.middleName,

lastNameEng = *user*.lastNameEng,

firstNameEng = *user*.firstNameEng))

*Ok*(views.html.user.edit.personalInfo(*form*, *user*))

}

def editPersonalInfoSubmit(*user*: User) = authorizedUpdate(*user*) { implicit *rc* =>

personalInfoForm.bindFromRequest().resolve[PersonalInfoDto](

*formWithErrors* => {

*Ok*(views.html.user.edit.personalInfo(*formWithErrors*, *user*))

},

*personalInfo* => {

val *editedUser* = *user*.copy(lastName = *personalInfo*.lastName,

firstName = *personalInfo*.firstName,

middleName = *personalInfo*.middleName,

firstNameEng = *personalInfo*.firstNameEng,

lastNameEng = *personalInfo*.lastNameEng)

*userService*.updateEntity(*editedUser*)

eventBus.fireEvent(*UserUpdateFIOEvent*(*editedUser*, *user*, *rc*))

Redirect(routes.*UserController*.showUser(*editedUser*))

}

)

}

def editContactInfo(*user*: User) = authorizedUpdate(*user*) { implicit *rc* =>

val *form* = *user*.*contactInfo*.map(*ci* => contactInfoForm.fill(*ContactInfoDto*(email = *ci*.email,

phone = *ci*.phone)))

.getOrElse(contactInfoForm)

*Ok*(views.html.user.edit.contactInfo(*user*, *form*))

}

def editContactInfoSubmit(*user*: User) = authorizedUpdate(*user*) { implicit *rc* =>

contactInfoForm.bindFromRequest().resolve[ContactInfoDto](

*formWithErrors* => {

*Ok*(views.html.user.edit.contactInfo(*user*, *formWithErrors*))

},

*contactInfoData* => {

*user*.*contactInfo* match {

case *Some*(*ci*) => contactInfoService.updateEntity(*ci*.copy(phone = *contactInfoData*.phone,

email = *contactInfoData*.email))

case \_ => contactInfoService.save(*ContactInformation*(id = randomUuid,

userId = *user*.id,

phone = *contactInfoData*.phone,

email = *contactInfoData*.email))

}

Redirect(routes.*UserController*.showUser(*user*))

}

)

}

def addJob(*user*: User) = authorizedFor(security.*ActionAttempt*(security.Permissions.system.*Users*, security.EI)) { implicit rc =>

Ok(views.html.user.editJob(

"Добавление места работы",

"Добавить место работы",

routes.UserController.addJobSubmit(user),

routes.UserController.showUser(user),

editJobForm(None),

user)

)

}

def editJobForm(employee: Option[Employee])(implicit ctx: Context) = FormBuilderImpl[EditJobFormData]()

.string(\_.employerId)(\_.required().reference(EmployerReferenceKey(employee = employee))

.label(l("user.card.workOfPlaceOrganisation.edit"))

.placeholder(l("user.card.workOfPlaceOrganisation.edit"))

.classes("span4 add-scroll"))

.string(\_.post)(\_.required().reference(EmployeePostReferenceKey()).label(l("user.card.position.edit"))

.placeholder(l("user.card.position.edit"))

.classes("span12").validator(LengthValidator(Some(2), Some(255))))

.string(\_.workPhone)(\_.withNullOption().label(l("user.card.workPhone.edit"))

.placeholder(l("user.card.workPhone.edit")).classes("span12")

.validator(LengthValidator(None, Some(255))))

.seq(\_.languageIds, ConvertFuns[String]())(\_.withNullOption().label(l("user.card.languages.edit"))

.placeholder(l("user.card.languages.edit")).reference(LanguageReferenceKey())

.classes("span4 add-scroll"))

.build

def addJobSubmit(user: User) = authorizedFor(security.ActionAttempt(security.Permissions.system.Users, security.EI)) { implicit ctx =>

editJobForm(null).bindFromRequest().resolve[EditJobFormData](

errors => BadRequest(views.html.user.editJob(

"Добавление места работы",

"Добавить место работы",

routes.UserController.addJobSubmit(user),

routes.UserController.showUser(user),

errors,

user)

),

data => {

val employerId = WorkPlaceHelper.extractTypedId(data.employerId)

employeeService.save(user.typedId, employerId, data.post, StringHelper.trimToOption(data.workPhone),

data.languageIds)

if (!authorizationRoleService.systemRoles(user.typedId).exists(\_.typedName == Constant.role.system.Active))

authorizationRoleService.addRoles(user.typedId, Set(Constant.role.system.Active))

Redirect(routes.UserController.showUser(user))

}

)

}

def editJob(job: Employee) = authorizedUpdate(job.user) { implicit rc =>

val filledForm = editJobForm(Some(job)).fill(EditJobFormData.fromModel(job))

Ok(views.html.user.editJob(

l("user.card.placeOfWork.edit"),

l("user.card.save"),

routes.UserController.editJobSubmit(job),

routes.UserController.showUser(job.user),

filledForm,

job.user)

)

}

def editJobSubmit(job: Employee) = authorizedUpdate(job.user) { implicit rc =>

editJobForm(null).bindFromRequest().resolve[EditJobFormData](

errors => BadRequest(views.html.user.editJob(

l("user.card.placeOfWork.edit"),

l("user.card.save"),

routes.UserController.editJobSubmit(job),

routes.UserController.showUser(job.user),

errors,

job.user)

),

data => {

val employerId = WorkPlaceHelper.extractTypedId(data.employerId)

employeeService.updateEntity(job, employerId, data.post, StringHelper.trimToOption(data.workPhone),

data.languageIds)

Redirect(routes.UserController.showUser(job.user))

}

)

}

}

Файл UserServiceImpl.scala:

class UserServiceImpl @Inject()(

contactInformationService: ContactInformationService,

journalRecordService: JournalRecordService,

tokenService: TokenService,

employeeService: EmployeeService,

candidateService: CandidateService,

persistedExceptionService: PersistedExceptionService,

organizationToCycleService: OrganizationToCycleService,

careerService: CareerService,

educationService: EducationService,

academicDegreeService: AcademicDegreeService,

academicStatusService: AcademicStatusService,

specializationService: SpecializationToUserService,

saltService: SaltService,

roleToUserService: ActivityKindToUserService,

authorizationRoleService: AuthorizationRoleService,

fileRepositoryService: FileRepositoryService,

val eventBus: EventBus,

val metaService: MetaClassService,

rememberMeTokenService: RememberMeTokenService,

anonymousSelectedOrganizationService: AnonymousSelectedOrganizationService,

mailTaskService: MailTaskService,

candidateStateChangeLogService: CandidateStateChangeLogService

) extends UserService

with UserQueries

with UpdateEntityServiceImpl

with QueryBuilderEntityServiceImpl

with DefaultEntityServiceImpl

with RemovingEntityServiceImplBase

with DeletionEntityServiceImpl

with UuidHelper

with StringHelper

with PasswordHelper

with JournalHelper

with CountRemovableEntitiesServiceImpl

with PagingQueries

with UserType {

import UserDatabaseSchema.\_

def deleteUserCascade(*user*: User)(implicit *ctx*: Context) = tx {

//EntityWrapper-ы тут не удаляются, при необходимости продумать их удаление и реализовать

val *userId*: UserId = *user*.typedId

val *candidates* = candidateService.list(\_.userId === *user*.id)

contactInformationService.deleteByUserId(*user*.typedId)

anonymousSelectedOrganizationService.deleteByUserId(*userId*)

specializationService.deleteByUserId(*userId*)

saltService.deleteByUserId(*userId*)

*passwordRecoveryLog*.deleteWhere(*prl* => *prl*.userId === *user*.id)

*loginAttemptLog*.deleteWhere(*lal* => *lal*.userId === *Some*(*user*.id))

roleToUserService.deleteByUserId(*userId*)

rememberMeTokenService.removeAllTokensForUser(*user*.id)

tokenService.deleteByUserId(*userId*)

authorizationRoleService.deleteByUserId(*userId*)

academicDegreeService.deleteByUserId(*userId*)

employeeService.listOrgEmployeeBy(*UserId*(*user*.id)).foreach(employeeService.deleteCascade)

*userToUserWorkplace*.deleteWhere(*utuw* => *utuw*.userId === *userId*.raw)

*candidates*.foreach { *candidate* =>

candidateService.deleteCandidate(*candidate*)

}

persistedExceptionService.clearUserField(*userId*)

organizationToCycleService.clearUserField(*userId*)

mailTaskService.deleteByUserId(*userId*)

journalRecordService.deleteByUserId(*userId*)

candidateStateChangeLogService.deleteByUserId(*userId*)

val *avatar* = trimToOption(*user*.avatar)

val *originalAvatar* = trimToOption(*user*.avatarOriginalFile)

delete(*user*)

*avatar*.foreach(*avatar* => fileRepositoryService.delete(*EiFileId*(*avatar*)))

*originalAvatar*.foreach(*aof* => fileRepositoryService.delete(*EiFileId*(*aof*)))

// *TODO это надо в onTxSuccess или что-то вроде этого, вообще, неплохо было бы разобраться с транзакционностью*

GlobalSearchIndexDispatcherActor.*ref* ! *DeleteUserFromIndex*(*user*)

eventBus.fireEvent(UserChanged)

}

override def importUser(

*email*: String,

*sex*: Option[String],

*lastNameEng*: Option[String] = None,

*firstNameEng*: Option[String] = None,

*lastName*: Option[String] = None,

*firstName*: Option[String] = None,

*middleName*: Option[String] = None,

*dateOfBirth*: Option[Date],

*birthPlace*: Option[String],

*password*: Option[String] = None,

*sendMail*: Boolean = false,

*isActive*: Boolean = true,

*createdByAgent*: Boolean = false

)(implicit *ctx*: Context): User = {

importUser(

User(

id = 0L,

email = *email*.toLowerCase,

passwordHash = null,

sex = *sex*.map(SexEnum.withName),

lastNameEng = *lastNameEng*,

firstNameEng = *firstNameEng*,

firstName = *firstName*.getOrElse(""),

lastName = *lastName*.getOrElse(""),

middleName = *middleName*,

dateOfBirth = *dateOfBirth*,

active = true,

activationString = "",

activationDate = null,

createdByAgent = *createdByAgent*

),

*password* = *password*.getOrElse(PasswordHelper.generatePassword()),

*sendMail* = *sendMail*,

*isActive*

)

}

def countByBirthday(*before*: Date, *after*: Date) =

countBy(\_.filter((*u*: User) => *u*.removed === false and

*u*.dateOfBirth.map(*birthDate* => *birthDate* <= *before* and *birthDate* > *after*).getOrElse(1 <> 1) ))

def countNoRoleUsers() = tx {

from(*user*)((*u*) =>

where(*u*.removed === false and notExists(from(*roleToUser*)((*rtu*) => where(*rtu*.user === *u*.id) select (*rtu*.id))))

compute (count(*u*.id))).single.measures

}

def importUser(*user*: User, *password*: String, *sendMail*: Boolean, *isActive*: Boolean)(implicit *ctx*: Context) = {

val *newSalt* = PasswordHelper.generateSalt

val *newUser* = *user*.copy(passwordHash = PasswordHelper.computeHash(*password*, *newSalt*))

onTxSuccess {

val *userId* = save(*newUser*)

saltService.save(*Salt*(0, *userId*.raw, *newSalt*))

if (*isActive*) {

authorizationRoleService.save(*AuthorizationRole*(randomUuid, *userId*.raw, *Some*("System"), "Active"))

}

*newUser*.copy(id = *userId*.raw)

} { *newUser*: User =>

if (*sendMail* && !*user*.email.endsWith(Constant.*fakeEmailDomain*)) {

eventBus.fireEvent(*UserImportedEvent*(*newUser*, *password*, *ctx*))

}

eventBus.fireEvent(UserChanged)

}

}

def listWithoutFriends(*userIdOpt*: Option[UserId], *queryOpt*: Option[String], *pagingOpt*: Option[Paging]) = tx {

type SearchFilter = User => LogicalBoolean

val *filters* = *Seq*[Option[SearchFilter]](

*Some*((*u*: User) => *u*.removed === false),

*queryOpt*.map(*query* => (*u*: User) =>

(lower(*u*.lastName) like (*query* + "%").toLowerCase) or

(lower(*u*.firstName) like (*query* + "%").toLowerCase)

)

)

val *resultFilter* = *filters*.collect { case *Some*(*f*) => *f* }

.reduceLeft((*l*, *r*) => (*u*: User) => *l*(*u*) and *r*(*u*))

val *q* = from(*user*)(*u* =>

where(*resultFilter*(*u*))

select (*u*)

orderBy(*u*.lastName.asc, *u*.firstName.asc)

)

(*pagingOpt*.map(*p* => *q*.page(*p*.begin, *p*.end - *p*.begin)).getOrElse(*q*).toList, *q*.count(u => true))

}

override def save(*user*: User)(implicit *ctx*: Context) = tx {

val *userSafe* = *user*.copy(

firstName = trimSafe(*user*.firstName),

middleName = *user*.middleName.map(\_.trim).filter(\_.nonEmpty),

lastName = trimSafe(*user*.lastName)

)

val *userId* = super.save(*userSafe*)

eventBus.fireEvent(*UserCreatedEvent*(*userSafe*, *ctx*))

eventBus.fireEvent(UserChanged)

GlobalSearchIndexDispatcherActor.*ref* ! *ReindexUser*(*userSafe*)

*userId*

}

def update(*userId*: UserId, *firstName*: String, *middleName*: String, *lastName*: String, *roles*: Seq[String])(implicit *ctx*: Context) = tx {

val *userBefore* = getBy(*userId*)

val *userAfter* = *userBefore*.copy(

firstName = trimSafe(*firstName*),

middleName = *Option*(*middleName*).map(\_.trim).filter(\_.nonEmpty),

lastName = trimSafe(*lastName*),

modified = now

)

updateUser(*userBefore*, *userAfter*)

roleToUserService.updateActivityKinds(*userId*, *roles*.map(CatalogItemId).toSet)

}

def updateAvatar(*user*: User, *avatar*: String)(implicit *ctx*: Context) = tx {

updateUser(*user*, *user*.copy(avatar = *avatar*, modified = now))

}

def updateAvatarAndOriginalFile(*user*: User, *avatar*: String, *avatarOriginalFile*: String)(implicit *ctx*: Context) = tx {

updateUser(*user*, *user*.copy(avatar = *avatar*, avatarOriginalFile = *avatarOriginalFile*, modified = now))

}

private def updateUser(*userBefore*: User, *userAfter*: User)(implicit *ctx*: Context) {

updateEntity(*userAfter*)

val *diff* = attributesDiff(*userBefore*, *userAfter*)

if (*diff*.nonEmpty) {

val *eventDescription* = "Изменены атрибуты пользователя " + *userAfter*.displayableFullTitle +

":\n" + *diff*.mkString("\n")

eventBus.fireEvent(*EntityUpdatedEvent*(*userBefore*, *userAfter*, *ctx*, *Some*(*eventDescription*)))

}

}

override def removeEntity(*entity*: Entity)(implicit *ctx*: Context) = tx {

removeChildren(*entity*)

GlobalSearchIndexDispatcherActor.*ref* ! *DeleteUserFromIndex*(*entity*)

super.removeEntity(*entity*.copy(passwordHash = randomUuid))

eventBus.fireEvent(UserChanged)

}

private def removeChildren(*user*: User)(implicit *ctx*: Context) {

*user*.*academicDegrees*.foreach(academicDegreeService.removeEntity)

*user*.*academicStatuses*.foreach(academicStatusService.removeEntity)

*user*.*career*.foreach(careerService.removeEntity)

*user*.*education*.foreach(educationService.removeEntity)

eventBus.fireEvent(UserChanged)

}

def filteredPage(*filter*: Option[String], *paging*: Paging) = tx {

pageQuery(*filter*.getOrElse(""), *paging*).toList

}

def findByTitleOrEmail(*filter*: String, *paging*: Option[Paging]) = tx {

filteredPageQuery(

(*u*: User) =>

((lower(*u*.lastName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.firstName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.middleName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.lastName + " " + *u*.firstName + " " + *u*.middleName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.email) like ("%" + *filter* + "%").toLowerCase))

and *u*.removed === false,

\_.lastName asc, *paging*).toList

}

def countByTitleOrEmail(*filter*: String) = tx {

countNotRemovedQuery(

*Seq*((*u*: User) =>

(lower(*u*.lastName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.firstName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.middleName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.lastName + " " + *u*.firstName + " " + *u*.middleName) like ("%" + *filter* + "%").toLowerCase) or

(lower(*u*.email) like ("%" + *filter* + "%").toLowerCase))

).toLong

}

def allUsers() = list((*u*: User) => *u*.removed === false)

def countBy(*queryBuilder*: ListQueryBuilder[Long, UserId, User] => ListQueryBuilder[Long, UserId, User]) =

countEntities(*queryBuilder*)

def listBy(*id*: Option[Int], *firstName*: Option[String], *middleName*: Option[String], *lastName*: Option[String],

*email*: Option[String], *roleName*: Option[String], *paging*: Paging): Seq[User] = tx {

val *q* = join(*user*, *authorizationRole*.leftOuter)((*u*, *role*) =>

where(*id*.map(*i* => *u*.id === *i*).getOrElse(1 === 1) and

*firstName*.map(*fn* => lower(*u*.firstName) like s"%**$**{*fn*.toLowerCase}%").getOrElse(1 === 1) and

*middleName*.map(*mn* => lower(*u*.middleName) like s"%**$**{*mn*.toLowerCase}%").getOrElse(1 === 1) and

*lastName*.map(*ln* => lower(*u*.lastName) like s"%**$**{*ln*.toLowerCase}%").getOrElse(1 === 1) and

*email*.map(*e* => lower(*u*.email) like s"%**$**{*e*.toLowerCase}%").getOrElse(1 === 1) and

*roleName*.map(rId => *role*.map(\_.name) === *roleName*).getOrElse(1 === 1)

)

select *u*

on (*role*.map(\_.user) === *Some*(*u*.id))

).distinct

PagingQueriesHelper.addPaging(*q*, *paging*).toList

}

def countBy(*id*: Option[Int], *firstName*: Option[String], *middleName*: Option[String], *lastName*: Option[String],

*email*: Option[String], *roleName*: Option[String]): Long = tx {

join(*user*, *authorizationRole*.leftOuter)((*u*, *role*) =>

where(*id*.map(*i* => *u*.id === *i*).getOrElse(1 === 1) and

*firstName*.map(*fn* => lower(*u*.firstName) like s"%**$**{*fn*.toLowerCase}%").getOrElse(1 === 1) and

*middleName*.map(*mn* => lower(*u*.middleName) like s"%**$**{*mn*.toLowerCase}%").getOrElse(1 === 1) and

*lastName*.map(*ln* => lower(*u*.lastName) like s"%**$**{*ln*.toLowerCase}%").getOrElse(1 === 1) and

*email*.map(*e* => lower(*u*.email) like s"%**$**{*e*.toLowerCase}%").getOrElse(1 === 1) and

*roleName*.map(rId => *role*.map(\_.name) === *roleName*).getOrElse(1 === 1)

)

compute count(*u*.id)

on (*role*.map(\_.user) === *Some*(*u*.id))

).single.measures

}

def searchBy(*qb*: QueryBuilder => QueryBuilder, *page*: Int, *pageSize*: Int = UserConstant.paging.*DefaultPageSize*) = {

val *start* = *pageSize* \* *page*

val *end* = *start* + *pageSize*

searchBy(*qb*, *Paging*(*start*, *end*))

}

def searchBy(*queryBuilder*: QueryBuilder => QueryBuilder, *paging*: Paging) = {

listWithBuilder((*lqb*: ListQueryBuilder[Long, UserId, User]) => {

*queryBuilder*(*lqb*.paging(*paging*))

})

}

def listBuilderImpl: QueryBuilder = new QueryBuilderImpl() {}

def emailExists(*email*: String): Boolean = list((*u*: User) => lower(*u*.email) === *email*.trim.toLowerCase, \_.removed === false).nonEmpty

def changePassword(*userId*: Long, *password*: String)(implicit *ctx*: RequestContext) = tx {

val *saltEntity* = saltService.byUser(*UserId*(*userId*)).copy(salt = generateSalt)

saltService.updateEntity(*saltEntity*)

val *userEntity* = getBy(*UserId*(*userId*))

val *userAfter*: User = *userEntity*.copy(passwordHash = computeHash(*password*, *saltEntity*.salt), modified = now)

updateUser(*userEntity*, *userAfter*)

val *remember* = *ctx*.*cookies*.get(RememberMe.*COOKIE\_NAME*)

val *rememberMe* = RememberMe.decodeFromCookie(*remember*)

rememberMeTokenService.removeTokensForUser(*RememberMeToken*(*rememberMe*))

}

def findByEmail(*email*: String): Option[User] = list((*u*: User) => lower(*u*.email) === *email*.trim.toLowerCase

and *u*.removed === false).headOption

def byLegacyId(*legacyId*: String): Option[User] = tx {

list(\_.legacyId === *legacyId*).headOption

}

def verifyUserPassword(*user*: User, *password*: String) =

*user*.passwordHash.equals(computeHash(*password*, saltService.byUser(*UserId*(*user*.id)).salt))

override def defaultOrdering = *u* => *u*.lastName.desc

def findActiveUser(*userId*: UserId) = tx {

findNotRemovedBy(*userId*).filterNot(\_.isBlocked)

}

override def innerRemove(*entity*: UserServiceImpl#Entity): UserServiceImpl#Entity = {

val *result* = super.innerRemove(*entity*)

eventBus.fireEvent(UserChanged)

*result*

}

def listMultipleRoleUsers(): Seq[User] = tx {

val *groupedByRolesCountQuery* = join(*user*, *authorizationRole*)((*u*, *ar*) =>

where(*u*.removed === false)

groupBy *u*.id

compute countDistinct(*ar*.id)

on (*ar*.user === *u*.id)

)

from(*groupedByRolesCountQuery*, *user*)((*q*, *u*) =>

where(*u*.id === *q*.key and *q*.measures.gt(1))

select *u*

).toList

}

override def updateEntity(*entity*: Entity)(implicit *ctx*: Context) = tx {

val *updatedEntityId* = super.updateQuery(*entity*)

GlobalSearchIndexDispatcherActor.*ref* ! *ReindexUser*(*entity*)

eventBus.fireEvent(UserChanged)

*updatedEntityId*

}

override def findByActivationString(*activationString*: String) = list(\_.activationString === *activationString*).headOption

override def activateUser(*userId*: UserId)(implicit *ctx*: Context): User = tx {

val *user* = getBy(*userId*)

authorizationRoleService.save(*AuthorizationRole*(id = randomUuid, user = *userId*.raw, target = *Some*("System"), name = "Active"))

*user*

}

override def hasOneOfRoles(*user*: User, *roleNames*: Seq[AuthorizationRoleName]): Boolean = tx {

join(*defaultTable*, *authorizationRole*)((*u*, *ar*) =>

where(*u*.id === *user*.id and

(*ar*.name in *roleNames*.map(\_.raw)))

select *ar*

on (*ar*.user === *u*.id)).nonEmpty

}

override def listByPhone(*phone*: String): List[User] = tx {

join(*defaultTable*, *contactInformation*)((*u*, *ci*) =>

where(*ci*.phone.getOrElse("") === *phone* or *ci*.phone.getOrElse("") === *phone*.replace(" ",""))

select *u*

on(*u*.id === *ci*.userId)

).toList

}

}

Файл Autharization.scala:

trait Authorization extends Auth with InjectHelper with Controller {

type RedirectFun = Request[AnyContent] => Result

private lazy val *appRoutes* = inject[ApplicationRoutes]

private lazy val *userService* = inject[UserService]

protected def authorizationService: AuthorizationService

def authorizedCreate[T <: AnyRef](*target*: MetaIdentifiable = EI, *predicate*: RequestContext => Boolean = (*rc*) => true)(*action*: RequestContext => Result) =

authorizedFor(security.*ActionAttempt*(*Create*, *target*), *predicate* = *predicate*)(*action*)

def authorizedRead[T <: AnyRef](*target*: MetaIdentifiable = EI, *predicate*: RequestContext => Boolean = (*rc*) => true)(*action*: RequestContext => Result) =

authorizedFor(security.*ActionAttempt*(*Read*, *target*), *predicate* = *predicate*)(*action*)

def authorizedReadAsync[T <: AnyRef](*target*: MetaIdentifiable = EI, *predicate*: RequestContext => Boolean = (*rc*) => true)(*action*: RequestContext => Future[Result]) =

authorizedForAsync(security.*ActionAttempt*(*Read*, *target*), *predicate* = *predicate*)(*action*)

def authorizedUpdate[T <: AnyRef](*target*: MetaIdentifiable = EI, *predicate*: RequestContext => Boolean = (*rc*) => true)(*action*: RequestContext => Result) =

authorizedFor(security.*ActionAttempt*(*Update*, *target*), *predicate* = *predicate*)(*action*)

def authorizedDelete[T <: AnyRef](*target*: MetaIdentifiable = EI, *predicate*: RequestContext => Boolean = (*rc*) => true)(*action*: RequestContext => Result) =

authorizedFor(security.*ActionAttempt*(*Delete*, *target*), *predicate* = *predicate*)(*action*: RequestContext => Result)

def authorizedAdmin[T <: AnyRef](*action*: RequestContext => Result) = authorizedFor(*adminAttempt*)(*action*)

def authorizedAdminAsync[T <: AnyRef](*action*: RequestContext => Future[Result]) = authorizedForAsync(*adminAttempt*)(*action*)

val *adminAttempt*: ActionAttempt = security.*ActionAttempt*(*Administration*, EI)

private def notAuthAction = {implicit *rc*: RequestContext => *rc*.principal match {

case Anonymous => *appRoutes*.redirectToLoginWithBackUrl(*rc*)

case *p*:UserPrincipal => *appRoutes*.redirectToAccessDenied

}}

def authorizedFor(*attempt*: ActionAttempt, *bodyParser*: BodyParser[AnyContent] = BodyParsers.parse.*anyContent*,

*predicate*: RequestContext => Boolean = (*rc*) => true,

*notAllowedAction*: RequestContext => Result = *rc* => *BadRequest*)

(*action*: RequestContext => Result) =

authorizedOrElseImpl(*attempt*, *bodyParser*, *predicate*, *notAllowedAction*)(*action*)(notAuthAction)

def authorizedOrElse(*attempt*: ActionAttempt)(*action*: RequestContext => Result)(*nAa*: RequestContext => Result) =

authorizedOrElseImpl(*attempt*, BodyParsers.parse.*anyContent*, (*rc*) => true, *rc* => *BadRequest*)(*action*)(*nAa*)

def authorizedForAsync(*attempt*: ActionAttempt, *bodyParser*: BodyParser[AnyContent] = BodyParsers.parse.*anyContent*,

*predicate*: RequestContext => Boolean = (*rc*) => true,

*notAllowedAction*: RequestContext => Future[Result] = *rc* => Future.*successful*(*BadRequest*))

(*action*: RequestContext => Future[Result]) =

authorizedOrElseAsync(*attempt*, *bodyParser*, *predicate*, *notAllowedAction*)(*action*)(*rc* => Future.*successful*(notAuthAction(*rc*)))

def authorizedForAny(*attempts*: Seq[ActionAttempt], *bodyParser*: BodyParser[AnyContent] = BodyParsers.parse.*anyContent*)

(*action*: RequestContext => Result) = authorizedAnyOrElse(*attempts*, *bodyParser*)(*action*)(notAuthAction)

def authorizedAnyOrElse(*attempts*: Seq[ActionAttempt], *bodyParser*: BodyParser[AnyContent] = BodyParsers.parse.*anyContent*)

(*action*: RequestContext => Result)

(*elseAction*: RequestContext => Result) = optionalUserAction(

{ *rc* =>

if (*attempts*.map(*attempt* => authorizationService.check(*attempt*, *rc*.principal)).filter(*p* => *p*).nonEmpty)

*action*(*rc*)

else

*elseAction*(*rc*) },

*bodyParser*

)

def authorizedOrElseImpl(*attempt*: ActionAttempt, *bodyParser*: BodyParser[AnyContent], *predicate*: RequestContext => Boolean, *notAllowedAction*: RequestContext => Result)

(*action*: RequestContext => Result)

(*notAuthAction*: (RequestContext) => Result) = authorizedOrElseBase[Result](optionalUserAction)(*attempt*, *bodyParser*, *predicate*, *notAllowedAction*)(*action*)(*notAuthAction*)

def authorizedOrElseAsync(*attempt*: ActionAttempt, *bodyParser*: BodyParser[AnyContent], *predicate*: RequestContext => Boolean, *notAllowedAction*: RequestContext => Future[Result])

(*action*: RequestContext => Future[Result])

(*notAuthAction*: (RequestContext) => Future[Result]) = authorizedOrElseBase[Future[Result]](optionalUserActionAsync)(*attempt*, *bodyParser*, *predicate*, *notAllowedAction*)(*action*)(*notAuthAction*)

def authorizedOrElseBase[T](*f*: (RequestContext => T, BodyParser[AnyContent]) => Action[AnyContent])

(*attempt*: ActionAttempt, *bodyParser*: BodyParser[AnyContent],

*predicate*: RequestContext => Boolean, *notAllowedAction*: RequestContext => T)

(*action*: RequestContext => T)(*elseAction*: RequestContext => T): Action[AnyContent] =

{

val *contextToResult* = { *rc* : RequestContext => (*predicate*(*rc*), authorizationService.check(*attempt*, *rc*.principal)) match {

case (false, \_) => *notAllowedAction*(*rc*)

case (true, true) => *action*(*rc*)

case (true, false) => *elseAction*(*rc*)

}

}

*f*(*contextToResult*, *bodyParser*)

}

def authorizedFormProcessingWithRc[F: Manifest](*form*: RequestContext => FormDescription[F])

(*attempt*: ActionAttempt, *predicate*: RequestContext => Boolean = (*rc*) => true)

(*handleErrors*: FormDescription[F] => RequestContext => Result)

(*handleSuccess*: F => RequestContext => Result) =

authorizedFor(*attempt*, *predicate* = *predicate*)

{ implicit *rc* =>

*form*(*rc*).bindFromRequest().resolve[F](*errors* => *handleErrors*(*errors*)(*rc*), *data* => *handleSuccess*(*data*)(*rc*))

}

def authorizedFormProcessing[F: Manifest](*form*: FormDescription[F])

(*attempt*: ActionAttempt, *predicate*: RequestContext => Boolean = (*rc*) => true)

(*handleErrors*: FormDescription[F] => RequestContext => Result)

(*handleSuccess*: F => RequestContext => Result) =

authorizedFor(*attempt*, *predicate* = *predicate*)

{ implicit *rc* =>

*form*.bindFromRequest().resolve[F](*errors* => *handleErrors*(*errors*)(*rc*), *data* => *handleSuccess*(*data*)(*rc*))

}

def authorizedFormProcessingAsync[F: Manifest](*form*: FormDescription[F])

(*attempt*: ActionAttempt, *predicate*: RequestContext => Boolean = (*rc*) => true)

(*handleErrors*: FormDescription[F] => RequestContext => Future[Result])

(*handleSuccess*: F => RequestContext => Future[Result]) =

authorizedForAsync(*attempt*, *predicate* = *predicate*)

{ implicit *rc* =>

*form*.bindFromRequest().resolveAsync[F](*errors* => *handleErrors*(*errors*)(*rc*), *data* => *handleSuccess*(*data*)(*rc*))

}

def justRegisteredOrAuthorizedFor(

*attempt*: ActionAttempt,

*activationStringOpt*: Option[String],

*user*: User,

*bodyParser*: BodyParser[AnyContent] = BodyParsers.parse.*anyContent*,

*predicate*: RequestContext => Boolean = (*rc*) => true,

*notAllowedAction*: RequestContext => Result = *rc* => *BadRequest*

)

(*action*: RequestContext => Result) = {

*activationStringOpt*.fold {

authorizedOrElseBase[Result](optionalUserAction)(*attempt*, *bodyParser*, *predicate*, *notAllowedAction*)(*action*)(notAuthAction)

} { *activationString* =>

val *actionFinal* = { *rc*: RequestContext =>

if(*userService*.findByActivationString(*activationString*).contains(*user*)) {

*action*(*rc*)

} else {

notAuthAction(*rc*)

}

}

Action(*bodyParser*)(processRequest(*actionFinal*, *redirect* => *redirect*))

}

}

}

trait ApplicationRoutes extends Controller with SessionHelper {

def redirectToLoginWithBackUrl(*rc*: RequestContext): Result

def redirectToAccessDenied: Result

def loginSucceeded[A](*userId*: Id, *minimized*: Boolean = false, *mailSessionIdOpt*: Option[String] = None)(implicit *rc*: RequestContext): Result

def loginSucceeded[A](*user*: User, *minimized*: Boolean, *mailSessionIdOpt*: Option[String])(implicit *rc*: RequestContext): Result

def logoutSucceeded[A](*request*: Request[A])(implicit *rc*: RequestContext): Result

def authenticationFailed[A](*request*: Request[A]): Result

def authorizationFailed[A](*request*: Request[A]): Result

def deletedObject(*entity*: TitledEntity, *active*: String = null)(implicit *rc*: RequestContext): Result

def getHomePage(*userOpt*: Option[User], *minimized*: Boolean, *mailSessionIdOpt*: Option[String])(implicit *rc*: RequestContext): Call

def localizedIndex(implicit *rc*: RequestContext): Call

}

trait ApplicationTemplates {

def mainPage(*title*: String,

*menu*: String = "",

*isPublic*: Boolean = false)

(*content*: Html)

(implicit *rc*: RequestContext): Html

def withMenu(*active*: String = "")(*title*: Html)(*content*: Html)(implicit *rc*: RequestContext): Html

def withMainMenu(*title*: String, *active*: String)(*menuTitle*: Html)(*content*: Html)(implicit *rc*: RequestContext): Html

}