

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
1 ПОСТАНОВКА ЗАДАЧИ .....	5
1.1 Тренировочные множества в машинном обучении .....	5
1.2 Базовые определения .....	5
1.3 Требования к функционалу .....	7
2 ИНСТРУМЕНТЫ РАЗРАБОТКИ И РЕАЛИЗАЦИЯ .....	8
2.1 Модули приложения .....	8
2.2 Объектно-ориентированная модель приложения .....	8
2.2.1 Модуль «Сервис» .....	8
2.2.2 Модуль «Контроллер» .....	11
2.3 REST API .....	12
2.3.1 Особенности Rest-подхода .....	12
2.3.2 JSON и XML .....	12
2.3.3 Запросы в Rest .....	13
2.3.3 REST API в C&CTool .....	13
2.4 Клиент-приложение .....	20
2.4.1 JavaFX .....	20
2.4.2 Реализация клиента .....	21
2.5 Аутентификация в C&CTool .....	22
2.6 Язык программирования Java .....	24
2.6.1 Обоснование выбора .....	24
2.6.2 Структура приложений на Java .....	24
2.6.3 Аннотации .....	25
2.6.4 Сервлеты .....	25
2.7 Spring Framework .....	26
2.7.1 Описание .....	26
2.7.2 Файлы конфигураций зависимостей .....	27
2.8 Среда разработки IntelliJ IDEA .....	28
2.9 База данных в C&CTool .....	30
2.9.1 PostgreSQL .....	30
2.9.2 Схема базы данных .....	31
2.9.3 Таблицы .....	31
2.10 Maven .....	33
2.10.1 Maven и Ant .....	33
2.10.2 Структура проекта Maven .....	34

2.10.3 Maven-плагины, используемые в проекте.....	35
2.10.4 Жизненный цикл сборки в контексте Maven .....	35
2.11 Удаленное размещение.....	37
2.11.1 Tomcat.....	37
2.11.3 Heroku.....	37
2.11.3 Advanced Rest Client.....	38
ЗАКЛЮЧЕНИЕ .....	39
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	40
ПРИЛОЖЕНИЕ.....	41
Исходный код приложения C&STool.....	41

## ВВЕДЕНИЕ

В настоящее время активно развивается область машинного обучения, подразумевающего наличие больших тренировочных множеств [1]. Задача формирования таких множеств не имеет тривиального решения. Метод краудсорсинга [2] успешно зарекомендовал себя, как инструмент решения данной задачи. Суть данного метода заключена в делегировании выполнения некоторых функций неопределённому кругу добровольцев; координация деятельности последних зачастую происходит с использованием информационных технологий.

Оказалось, что возможности краудсорсинга позволяют достигать качественных результатов при решении крупных задач [3]. Так, данный подход широко и успешно применяется при решении задач «человеческого интеллекта» - то есть задач, выполнение которых требует обязательного участия группы экспертов. К таким задачам относятся, например, выбор лучшей фотографии из набора, составление описания какого-либо продукта или определение исполнителя композиции.

Несмотря на стремительное развитие направления краудсорсинга, существует ряд проблем, решение которых позволило бы стать краудсорсингу серьезной сферой бизнеса. Данные проблемы были рассмотрены в статье Хэла Ходсона [4] применительно к наиболее распространенной в наши дни краудсорсинг-платформе АМТ<sup>1</sup>. К описанным здесь проблемам, в первую очередь, относится проблема отсутствия правовой основы для данного вида деятельности. Отсутствие обратной связи между работниками и заказчиками также не способствует качеству выполняемой работы. Отдельной проблемой краудсорсинг-подхода является исключение такого важного компонента, как командная работа - современные краудсорсинг-платформы не позволяют собирать определенный набор участников для решения какой-либо задачи. Современные платформы предоставляют доступ к большой аудитории работников, но не имеют специальных инструментов для создания тренировочных множеств.

---

<sup>1</sup> <https://www.mturk.com/mturk/welcome>

Заказчикам необходимо самим разрабатывать такие инструменты, что требует больших временных затрат. Решением данной проблемы является разработка интегрированных в краудсорсинг-платформу инструментов формирования тренировочных множеств.

Согласно статье «The Language Demographics of Amazon Mechanical Turk»[5] системы краудсорсинга наиболее распространены в англоязычных странах, а также в Индии и Мексике. Тем не менее, адаптация под отечественную систему оплаты и русскоязычный интерфейс позволит сфере краудсорсинга стать инструментом решения проблем отечественной науки, включая финансовые.

Также в последнее время все более популярно использование мобильных технологий. Наличие программного интерфейса для мобильных устройств сделало бы разработку краудсорсинг-платформ более эффективной.

С учетом вышеописанных проблем, целью настоящей дипломной работы ставится создание отечественной краудсорсинг-платформы C&STool с возможностями командной работы и интегрированной системой взаимодействия заказчиков и работников.

Для достижения поставленной цели были решены следующие задачи:

- Разработка программного компонента для краудсорсинг-платформы, обеспечивающего взаимодействие между участниками рабочего процесса.
- Разработка модуля, позволяющего координировать работу определенной группы участников процесса с целью обеспечения командной работы внутри платформы.
- Разработка API с целью обеспечения взаимодействия клиентов различного типа (мобильные устройства, настольные системы, веб-сервисы) с краудсорсинг-платформой.
- Разработка клиентского модуля, позволяющего решать задачу разметки множества изображений, использующего API платформы.

# **1 ПОСТАНОВКА ЗАДАЧИ**

## **1.1 Тренировочные множества в машинном обучении**

В настоящей работе рассматривается такое направление машинного обучения, как обучение по прецедентам [6].

В общем случае задача машинного обучения по прецедентам ставится следующим образом. Пусть имеется множество  $X$  – объекты, примеры; также имеется множество  $Y$  – ответы, отклики. Предполагается, что существует некоторая зависимость, предоставляющая возможность по  $x \in X$  предсказать появление  $y \in Y$ . Задачей машинного обучения по прецедентам является построение алгоритма, позволяющего по объектам предсказывать появление меток, учитывая, что сама зависимость между объектами и метками известна только на элементах тренировочного множества – пар  $(x, y)$ , называемых прецедентами. Примером задачи машинного обучения по прецедентам является задача определения пешехода на снимке. В данном случае прецедентом является снимок и координаты области снимка, где находится пешеход [7].

Понятно, что высокой точности система машинного обучения получит только при наличии достаточно большого тренировочного множества. Возникает вопрос, как получить большое количество прецедентов?

Ответ на данный вопрос был дан в статье [3]. Краудсорсинг – метод, когда большому количеству пользователей можно делегировать выполнение задачи «человеческого интеллекта» за определенное вознаграждение. «Заказчик» может выгрузить на краудсорсинг-платформу задачу по разметке документов, и она будет распространена среди работников платформы.

## **1.2 Базовые определения**

Платформа, реализация которой описана в данной дипломной работе, должна предоставлять возможность размещения задачи заказчиком. Помимо этого, приложение должно иметь клиент-серверную архитектуру. Сервер предоставляет API для клиентов разного типа. Также платформа должна включать реализацию клиентов, использующих API сервера и ориентированных на

задачу создания тренировочных множеств для таких направлений машинного обучения, как обработка естественно-языковых текстов и компьютерное зрение. В первом случае приложение-клиент должно предоставлять возможности закрепления меток за определенной последовательностью символов внутри текста, а во втором случае значения меток должны закрепляться за регионом на изображении. Ниже приводятся базовые определения, используемые при описании реализации платформы C&STool.

- **Задача**

Задача является основным понятием в рамках платформы C&STool. Задачу размещает «Заказчик». Задача обязательно должна содержать описание, инструкцию к выполнению, а также набор прикрепленных к ней документов.

После того, как задача размещена, к ней открывается доступ всем «работникам» платформы. Каждый работник может отправить по задаче прецеденты, служащие решением данной задачи.

- **Прецедент**

Прецедент представляет собой определенный фрагмент документа, к которому прикреплена метка.

- **Домен**

Домен определяет множество значений меток, привязанных к задаче.

- **Пользователь «Работник»**

Работник должен выполнять задачи, назначенные на него «Заказчиком», либо те задачи, которые работник сам взял на выполнение.

- **Пользователь «Заказчик»**

Заказчик размещает задачу на сервисе. Он может также иметь привилегии на назначение задач конкретным исполнителям. После того, как задача будет выполнена, он может получить набор прецедентов.

### 1.3 Требования к функционалу

Необходимо разработать приложение C&STool, предоставляющее пользователям следующие функциональные возможности:

- Для пользователя типа «Заказчик»

1. Разместить задачу.
2. Удалить задачу.
3. Посмотреть прецеденты.
4. Оценить результаты работы.
5. Просмотреть рейтинг работников.
6. Принять работу.
7. Отвергнуть работу, обязательно указав причину.
8. Назначить работников на выполнение какой-либо задачи.

- Для пользователя типа «Работник»

1. Просмотреть доступные задачи.
2. Получить задачу.
3. Получить документы по задаче.
4. Отправить прецеденты.
5. В случае, если работник не знает, как выполнить задачу по текущему документу, он должен иметь возможность отложить документ.
6. Пожаловаться на задачу.
7. Оценить заказчика.

Администратор сервиса должен иметь глобальные права на работу с базой данных сервиса.

## **2 ИНСТРУМЕНТЫ РАЗРАБОТКИ И РЕАЛИЗАЦИЯ**

### **2.1 Модули приложения**

Приложение C&STool представляет собой клиент-серверное корпоративное приложение, состоящие из следующих модулей:

- CСТ-Core

Модуль «Сервис». В данном модуле заложена основная логика приложения. Здесь реализован функционал работы с базами данных, модели, а также базовые алгоритмы системы.

- CСТ-API

Модуль «Контроллер + API». Модуль предоставляет функционал модуля «Сервис» клиентским приложениям. Модуль размещается непосредственно на сервере и принимает клиентские запросы.

- CСТ-Client

Демонстрационный клиентский модуль, реализованный с использованием технологии Java FX<sup>2</sup>.

### **2.2 Объектно-ориентированная модель приложения**

#### **2.2.1 Модуль «Сервис»**

Модуль «Сервис» представлен набором классов и интерфейсов, задача которых заключается во взаимодействии с базами данных, а также в предоставлении функционала для работы с ними. С модулем «Сервис» напрямую взаимодействует модуль «Контроллер».

В реализации модуля использованы некоторые паттерны проектирования, описанные в книге «Design Patterns» [8]. Данные паттерны являются лучшими практиками при проектировании систем, они позволяют решать проблемы в рамках определенных контекстов их возникновения.

Ниже представлены описания каждого из основных классов модуля, а также их функциональные характеристики.

- Классы и интерфейсы типа «DAO»

---

<sup>2</sup> <http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784>



DAO (англ. Domain Access Object) – объект предоставления доступа к данным. Независимо от типа источника данных, DAO предоставляет набор функций для получения этих данных. В общем случае, DAO является интерфейсом с набором методов для взаимодействия с данными. Определенный класс может реализовать данный интерфейс с ориентацией на конкретный тип набора (например, база данных или текстовый файл).

На рисунке приведена UML-диаграмма DAO для взаимодействия с документами.

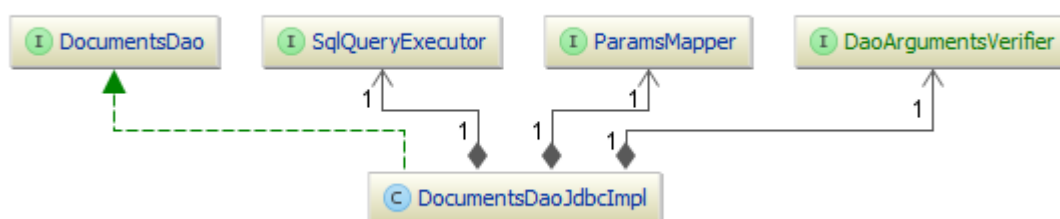


Рисунок 1 – Интерфейс «DocumentsDao» и зависимости его реализации

- Классы и интерфейсы типа «Фасад»

Паттерн проектирования «Фасад» используется для существенного снижения сложности системы с помощью включения определенного набора вызовов в один класс. Таким образом, фасад содержит в себе набор функций разных классов, образуя некую подсистему. При этом любое изменение подсистемы никоим образом не отразится на системе в целом. Общая UML-диаграмма паттерна «Фасад» представлена на Рисунке 2.

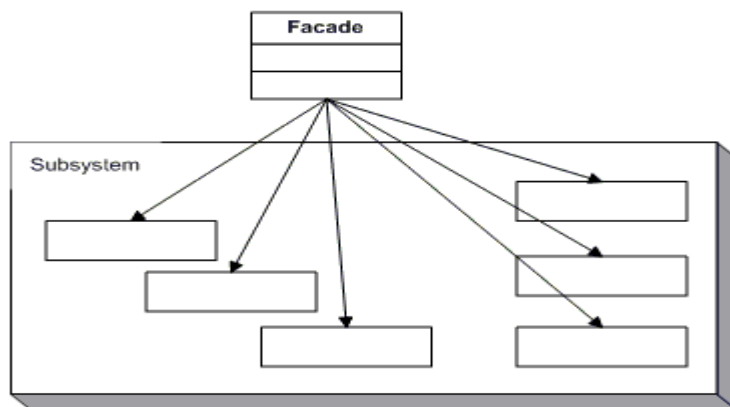


Рисунок 2 – Паттерн проектирования «Фасад»

В модуле «Сервис» проекта C&CTool фасадами являются подсистемы функций «работников», «авторов задач», а также «администраторов». Каждый из этих фасадов агрегирует набор функций из определенных DAO.

Класс «WorkerServiceFacadeImpl» является одним из «фасадов», используемых в модуле. Он выделяет подсистему, отвечающую требованиям пользователя роли «Работник» и предоставляет набор функций для работы в платформе в качестве исполнителя задач.

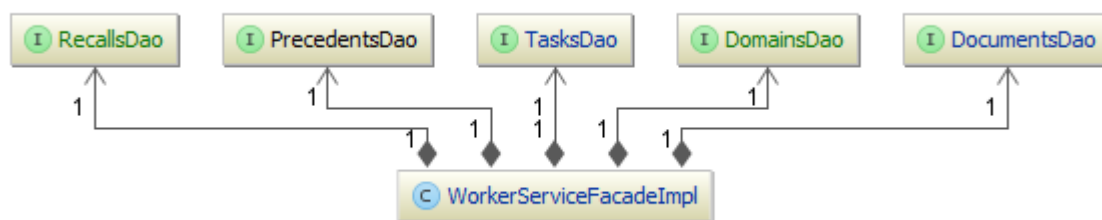


Рисунок 3– Фасад «WorkerServiceFacadeImpl»

Также фасадами являются классы «AdminServiceFacadeImpl» и «TaskOwnerServiceFacadeImpl».

- Классы-утилиты

Классы-утилиты являются вспомогательными классами. Выделение определенных повторяющихся конструкций в отдельные классы позволило соблюсти принцип «минимальной ответственности». Функционал каждого класса должен быть в определенной зоне ответственности, и никоим образом не должен пересекаться с функционалом других классов.

Класс «SqlQueryExecutorImpl» направляет запросы из DAO непосредственно в SQL-базу данных. Задачей класса «ParamsMapperImpl» является создание оберток для объектов системы с целью их передачи в базу данных. Поскольку некоторые запросы в базу данных могут быть составлены неверно, это может вызвать исключения на уровне источника данных, что очень плохо может сказаться на устойчивости системы. Для того, чтобы своевременно выполнять проверку корректности запросов был разработан класс

«DaoArgumentsVerifier». Задачами данного класса является выполнение запросов в источник данных с целью проверки корректности вносимых изменений.

### **2.2.2 Модуль «Контроллер»**

Модуль «Контроллер» отвечает за взаимодействие между «Сервисом» и «Клиентом». В данном модуле выделено несколько семантических блоков. Классы каждого блока обладают конкретной зоной ответственности.

- **Контроллеры**

Классы-контроллеры являются обертками над классами-фасадами модуля «Сервис». Контроллер отображает URL-запрос в конкретный метод класса-фасада. Каждый контроллер также переводит модель, полученную из «Сервиса», в DTO-сущность, которая будет передана клиенту. Обратная конвертация также является зоной ответственности контроллера.

- **Конфигурации**

Классы данного блока отвечают за построение конфигурации всего приложения – контекст приложения, конфигурация безопасности приложения.

- **DTO-классы**

DTO (англ. Data Transfer Object) – объекты для передачи данных. В отличие от моделей модуля «Сервис» данные объекты не обладают никаким поведением, и, по сути, являются структурами данных. Данные объекты конвертируются в JSON-представления и передаются клиенту. Контроллер также обрабатывает в DTO все данные, полученные от клиента.

- **Классы-утилиты**

Как и в «Сервис»-модуле, в данном случае некоторое поведение было инкапсулировано внутри отдельных классов. Так, пара класс-интерфейс «DtoAndEntityConverter» выполняет конвертацию моделей и DTO-объектов, а класс «ResponseBuilder» отвечает за генерацию объектов-ответов.

Класс «ExceptionHandler» перехватывает все возникающие исключения в программе и передает их описания клиенту.

## **2.3 REST API**

### **2.3.1 Особенности Rest-подхода**

Для того, чтобы эффективно выстроить взаимодействие серверной части с клиентами различного типа в C&CTool был использован Rest-подход, описанный в статье [9].

Данный подход описывает принципы взаимодействия клиента и сервера. Основным принципом является ориентация Rest-сервиса на ресурсы. Каждое обращение клиента к серверу происходит посредством запроса данных определенного ресурса, адрес ресурса и параметры фильтрации указаны в URL-запроса. При этом клиенту приходит не сам ресурс, а лишь его представление в формате JSON или XML.

Благодаря Rest-подходу можно полностью отказаться от состояний взаимодействия клиента и сервера: вся информация, передаваемая между сервером и клиентом уже должна содержать информацию о состоянии, а понятие сеанса и сессии отсутствуют.

Описанные принципы, а также многослойность Rest-серверов позволяет не только эффективно настроить политику безопасности, но и повысить масштабируемость приложения.

Достоинства всех вышеописанных принципов определили выбор Rest-подхода в платформе C&CTool.

### **2.3.2 JSON и XML**

В качестве формата для представления ресурсов в C&CTool был выбран формат JSON. Для данного формата, в отличие от XML, характерны удобочитаемость и «немногословность», таким образом он лучше поддается сериализации и обработке на стороне клиента. Одной из задач данной работы является построение эффективного для клиента API, и именно JSON позволяет добиться необходимого результата.

Тем не менее, предусмотрена поддержка формата XML в будущем: для того, чтобы получить представление ресурса в формате XML, пользователю будет достаточно указать соответствующее расширение в конце URL запроса.

### **2.3.3 Запросы в Rest**

HTTP-запрос, описанный в рамках Rest представляет собой запрос, включающий один из пяти методов-операций над необходимым ресурсом:

- GET – метод запроса содержимого ресурса. Используется для получения представления ресурса в необходимом формате. Согласно рекомендациям, описанным в книге, в конце URL-ресурса, указанного в запросе, следует обозначать расширение представления. Например, tasks.json, или tasks.xml.
- POST – метод отправки какого-либо представления ресурса в серверную часть. В теле запроса в определенном формате размещается описание ресурса.
- PUT – метод обновления данных. В теле запроса размещается описание данных, которыми следует обновить какой-либо ресурс.
- DELETE – метод удаления данных.

URL ресурса, куда отправляются запросы, может содержать также параметры, необходимые для фильтрации данных. Помимо этого, в параметрах URL могут быть указаны данные для идентификации пользователя.

В качестве ответа на каждый запрос приходит статус обработки запроса. Получив статус, клиент знает, как именно следует обработать полученные данные, и, если возникла ошибка, определить, какого она типа.

### **2.3.3 REST API в C&CTool**

Платформа C&CTool предоставляет разработчикам клиентских приложений программный интерфейс в формате Rest.

Документация по данному интерфейсу выложена в системе контроля версий в открытом доступе<sup>3</sup>. Каждый ответ для GET-запроса содержит следующие обязательные поля, необходимые для качественной обработки полученных ответов клиентом от сервера:

- `code` – код, полученный клиентом от сервера.
- `status` – метка, указывающая, успешно ли был выполнен запрос.
- `data` – содержимое данного поля представляет собой набор запрашиваемых данных, полученных клиентом от сервера.

В случае, когда запрос имеет тип POST, клиенту возвращаются сами данные, которые он отправил на сервер, а также значения полей обязательных полей.

Любой запрос должен содержать обязательный параметр `auth`, позволяющий идентифицировать пользователя.

В ответ на каждый запрос сервер возвращает определенный статус. Наиболее часто используемые статусы, которые следует использовать в Rest API описаны в статье [10]. В C&STool используется следующий набор статусов:

- 200 – запрос выполнен успешно.
- 201 – сущность, переданная на сервер, была успешно обработана.
- 401 – пользователь не авторизован.
- 403 – у пользователя нет привилегий, либо запрос составлен некорректно.
- 404 – запрашиваемый ресурс не найден.

В случае, когда запрос не может быть выполнен, клиенту, помимо статуса, возвращается представление объекта особого случая, в котором указано подробное описание ошибки.

Пример представления объекта особого случая в формате JSON:

```
{code: "404"}
```

---

<sup>3</sup> [https://bitbucket.org/marsel\\_sidikov/cctool/wiki/](https://bitbucket.org/marsel_sidikov/cctool/wiki/)

```
status: "error"
message: "Task with id <4> not found."
data: "TaskNotFoundException" }
```

Функции API дифференцированы по роли их пользователя. В настоящей работе будет описан набор функций, доступный пользователю роли «Работник».

- Получение списка всех доступных задач, а также задач, назначенных данному работнику.

Полученный набор задач содержит такие характеристики, как идентификатор задачи и ее описание.

Пример запроса:

```
GET tasks.json?auth=1ff45jk7858b8a
```

Пример содержимого поля data ответа в формате JSON:

```
{
  "tasks": [
    {
      "id": "1",
      "description": "Определить расположение
человека и животного на фотографии."
    }
  ]
}
```

- Получение списка отзывов об авторе задачи.

В запросе следует указать значение параметра `user`, который является идентификатором пользователя, отзывы о котором необходимо получить.

Пример запроса:

```
GET /recalls.json?user=1&auth=1ff45jk7858b8a
```

Пример содержимого поля data ответа в формате JSON:

```
{
  "recalls": [
```

```

    {
        "workerId": "3",
        "authorId": "1",
        "mark": "10",
        "text": "Интересные задачи!"
    }
]
}

```

- Получение задачи.

В качестве параметра пути следует указать идентификатор задачи, информацию о которой необходимо получить.

Пример запроса:

```
GET /tasks/1.json?auth=1ff45jk7858b8a
```

Пример содержимого поля data ответа в формате JSON:

```

{
    "id": "1",
    "authorId": "1",
    "description": "Определить расположения человека
и животного на фотографии",
    "attributeDomainId": "1",
    "links": {
        "instructionsFile":
"http://comp.mq.edu.au/units/comp348/ch2.pdf"
    }
}

```

- Назначение задачи для выполнения.

В роли «Работник» пользователь может назначить задачу только на себя. В случае, если указанный в запросе идентификатор пользователя, и параметр `auth` не будут соответствовать, запрос выполнен не будет.

Пример запроса:



POST /tasks/assignments?auth=1ff45jk7858b8a

Пример содержимого поля data ответа в формате JSON:

```
{
  "taskId": "1",
  "workerId": "1"
}
```

- Получение домена, привязанного к задаче.

GET /domains/1.json?auth=1ff45jk7858b8a

Пример содержимого поля data ответа в формате JSON:

```
{
  "id": "1",
  "values": [
    "животное",
    "человек"
  ]
}
```

- Получение набора документов, привязанных к задаче.

Представления набора документов не содержат информации о URL каждого документа.

Пример запроса:

GET /tasks/1/documents.json?auth=1ff45jk7858b8a

Пример содержимого поля data ответа в формате JSON:

```
{
  "documents": [
    {
      "id": "1",
      "taskId": "1",
      "fileName": "photo1",
      "type": "imag",
      "folderName": "task_1_images",

```

```

        "size": "0"
    }
]
}

```

- Получение конкретного документа в рамках задачи.

Пример запроса:

GET /tasks/1/documents/1.json?auth=1ff45jk7858b8a

Пример содержимого поля data ответа в формате JSON:

```

{
    "id": "1",
    "taskId": "1",
    "fileName": "photo1",
    "type": "imag",
    "folderName": "task_1_images",
    "size": "0",
    "links": {
        "self": "http://lady.gazeta.kz/preview/image12845.jpg"
    }
}

```

- Подача жалобы на задачу.

Пример запроса:

POST /tasks/complaints?auth=1ff45jk7858b8a

Пример тела запроса в формате JSON:

```

{
    "taskId": "1",
    "workerId": "1",
    "description": "Нарушение правил сервиса."
}

```

- Отложить просмотр документа.

Пример запроса:

POST /tasks/1/documents/postponed?auth=1ff45jk7858b8

Пример тела запроса в формате JSON:

```
{
  "userId": "1",
  "documentId": "1"
}
```

- Отправка прецедента.

Пример запроса:

POST /tasks/1/precedents?auth=1ff45jk7858b8

Пример тела запроса в формате JSON:

```
{
  "taskId": "human_animal_task_001",
  "userId": "sidikov",
  "object": {
    "documentId": "1",
    "fragment": [
      "90",
      "100",
      "200",
      "300"
    ]
  },
  "label": {
    "value": "животное"
  }
}
```

- Просмотр папки с отложенными пользователем документами.

Пример запроса:

GET /tasks/1/documents/postponed.json?auth=1ff45jk7

Ответ в формате JSON аналогичен ответу на запрос получения всех документов по задаче.

- Завершение выполнения задачи

Пример запроса:

```
POST /tasks/finished?auth=1ff45jk7
```

Пример тела запроса в формате JSON:

```
{
    "taskId" : "1",
    "userId" : "1"
}
```

- Отправка отзыва

Пример запроса:

```
POST /recalls?auth=1ff45jk7
```

Пример тела запроса в формате JSON

```
{
    "recallText" : "Интересные задачи!",
    "userId" : "1"
}
```

## 2.4 Клиент-приложение

### 2.4.1 JavaFX

С целью демонстрации возможностей серверного API было создано клиент-приложение для создания тренировочных множеств. Данное приложение было написано с применением технологии JavaFX, позволяющей разрабатывать Desktop-приложения на языке программирования Java.

JavaFX предлагает широкий набор графических компонентов для разработки интерфейса приложения, а также эффективно интегрируется со средой программирования IntelliJ IDEA.

## 2.4.2 Реализация клиента

Было реализовано два клиента, взаимодействующих с серверным Rest API. Скриншоты их работы представлены ниже.

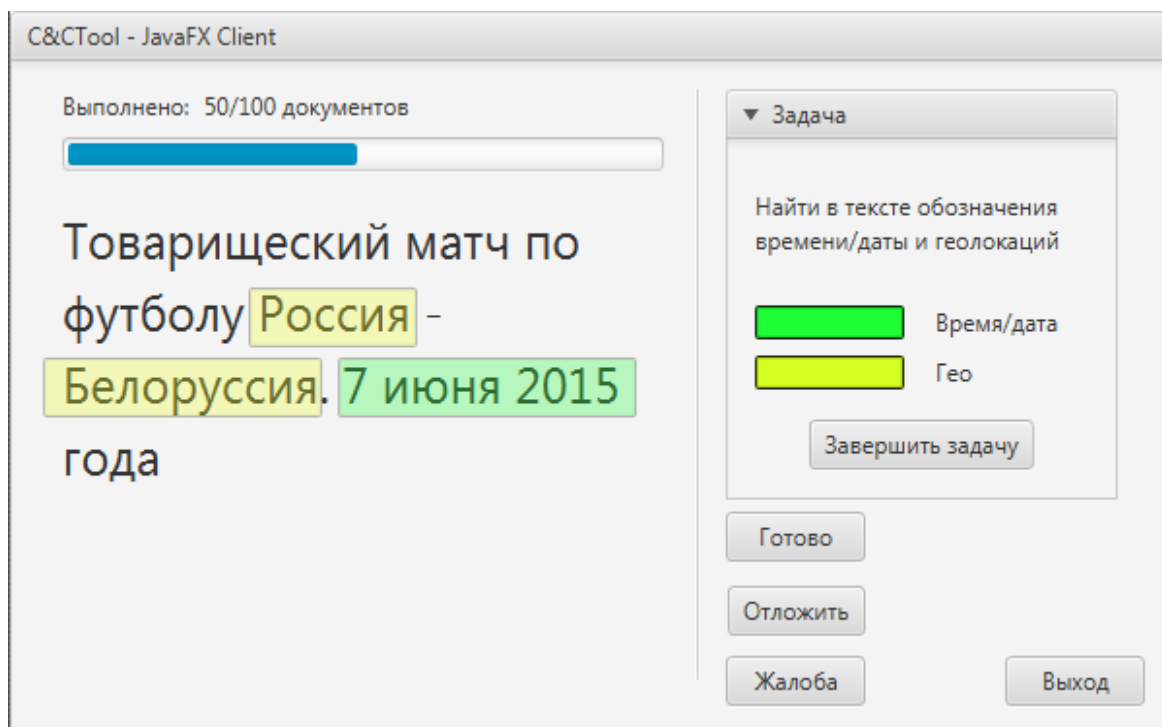


Рисунок 4 – Клиент-приложение с задачей разметки текста

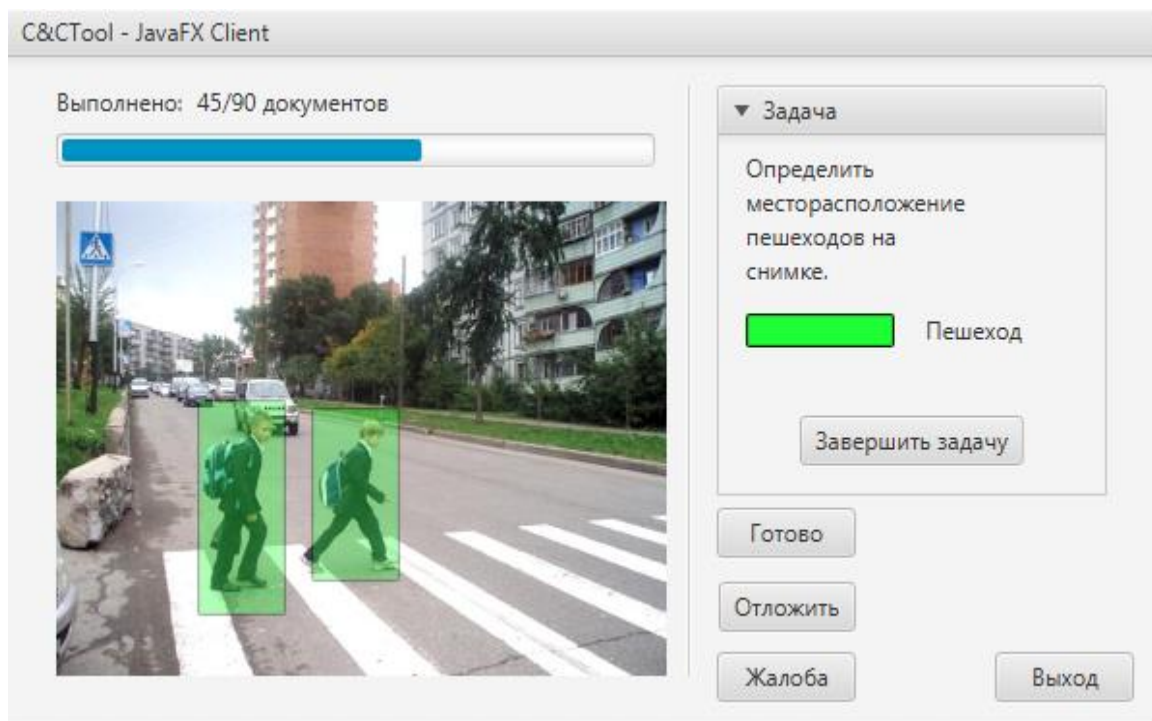


Рисунок 5 – Клиент-приложение с задачей разметки изображения

Приложения позволяют пользователю-работнику выделять определенные области текстового или графического документа и назначать ему определенную метку, формируя прецедент. После того, как пользователь завершит работу, все прецеденты будут отправлены на сервер.

## **2.5 Аутентификация в C&STool**

Учитывая подход Rest, выбранный в качестве механизма взаимодействия клиента и сервера, требуется разработать политику безопасности в приложении.

Понятно, что для обращения к каждому ресурсу необходимо каким-либо способом передавать системе информацию о привилегиях пользователя. Постоянные запросы пароля и логина понижают удобство использования системы и повышают вероятность перехвата пароля.

В C&STool была выбрана политика аутентификации, основанная на токенах [11].

Общий сценарий работы в приложении, использующем аутентификацию, основанную на токенах, приведен на Рисунке.

Для того, чтобы получить доступ к API, клиенту необходимо передать логин и пароль системе C&STool. После проверки совпадения паролей, сервис передает пользователю его токен, который он использует в течении некоторого времени.

Каждый запрос, посылаемый серверу пользователем должен сопровождаться токеном (кроме запроса на регистрацию).

Краткое описание компонентов аутентификации:

- TokenService

Класс, отвечающий за создание токенов, а также назначению их пользователям. Также в обязанности данного класса входит определение пользователя по токену.

- TokenAuthProvider

Компонент, отвечающий за аутентификацию через токен.

- UsernameWithPasswordAuthProvider

Компонент, отвечающий за аутентификацию через имя пользователя и пароль.

UML-диаграмма компонентов, отвечающих за политику безопасности приведена на Рисунке 6.

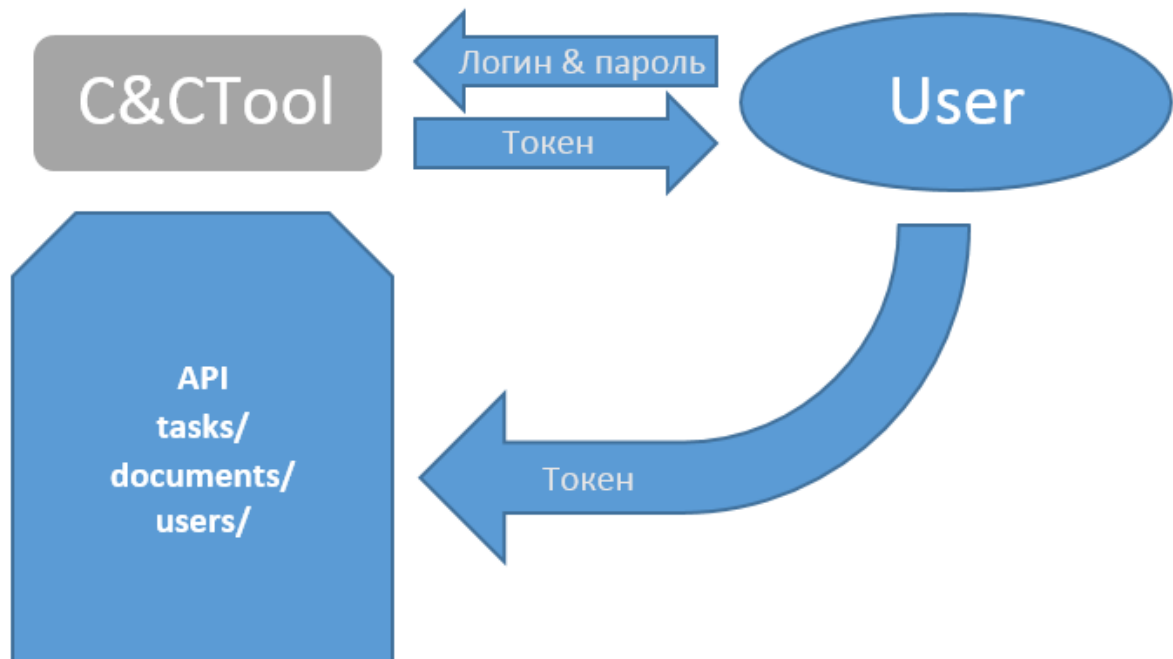


Рисунок 6 – Аутентификация в C&CTool

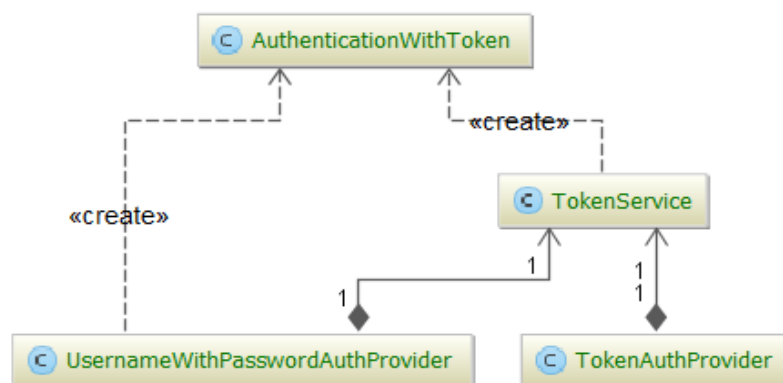


Рисунок 7 – UML-диаграмма компонентов безопасности C&CTool

## **2.6 Язык программирования Java**

### **2.6.1 Обоснование выбора**

Прежде, чем приступить к реализации описанной архитектуры, необходимо выбрать основной инструмент разработки – язык программирования.

Следует выбрать язык программирования, имеющий строгую объектно-ориентированную модель, поддерживающий большое количество фреймворков для работы с базами данных, а также поддерживающих работу клиент-серверных приложений. Таковыми языками являются С# и Java. Язык программирования С++ не имеет строго определенную объектно-ориентированную модель и имеет большую ориентацию на аппаратную составляющую.

Поскольку предполагается, что сервисная часть платформы должна иметь возможность размещения на удаленном сервере, необходимо, чтобы программная часть S&STool была кроссплатформенной, то есть независимой ни от операционной системы сервера, ни от его аппаратной составляющей. Язык С#, реализованный в контексте платформы .NET не может обеспечить надлежащей кроссплатформенности, поскольку его библиотеки и функциональные возможности сильно привязаны к ОС Windows.

Java, будучи одним из первых кроссплатформенных объектно-ориентированных языков программирования, имеет большой набор библиотек, драйверов и фреймворков для работы с наиболее актуальными решениями в области разработок корпоративных приложений.

### **2.6.2 Структура приложений на Java**

Базовым понятием языка программирования Java является класс – абстрактный тип данных, являющийся шаблоном для создания объектов, сущностей, обладающих некоторым заданным состоянием и поведением. Состояние объекта определяют конкретные значения его полей – членов класса. Поведение объекта описывается набором методов – функций и процедур, описанных внутри класса.



Помимо классов и объектов, одним из элементов языка Java являются интерфейсы, которые представляют собой наборы методов, задающих поведение конкретных объектов классов, реализующих данный интерфейс.

Каждое Java-приложение является набором классов. Входной точкой запуска приложения, как и во всех C-подобных языках программирования, является метод-процедура `void Main(String args[])`. Именно с данного метода начинается запуск java-приложения.

В свою очередь, классы описываются в файлах с расширением `.java`. После того, как компилятор `javac` обработает входные файлы, на выходе мы получим файлы с расширением `.class`. Данные файлы являются исполняемыми и представляют собой скомпилированный исходный код в байт-код.

Каждое java-приложение запускается внутри виртуальной машины JVM, что и обеспечивает кроссплатформенность приложений.

В случае, когда следует объединить определенный набор классов в единую библиотеку, `.class` архивируют в `jar`-файл. В данном файле также описывается `MANIFEST.MF` в каталоге `META-INF`. В манифесте определяется, какой именно класс будет запускаться, в случае, когда необходимо сделать исполняемый архив классов.

### **2.6.3 Аннотации**

Аннотации в Java являются мощным механизмом для включения различных метаданных в исходный код. Аннотации проверяются как во время компиляции, так и во время выполнения программы.

Конфигурация фреймворка Spring в новых версиях основана на аннотациях, позволяющих автоматизировать инъекцию зависимостей между классами.

### **2.6.4 Сервлеты**

Отдельным понятием языка программирования Java является понятие сервлета, который представляет собой класс, размещаемый на серверной части

приложений. Сервлет позволяет организовать взаимодействие между клиентом и сервером в рамках принципа запрос-ответ. Для каждого запроса сервлет выделяет отдельный поток. Обеспечить работу большого количества сервлетов можно с помощью контейнера сервлетов, позволяющих помимо размещения сервлетов, определить большое количество конфигурационных параметров активации.

## **2.7 Spring Framework**

### **2.7.1 Описание**

Одним из основных инструментов разработки проекта C&STool является фреймворк Spring<sup>4</sup>. Данный фреймворк включает большое количество различных библиотек классов для написания качественных приложений на Java.

Центральным принципом Spring является принцип инверсии управления. При использовании этого принципа все механизмы создания и установления зависимостей между компонентами становятся внешними.

В классической схеме, если объект класса А зависит от объекта класса В, экземпляру А необходимо с помощью оператора выделения памяти создать экземпляр класса В и привязывать его к своему полю. Используя подход инверсии управления объект В будет создан неким внешним процессом и передан объекту А. Данный метод называется внедрением зависимостей.

Внедрение зависимостей в Spring основано на концепции JavaBean и Java-интерфейсах.

С помощью интерфейсов программисту не требуется указывать, экземпляром какого класса должно быть значение определенной зависимости. Указав в качестве зависимости интерфейс, а не класс, программист предоставляет возможность фреймворку Spring самому выбрать подходящую реализацию зависимости. Каждая реализация интерфейса-зависимости помечается как JavaBean.

---

<sup>4</sup> <http://projects.spring.io/spring-framework/>

### 2.7.2 Файлы конфигураций зависимостей

В последней версии Spring существует возможность описания конфигураций зависимостей внутри специализированных Java-классов. В более ранних версиях требовалось указывать зависимости в XML-файлах, что вызывало определенные трудности при созданий конфигураций.

Пример содержимого конфигурационного файла проекта C&CTool приведен в листинге 1.

*Листинг 1.*

```
package ru.kpfu.tools.cct.rest.config;

import .*;

@EnableWebMvc
@Configuration
@ComponentScan(basePackages = "ru.kpfu")
public class WebApplicationContext extends WebMvcConfig-
urerAdapter {

    @Bean
    public DataSource dataSource() throws URISyn-
taxException {

        DriverManagerDataSource dataSource = new
DriverManagerDataSource();

        String username = "postgres";
        String password = "1234";
        String dbUrl = "jdbc:postgresql://lo-
calhost:5432/cct";

        dataSource.setUrl(dbUrl);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        dataSource.setDriverClassName("org.post-
gresql.Driver");

        return dataSource;    }}
```

Здесь метод `dataSource()`, помеченный аннотацией `@Bean`, создаст объект источника данных и передаст его классам, взаимодействующим с базами данных.

Ниже приводится описание аннотаций, используемых в данном конфигурационном файле.

- `@EnableWebMvc`

Данная аннотация позволяет подключать контроллеры, управляемые аннотациями.

- `@Configuration`

Аннотация определяет данный класс как конфигурационный.

- `@ComponentScan`

Аннотация указывает набор пакетов, в которых следует искать компоненты Spring для инъекции зависимостей.

## 2.8 Среда разработки IntelliJ IDEA

Главным инструментом разработки, безусловно, является среда программирования. Наиболее распространенными системами разработки на Java являются Eclipse<sup>5</sup> и IntelliJ IDEA<sup>6</sup>. В случае разработки корпоративного клиент-серверного приложения C&STool выбор был сделан в пользу IntelliJ IDEA. Список критериев, которые учитывались при выборе инструмента, а также сравнительный анализ двух систем приведен ниже.

- Интуитивно-понятный интерфейс, удобство использования.

IDEA, в отличие от Eclipse, обладает более качественно проработанным интерфейсом. Для получения доступа к различным инструментам внутри IDE достаточно воспользоваться интерфейсом главного окна. Достаточно трех кликов, чтобы передать весь проект в систему контроля версий.

- Горячие клавиши.

---

<sup>5</sup> <https://eclipse.org/>

<sup>6</sup> <https://www.jetbrains.com/idea/>

Большое количество горячих клавиш позволяют моментально переключаться между компонентами интерфейса. Помимо этого, IntelliJ IDEA поддерживает комбинации горячих клавиш не только непосредственно в Java-коде. Комбинации работают в конфигурационных файлах фреймворков, а также в файлах исходного кода других языков. В Eclipse горячие клавиши работают только внутри исходного кода на Java.

- Автозаполнение.

Автозаполнение поддерживается для большого количества языков. Непосредственно в Java-коде IDEA может автозаполнять код на SQL, JSON, JavaScript и HTML. Данной функции в Eclipse не предусмотрено.

- Отладчик

Отладчик является незаменимым инструментом разработки. Отладчик в IDEA позволяет не только содержательно смотреть объекты времени выполнения, но и выводит значения переменных непосредственно в окне редактирования кода.

- Проверка файлов конфигураций

IDEA позволяет проверять правильность написания файлов конфигураций. Эта информация позволяет не отвлекаться на перезапуск сервера или базы данных, в случае, когда что-то было заполнено в конфигурационном файле неверно.

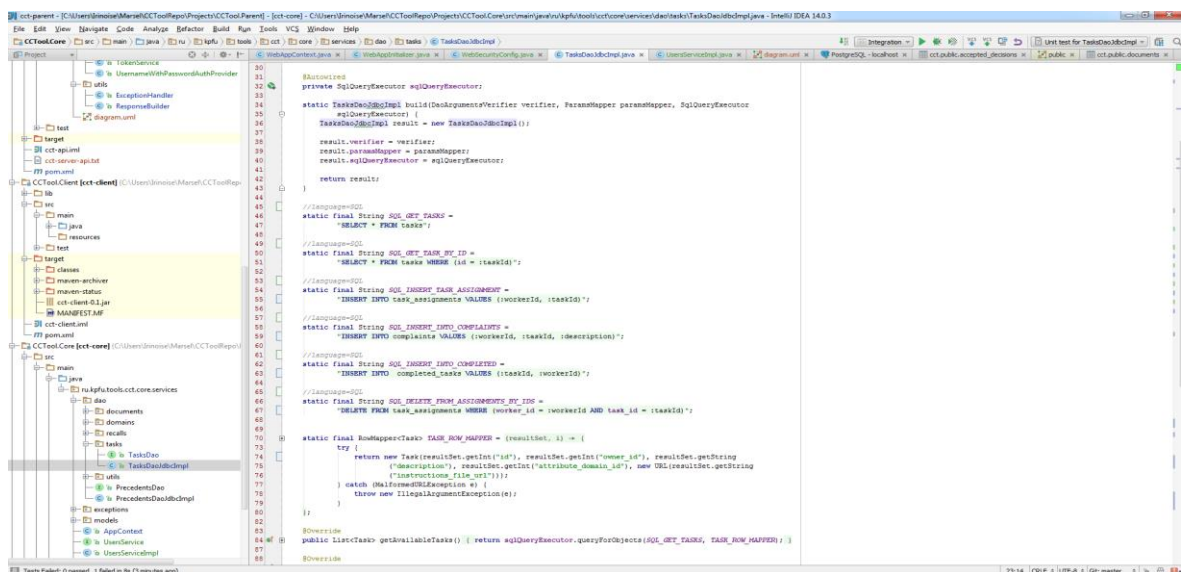


Рисунок 8 – Интерфейс IntelliJ IDEA

## 2.9 База данных в C&CTool

### 2.9.1 PostgreSQL

Наиболее важным элементом при разработке корпоративных приложений является выбор типа хранилища данных. Очевидно, что в случае разработки клиент-серверного приложения с большим количеством хранимой информацией наилучшим выбором является подключение базы данных.

IntelliJ IDEA эффективно интегрируется с SQL Базами данных, в частности, с PostgreSQL<sup>7</sup>. Поскольку мы должны иметь возможность размещения нашей базы данных на удаленном сервере, в качестве СУБД была выбрана именно PostgreSQL, поддерживаемая хостинг-сервером приложений Heroku.

Взаимодействие IntelliJ IDEA и СУБД PostgreSQL происходит с помощью мощного инструмента взаимодействия с базами данных, интегрированного в IDEA.

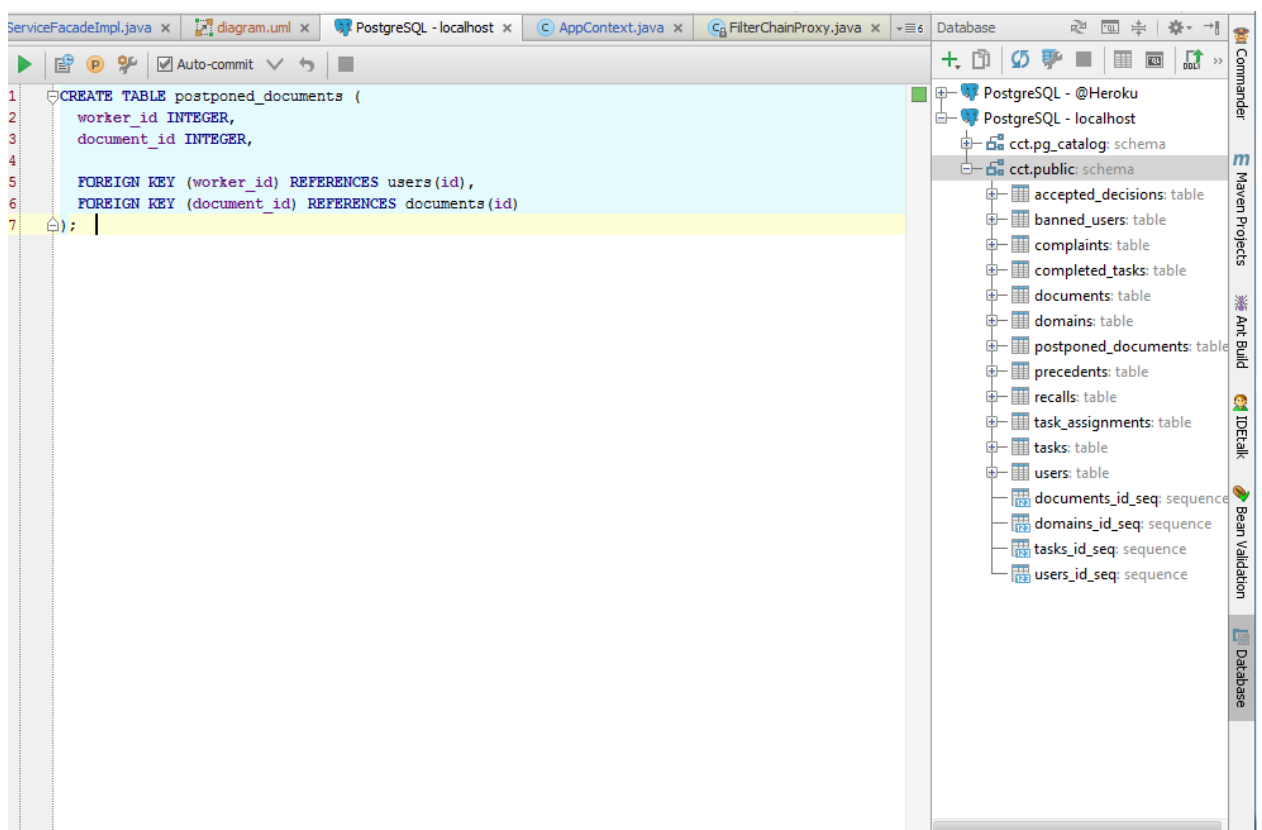


Рисунок 9 – Инструмент «Database» в IDEA

<sup>7</sup> <http://www.postgresql.org/>

## 2.9.2 Схема базы данных

Ниже приведена диаграмма базы данных, используемая в проекте C&STool, составленная с помощью встроенных инструментов IntelliJ IDEA.

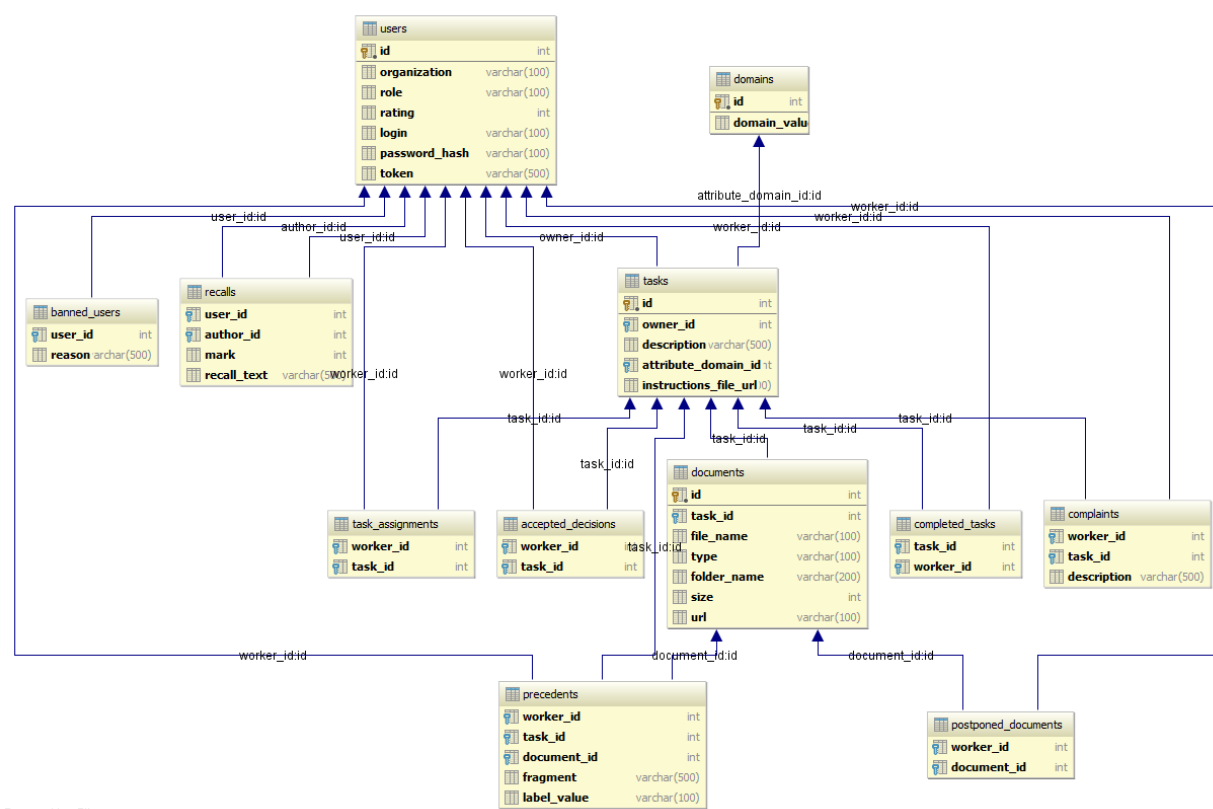


Рисунок 10 – схема базы данных C&STool

## 2.9.3 Таблицы

Здесь приведено описание некоторых таблиц базы данных C&STool.

- Таблица «USERS»

Таблица содержит сведения о пользователях, а также информацию для их аутентификации.

Поля таблицы:

1. ID – идентификатор пользователя.
2. ORGANIZATION – организация пользователя.
3. ROLE – роль пользователя в системе.
4. RAITING – рейтинг пользователя.
5. LOGIN – логин пользователя.

6. PASSWORD\_HASH – хешированный пароль.

7. TOKEN – токен для аутентификации.

- Таблица «TASKS»

Таблица содержит сведения о размещенных в системе задачах.

Поля таблицы:

1. ID – идентификатор задачи.
2. OWNER\_ID – идентификатор заказчика.
3. DESCRIPTION – описание задачи.
4. ATTRIBUTE\_DOMAIN\_ID – идентификатор домена.
5. INSTRUCTIONS\_FILE\_URL – адрес файла с инструкциями по выполнению задачи.

- Таблица «DOCUMENTS»

Таблица содержит информацию о документах, привязанных к определенным задачам.

Поля таблицы:

1. ID – идентификатор документа.
2. TASK\_ID – идентификатор задачи.
3. FILE\_NAME – имя файла.
4. TYPE – типа файла.
5. FOLDER\_NAME – папка документа.
6. SIZE – размер документа.
7. URL – адрес документа.

- Таблица «PRECEDENTS»

Таблица содержит сведения об отправленных пользователями прецедентах.

Поля таблицы:

1. WORKER\_ID – идентификатор работника.
2. TASK\_ID – идентификатор задачи.
3. DOCUMENT\_ID – идентификатор документа.



4. FRAGMENT – сведения о выделенном фрагменте.
5. LABEL\_VALUE – присвоенная фрагменту метка.

## **2.10 Maven**

### **2.10.1 Maven и Ant**

Система Maven<sup>8</sup> является инструментом автоматизированной сборки проектов, разработанных с применением языка программирования Java.

В отличие от распространенной системы Ant<sup>9</sup> инструмент Maven хорошо интегрируется со средой разработки IntelliJ IDEA. Отличительной особенностью Maven является мощный механизм для работы с зависимостями. В классической схеме Ant сторонние библиотеки скачиваются программистом из различных источников, а затем вручную добавляются в папку зависимостей проекта. Используя Maven достаточно написать название нужной библиотеки в специализированном конфигурационном файле проекта, после чего Maven автоматически скачает и подключит как саму библиотеку, так и необходимые для ее работы файлы. Поиск библиотек осуществляется в удаленном репозитории Maven Central Repository<sup>10</sup>.

При установке инструмента Maven, последний создает в корневом каталоге ОС папку «.m2», где хранятся конфигурационные файлы, а также размещен локальный репозиторий, куда сохраняются скачанные библиотеки. Таким образом, для любого вновь созданного проекта Maven в первую очередь попытается найти необходимые зависимости в локальном репозитории, и, только потом, будет загружать их из сети.

В случае, когда несколько проектов используют одинаковые зависимости, имеет смысл объединить описание этих зависимостей в одном месте. Ма-

---

<sup>8</sup> <https://maven.apache.org/>

<sup>9</sup> <http://ant.apache.org/>

<sup>10</sup> <http://search.maven.org/>

ven предоставляет такую возможность. Все зависимости можно описать в специальном Parent-модуле, являющимся предком определенной группы проектов, которые наследуют все зависимости Parent.

Таким образом, учитывая все достоинства системы Maven, она была выбрана в качестве инструмента сборки в проекте C&STool. Каждый модуль C&STool представлял собой Maven проект со своими зависимостями и настроенными фазами сборки.

### 2.10.2 Структура проекта Maven

Ниже приведена структура проекта в контексте инструмента Maven:

- src
  - main
    - java
    - resources
    - ...
  - test
    - java
    - test-resources
    - ...
- target
  - classes
  - \*.jar/\*.war
  - MANIFEST.MF
- pom.xml

Папка src содержит внутри себя исходный код проекта, файлы ресурсов, настроек баз данных и прочие файлы, необходимые во время компиляции сборки. Файлы в подпапке main будут полностью скомпилированы и перемещены в папку target. Классы модульных и интеграционных тестов размещаются в подпапке test. Данные файлы используются только во время запуска

компиляции: в папку target они перемещены не будут. Основным назначением данных файлов является проверка корректности сборки.

Папка target содержит в подпапке classes скомпилированные классы. Файлы, не являющиеся java-классами будут перемещены в target без каких-либо изменений. Также в папке target формируются jar/war файлы, являющиеся библиотеками классов, исполняемыми модулями, либо сервлетами.

Файл pom.xml является главным конфигурационным файлом проекта в контексте Maven. Именно здесь описываются все зависимости, профили сборки, а также подключаются дополнительные плагины.

### **2.10.3 Maven-плагины, используемые в проекте**

- maven-jar-plugin – плагин сборки jar-файлов. Используется для создания jar-файлов с классами модульных тестов, а также для корректной упаковки dto-классов.
- maven-war-plugin – плагин для сборки сервлетов. Используется для сборки war-файлов, содержащих конфигурации сервлетов, а также файлы настроек баз данных.
- heroku-maven-plugin – плагин для удаленного размещения проекта на серверах Heroku.

### **2.10.4 Жизненный цикл сборки в контексте Maven**

Maven позволяет построить весь жизненный цикл сборки проекта, начиная от компиляции и заканчивая удаленным размещением готового проекта. Рассмотрим фазы жизненного цикла проекта подробно:

- 1) clean – очистка решения. На данном этапе содержимое папки target полностью удаляется. Предназначено для обеспечения актуальности библиотек и файлов, включенных в сборку.
- 2) validate – фаза проверки корректности сборки, а также проверка наличия всех данных проекта.

- 3) `compile` – компиляция `.java` файлов в `.class` файлы. На данной фазе сборки проверяется наличие синтаксических ошибок в файлах сборки.
- 4) `test` – фаза автоматического запуска модульных и интеграционных тестов сборки.
- 5) `package` – упаковка всех файлов в `.jar/.war`-модуль.
- 6) `install` – размещение полученных `.jar`-модулей в локальном репозитории.
- 7) `deploy` – размещение `jar`-модулей в удаленном репозитории.

Запуск каждой фазы проекта возможен как с использованием командной строки, так и с использованием плагина для интегрированной среды разработки.

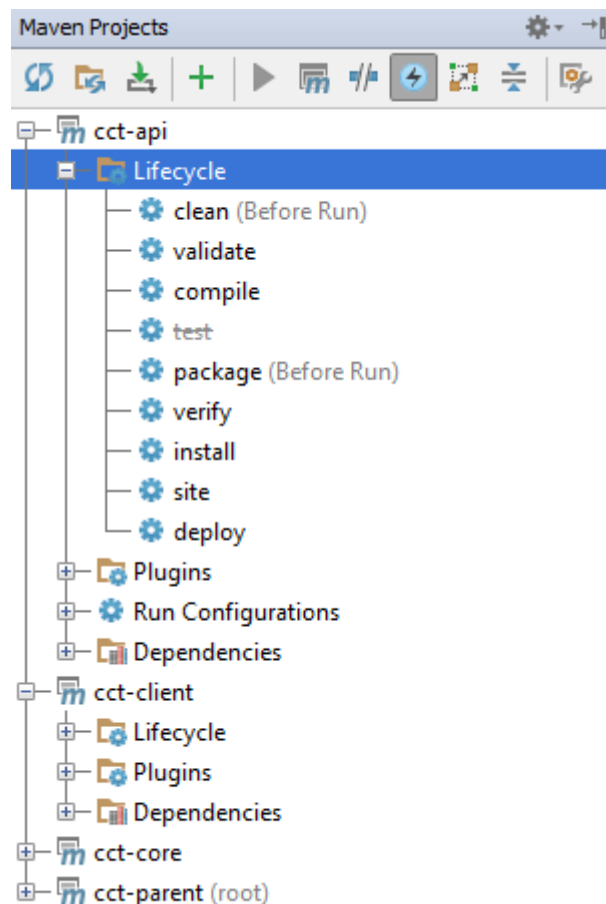


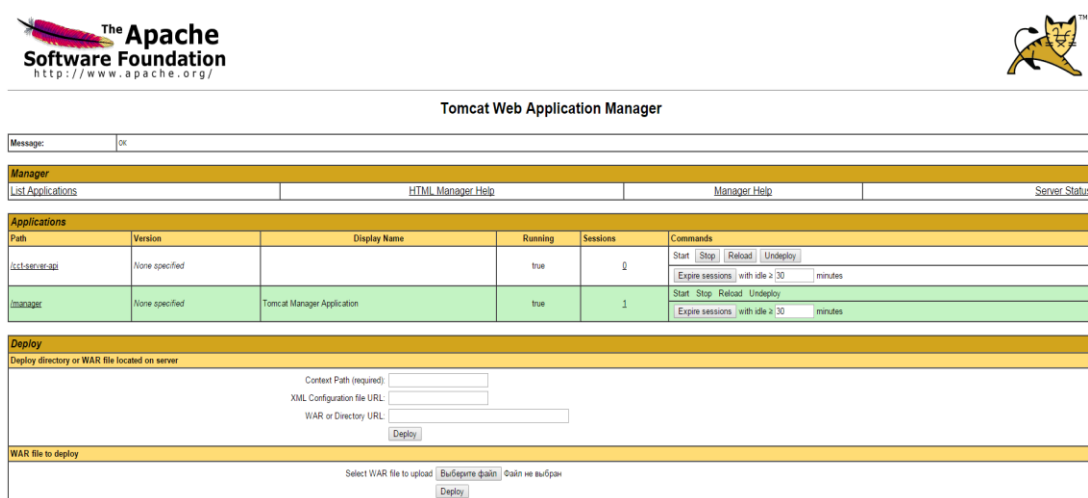
Рисунок 11 – Плагин Maven в IntelliJ IDEA

## 2.11 Удаленное размещение

### 2.11.1 Tomcat

Для того, чтобы к приложению можно было отправлять запросы, необходимо разместить сервлет-компоненту этого приложение в контейнере сервлетов. Наиболее распространенным инструментом для работы с сервлетами является Apache Tomcat<sup>11</sup>.

Данный инструмент предоставляет возможность размещение war-файлов, после чего к ним можно обращаться с помощью HTTP-запросов. Интерфейс стартовой страницы Tomcat представлен ниже.



The Apache Software Foundation  
<http://www.apache.org/>

Tomcat Web Application Manager

Message: OK

Manager

List Applications HTML Manager Help Manager Help Server Status

Path	Version	Display Name	Running	Sessions	Commands
localhost:8080	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes
manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle > 30 minutes

Deploy

Deploy directory or WAR file located on server

Context Path (required):

XML Configuration file URL:

WAR or Directory URL:

Deploy

WAR file to deploy

Select WAR file to upload

Deploy

Рисунок 12 - Tomcat

### 2.11.3 Heroku

Заключительным этапом разработки приложения C&STool стало размещение приложения на удаленном сервере.

В качестве системы удаленного размещения была выбрана облачная система Heroku<sup>12</sup>.

Данная система позволяет размещать приложения на многих языках программирования, в том числе и Java-приложения, для которых в Heroku встроен Apache Tomcat.

<sup>11</sup> <http://tomcat.apache.org/>

<sup>12</sup> <https://dashboard.heroku.com/apps>

Heroku интегрирует в размещаемый проект базу данных PostgreSQL и предоставляет URL для доступа к ней.

В настоящий момент API приложения является общедоступным.

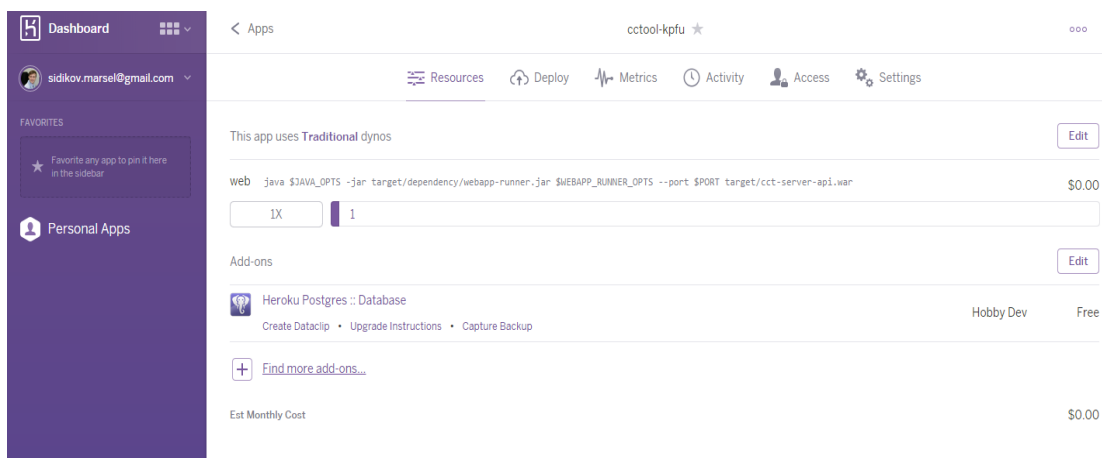


Рисунок 13 – Heroku

### 2.11.3 Advanced Rest Client

Для того, чтобы иметь возможность тестировать серверную часть проекта C&CTool было использовано специальное расширение Chrome Advanced Rest Client. Данное расширение позволяет отправлять различные запросы по определенному адресу и получать представления JSON или XML.

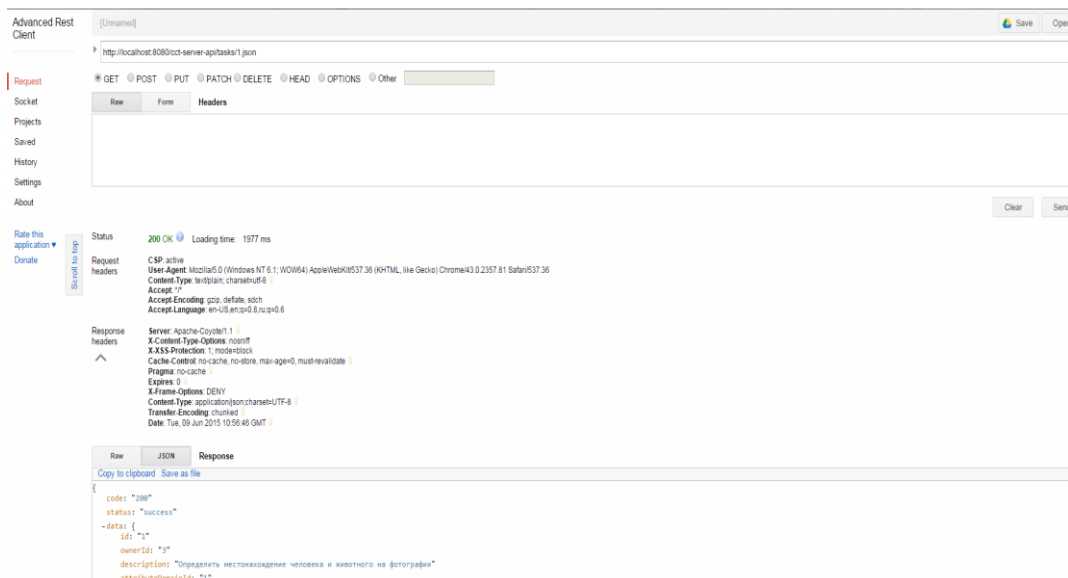


Рисунок 14 – Тестирование C&CTool с помощью Advanced Rest Client

## ЗАКЛЮЧЕНИЕ

В ходе выполнения настоящей дипломной работы было разработано клиент-серверное приложение C&STool для создания тренировочных множеств систем машинного обучения.

Наиболее важными результатами проведенной работы является реализация серверного API приложения, а также реализация клиентского компонента.

Данное приложение

- Имеет русскоязычную документацию серверной части, а также русскоязычный интерфейс клиентского модуля;
- включает систему регистрации и аутентификации;
- поддерживает возможность командной работы. Приложение выложено на удаленном сервере и его функционал общедоступен;
- предлагает программный интерфейс (API) для поддержки взаимодействия с различными типами клиентов. В целях удобства использования, документация по API приложения размещена в системе контроля версий;
- включает систему обратной связи между заказчиками и работниками;
- включает реализацию Desktop-клиента с использованием технологии JavaFX и клиента на платформе ОС Android, интерфейс и функционал которых ориентирован на конкретные задачи формирования тренировочных множеств для систем машинного обучения.

Таким образом, все поставленные перед дипломной работой задачи полностью выполнены.

В ближайшее время планируется дальнейшее развитие C&STool – а именно, его интеграция с популярными платежными системами и разработка клиентов для всех существующих мобильных и веб-платформ.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Christopher M. Bishop Pattern Recognition and Machine Learning [Текст]: учебное пособие / Christopher M. Bishop, Springer Science, Business Media, 2006 – 738 с.
2. The Crowd in the Cloud: Exploring the Future of Outsourcing [Электронный ресурс] - [http://www.lionbridge.com/files/2012/11/Lionbridge-White-Paper\\_The-Crowd-in-the-Cloud-final.pdf](http://www.lionbridge.com/files/2012/11/Lionbridge-White-Paper_The-Crowd-in-the-Cloud-final.pdf)
3. Tagging Human Activities in Video by Crowdsourcing [Электронный ресурс] - <http://dl.acm.org/citation.cfm?id=2461508>
4. Crowdsourcing grows up as online workers unite [Электронный ресурс] - [http://www.newscientist.com/article/mg21729036.200-crowdsourcing-grows-up-as-online-workers-unite.html#.VXbNbM\\_tlBd](http://www.newscientist.com/article/mg21729036.200-crowdsourcing-grows-up-as-online-workers-unite.html#.VXbNbM_tlBd)
5. The Language Demographics of Amazon Mechanical Turk [Электронный ресурс] - <http://www.cis.upenn.edu/~ccb/publications/language-demographics-of-mechanical-turk.pdf>
6. Машинное обучение [Электронный ресурс] - [http://www.uic.unn.ru/~zny/ml/Lectures/ml\\_pres.pdf](http://www.uic.unn.ru/~zny/ml/Lectures/ml_pres.pdf)
7. Monocular Pedestrian Detection: Survey and Experiments [Электронный ресурс] - <http://www.gavrila.net/pami09.pdf>
8. Erich Gamma Design Patterns: Elements of Reusable Object-Oriented Software [Текст]: учебное пособие / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison-Wesley, 1994 – 395 с.
9. Representational State Transfer (REST) [Электронный ресурс] - [https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
10. Web API Design - - Crafting Interfaces that Developers Love [Электронный ресурс] - <https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf>
11. Token Based Authentication, Implementation Demonstration [Электронный ресурс] - [http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token\\_based\\_authentication/](http://www.w3.org/2001/sw/Europe/events/foaf-galway/papers/fp/token_based_authentication/)



## ПРИЛОЖЕНИЕ

### Исходный код приложения C&STool

```
package ru.kpfu.tools.cct.rest.config;

import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.datasource.DriverManagerData-
Source;
import org.springframework.web.servlet.config.annotation.De-
faultServletHandlerConfigurer;
import org.springframework.web.servlet.config.annotation.Ena-
bleWebMvc;
import org.springframework.web.servlet.config.annota-
tion.WebMvcConfigurerAdapter;

import javax.sql.DataSource;
import java.net.URI;
import java.net.URISyntaxException;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@EnableWebMvc
@Configuration
@ComponentScan(basePackages = "ru.kpfu")
public class WebApplicationContext extends WebMvcConfigurerAdapter {

    /**
     * For heroku
     * @return
     * @throws URISyntaxException
     */
    /*
    @Bean
    public DataSource dataSource() throws URISyntaxException
    {
        DriverManagerDataSource dataSource = new DriverMan-
agerDataSource();

        String username = "dyeoorrkaoccab";
        String password = "Gp53klVfBfdveZsVVMWJrTefsG";
    }
}
```

```

        String dbUrl = "jdbc:postgresql://" + "ec2-107-20-222-
114.compute-1.amazonaws.com" + ":5432" +
            "/d6r6glb2gm4r30";

        dataSource.setUrl(dbUrl);
        dataSource.setUsername(username);
        dataSource.setPassword(password);

        return dataSource;
    }
    */

    @Bean
    public DataSource dataSource() throws URISyntaxException
    {
        DriverManagerDataSource dataSource = new DriverMan-
agerDataSource();

        String username = "postgres";
        String password = "1234";
        String dbUrl = "jdbc:postgresql://lo-
calhost:5432/cct";

        dataSource.setUrl(dbUrl);
        dataSource.setUsername(username);
        dataSource.setPassword(password);
        dataSource.setDriverClassName("org.post-
gresql.Driver");
        return dataSource;
    }

    @Override
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }
}

package ru.kpfu.tools.cct.rest.config;

import org.springframework.core.annotation.Order;
import org.springframework.web.context.request.RequestCon-
textListener;
import org.springframework.web.filter.CharacterEncodingFil-
ter;
import org.springframework.web.filter.DelegatingFilterProxy;
import org.springframework.web.filter.HttpPutFormContentFil-
ter;
import org.springframework.web.servlet.support.AbstractAnno-
tationConfigDispatcherServletInitializer;

```

```

import javax.servlet.*;

@Order(1)
public class WebAppInitializer extends AbstractAnnotationCon-
figDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class<?>[]{WebSecurityConfig.class};
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[]{WebApplicationContext.class};
    }

    @Override
    protected String[] getServletMappings() {
        return new String[]{"/*"};
    }

    @Override
    protected Filter[] getServletFilters() {
        DelegatingFilterProxy securityFilterChain = new Dele-
gatingFilterProxy("springSecurityFilterChain");
        CharacterEncodingFilter encodingFilter = new Charac-
terEncodingFilter();
        encodingFilter.setEncoding("UTF-8");
        encodingFilter.setForceEncoding(true);
        // MultipartFilter multipartFilter=new MultipartFil-
ter();

        return new Filter[]{
            encodingFilter,
            securityFilterChain,
            new HttpPutFormContentFilter()
        };
    }

    @Override
    public void onStartup(ServletContext servletContext)
throws ServletException {
        super.onStartup(servletContext);
    }

    @Override
    protected void registerContextLoaderListener(ServletCon-
text servletContext) {
        super.registerContextLoaderListener(servletContext);
        servletContext.addListener(new RequestContextLis-
tener());
    }
}

```

```

package ru.kpfu.tools.cct.rest.config;

import org.codehaus.jackson.map.ObjectMapper;
import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.Authenti-
cationManager;
import org.springframework.security.authentication.Authenti-
cationProvider;
import org.springframework.security.authentication.encod-
ing.ShaPasswordEncoder;
import org.springframework.security.config.annotation.authen-
tication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annota-
tion.web.builders.HttpSecurity;
import org.springframework.security.config.annota-
tion.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annota-
tion.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCrea-
tionPolicy;
import org.springframework.security.web.AuthenticationEntry-
Point;
import org.springframework.security.web.authentica-
tion.www.BasicAuthenticationFilter;
import ru.kpfu.tools.cct.rest.security.auth.Filter;
import ru.kpfu.tools.cct.rest.security.auth.RestAuthentica-
tionEntryPoint;
import ru.kpfu.tools.cct.rest.security.auth.TokenAuthPro-
vider;

@Configuration
@EnableWebSecurity
@ComponentScan(basePackages = {"ru.kpfu.tools.cct.rest"})
public class WebSecurityConfig extends WebSecurityConfig-
urerAdapter {

    @Autowired
    @Qualifier("domainAuthProvider")
    private AuthenticationProvider authenticationProvider;

    @Override
    protected void configure(HttpSecurity http) throws Excep-
tion {
        // AuthenticationEntryPoint unauthorizedEntryPoint
        = unauthorizedEntryPoint(mapper);

```

```

        http.
            csrf().disable().
            sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).
            and().
            authorizeRequests().antMatchers(HttpMethod.GET, "/tasks.json").hasRole("ADMIN");
        http.addFilterBefore(new Filter(authenticationManager()), BasicAuthenticationFilter.class);
    }

    // @Bean
    // public AuthenticationEntryPoint unauthorizedEntryPoint(ObjectMapper mapper) {
    //     RestAuthenticationEntryPoint restAuthenticationEntryPoint = new RestAuthenticationEntryPoint();
    //     restAuthenticationEntryPoint.setMsgMapper(mapper);
    //     return restAuthenticationEntryPoint;
    // }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authenticationProvider).
            authenticationProvider(tokenAuthenticationProvider());
    }

    @Bean
    public AuthenticationProvider tokenAuthenticationProvider() {
        return new TokenAuthProvider();
    }

    @Bean
    public ShaPasswordEncoder getShaPasswordEncoder() {
        return new ShaPasswordEncoder();
    }

    @Bean(name = "myAuthenticationManager")
    @Override
    public AuthenticationManager authenticationManagerBean()
    throws Exception {
        return super.authenticationManagerBean();
    }
}

package ru.kpfu.tools.cct.rest.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;

```

```

import org.springframework.security.authentication.encoding.ShaPasswordEncoder;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import ru.kpfu.tools.cct.core.services.UserService;
import ru.kpfu.tools.cct.core.services.models.User;
import ru.kpfu.tools.cct.rest.dto.UserDto;

@RestController
public class RegistrationController {

    @Autowired
    UserService userService;

    @Autowired
    ShaPasswordEncoder shaPasswordEncoder;

    @RequestMapping(value = "/users", method = RequestMethod.POST, produces = MediaType.APPLICATION_JSON_VALUE)
    public void registrationUser(@RequestBody UserDto userDto) {
        User user = new User();
        String passwordHash = shaPasswordEncoder.encodePassword(userDto.getPassword(), "");
        user.setLogin(userDto.getLogin());
        user.setRole(userDto.getRole());
        user.setPasswordHash(passwordHash);
        userService.addUser(user);
    }

    @RequestMapping(value = "/login", method = RequestMethod.POST)
    public String login() {
        return null;
    }
}

package ru.kpfu.tools.cct.rest.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.kpfu.tools.cct.core.services.models.Document;
import ru.kpfu.tools.cct.core.services.models.Domain;
import ru.kpfu.tools.cct.core.services.models.Recall;
import ru.kpfu.tools.cct.core.services.models.Task;

```

```

import ru.kpfu.tools.cct.rest.dto.*;
import ru.kpfu.tools.cct.rest.dto.converters.DtoAndEntityCon-
verter;
import ru.kpfu.tools.cct.core.services.WorkerServiceFacade;
import ru.kpfu.tools.cct.rest.utils.ResponseBuilder;

import java.util.List;
import java.util.Map;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@RestController
public class WorkerActionController {

    @Autowired
    private WorkerServiceFacade workerServiceFacade;

    @Autowired
    private DtoAndEntityConverter converter;

    @RequestMapping(value = "/tasks.json",
                    method = RequestMethod.GET, produces = Medi-
aType.APPLICATION_JSON_VALUE)
    public ResponseEntity<ResponseObjectDto> getTasks() {
        List<Task> tasks = workerServiceFacade.getAvaila-
bleTasks();
        TasksDto tasksDto = converter.fromTasks(tasks);
        return ResponseBuilder.buildResponseGet(tasksDto);
    }

    @RequestMapping(value = "/recalls.json",
                    method = RequestMethod.GET, produces = Medi-
aType.APPLICATION_JSON_VALUE)
    public ResponseEntity<ResponseObjectDto> getRecalls(@Re-
questParam int user) {
        List<Recall> recalls = workerServiceFa-
cade.getRecalls(user);
        RecallsDto recallsDto = converter.fromRecalls(re-
calls);
        return ResponseBuilder.buildResponseGet(recallsDto);
    }

    @RequestMapping(value = "/tasks/{task-id}.json",
                    method = RequestMethod.GET, produces = Medi-
aType.APPLICATION_JSON_VALUE)
    public ResponseEntity<ResponseObjectDto> getTask(@Path-
Variable("task-id") int taskId) {
        Task task = workerServiceFacade.getTask(taskId);
        TaskDto taskDto = converter.fromTask(task);
        return ResponseBuilder.buildResponseGet(taskDto);
    }
}

```

```

        @RequestMapping(value = "/tasks/assignments",
                        method = RequestMethod.POST, consumes = MediaType.APPLICATION_JSON_VALUE,
                        produces = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> taskAssign(@RequestBody TaskAssignmentDto taskAssignmentDto) {
            Map<String, Integer> dtoValues = converter.getIdsAsIntegerMap(taskAssignmentDto);
            workerServiceFacade.taskAssign(dtoValues.get("workerId"), dtoValues.get("taskId"));
            return ResponseBuilder.buildResponsePost(taskAssignmentDto);
        }

        @RequestMapping(value = "domains/{domain-id}.json",
                        method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> getDomain(@PathVariable("domain-id") int domainId) {
            Domain domain = workerServiceFacade.getDomain(domainId);
            DomainDto domainDto = converter.fromDomain(domain);
            return ResponseBuilder.buildResponseGet(domainDto);
        }

        @RequestMapping(value = "/tasks/{task-id}/documents.json",
                        method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> getDocuments(@PathVariable("task-id") int taskId) {
            List<Document> documents = workerServiceFacade.getDocuments(taskId);
            DocumentsDto documentsDto = converter.fromDocuments(documents);
            return ResponseBuilder.buildResponseGet(documentsDto);
        }

        @RequestMapping(value = "/tasks/{task-id}/documents/{document-id}.json",
                        method = RequestMethod.GET, produces = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> getDocument(@PathVariable("document-id") int documentId) {
            Document document = workerServiceFacade.getDocument(documentId);
            DocumentDto documentDto = converter.fromDocument(document);
            return ResponseBuilder.buildResponseGet(documentDto);
        }
    }

```



```

        @RequestMapping(value = "/tasks/complaints",
            method = RequestMethod.POST, produces = MediaType.APPLICATION_JSON_VALUE,
            consumes = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> complaintToTask(@RequestBody ComplaintDto complaintDto) {
            Map<String, Integer> dtoValues = converter.getIdsAsIntegerMap(complaintDto);
            workerServiceFacade.complaintToTask(dtoValues.get("workerId"), dtoValues.get("taskId"),
                complaintDto.getDescription());
            return ResponseBuilder.buildResponsePost(complaintDto);
        }

        @RequestMapping(value = "/tasks/{task-id}/documents/postponed",
            method = RequestMethod.POST, produces = MediaType.APPLICATION_JSON_VALUE,
            consumes = MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> postponeDocument(@RequestBody PostponeDocumentDto postponeDocumentDto) {
            Map<String, Integer> dtoValues = converter.getIdsAsIntegerMap(postponeDocumentDto);
            workerServiceFacade.postponeDocument(dtoValues.get("workerId"), dtoValues.get("documentId"));
            return ResponseBuilder.buildResponsePost(postponeDocumentDto);
        }

        @RequestMapping(value = "/tasks/{task-id}/documents/postponed.json", method = RequestMethod.GET, produces =
            MediaType.APPLICATION_JSON_VALUE)
        public ResponseEntity<ResponseObjectDto> getPostponedDocuments(@PathVariable("task-id") int taskId, @RequestParam
            ("worker") int workerId) {
            List<Document> documents = workerServiceFacade.getPostponedDocuments(workerId, taskId);
            DocumentsDto documentDtos = converter.fromDocuments(documents);
            return ResponseBuilder.buildResponseGet(documentDtos);
        }
    }

    package ru.kpfu.tools.cct.rest.dto.converters;

    import com.google.common.collect.ImmutableMap;
    import com.inspiresoftware.lib.dto.geda.adapter.BeanFactory;
    import com.inspiresoftware.lib.dto.geda.adapter.ValueConverter;

```

```

import com.inspiresoftware.lib.dto.geda.assembler.Assembler;
import com.inspiresoftware.lib.dto.geda.assembler.DTOAssembler;
import org.springframework.stereotype.Component;
import ru.kpfu.tools.cct.core.services.models.Document;
import ru.kpfu.tools.cct.core.services.models.Domain;
import ru.kpfu.tools.cct.core.services.models.Recall;
import ru.kpfu.tools.cct.rest.dto.*;
import ru.kpfu.tools.cct.core.services.models.Task;

import java.net.URL;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Component
public class DtoAndEntityConverterGedaImpl implements
DtoAndEntityConverter {

    private final String INT_TO_STR_ADAPTER_NAME = "Integer-
ToString";
    private final String URL_TO_STR_TASKS_ADAPTER_NAME =
"UrlToLinksInTasks";
    private final String URL_TO_STR_DOCUMENTS_ADAPTER_NAME =
"UrlToLinksInDocuments";

    private final ValueConverter integerToStringConverter =
new ValueConverter() {
        @Override
        public Object convertToDto(Object o, BeanFactory
beanFactory) {
            return String.valueOf(o);
        }

        @Override
        public Object convertToEntity(Object o, Object o1,
BeanFactory beanFactory) {
            return Integer.parseInt(o.toString());
        }
    };

    private final ValueConverter urlToLinksInTasksConverter =
new ValueConverter() {
        @Override
        public Object convertToDto(Object o, BeanFactory
beanFactory) {
            URL url = (URL)o;
            LinksInTaskDto linksInTaskDto = new
LinksInTaskDto();

```

```

        linksInTaskDto.setInstructionsFile(url.toString());
        return linksInTaskDto;
    }

    @Override
    public Object convertToEntity(Object o, Object ol,
BeanFactory beanFactory) {
        // TODO
        return null;
    }
};

    private final ValueConverter urlToLinksInDocumentsCon-
verter = new ValueConverter() {
        @Override
        public Object convertToDto(Object object, BeanFactory
beanFactory) {
            URL url = (URL)object;
            LinksInDocumentDto linksInDocumentDto = new
LinksInDocumentDto();
            linksInDocumentDto.setSelf(url.toString());
            return linksInDocumentDto;
        }

        @Override
        public Object convertToEntity(Object object, Object
oldEntity, BeanFactory beanFactory) {
            //TODO
            return null;
        }
    };

    private final Assembler taskAssembler = DTOAssem-
bler.newAssembler(TaskDto.class, Task.class);
    private final Assembler documentAssembler = DTOAssem-
bler.newAssembler(DocumentDto.class, Document.class);
    private final Assembler recallAssembler = DTOAssem-
bler.newAssembler(RecallDto.class, Recall.class);
    private final Assembler domainAssembler = DTOAssem-
bler.newAssembler(DomainDto.class, Domain.class);

    @Override
    public TaskDto fromTask(Task entity) {
        TaskDto taskDto = new TaskDto();
        Map<String, Object> adapters = new HashMap<String,
Object>();
        adapters.put(INT_TO_STR_ADAPTER_NAME, integerTo-
StringConverter);
        adapters.put(URL_TO_STR_TASKS_ADAPTER_NAME, url-
ToLinksInTasksConverter);
    }

```

```

        taskAssembler.assembleDto(taskDto, entity, adapters,
null);

        return taskDto;
    }

    private void taskDtosPrepare(List<TaskDto> taskDtos) {
        for (TaskDto taskDto : taskDtos) {
            taskDto.setAttributeDomainId(null);
            taskDto.setOwnerId(null);
            taskDto.setLinks(null);
        }
    }

    private void documentDtosPrepare(List<DocumentDto> docu-
mentDtos) {
        for (DocumentDto documentDto : documentDtos) {
            documentDto.setLinks(null);
        }
    }

    @Override
    public TasksDto fromTasks(List<Task> entities) {
        List<TaskDto> dtos = new LinkedList<TaskDto>();
        for (Task task : entities) {
            dtos.add(fromTask(task));
        }

        taskDtosPrepare(dtos);
        TasksDto result = new TasksDto();
        result.setTasks(dtos);
        return result;
    }

    @Override
    public RecallsDto fromRecalls(List<Recall> entities) {
        List<RecallDto> dtos = new LinkedList<RecallDto>();
        for (Recall recall : entities) {
            dtos.add(fromRecall(recall));
        }

        RecallsDto recallsDto = new RecallsDto();
        recallsDto.setRecalls(dtos);
        return recallsDto;
    }

    private RecallDto fromRecall(Recall entity) {
        RecallDto recallDto = new RecallDto();
        Map<String, Object> adapters = new HashMap<String,
Object>();

```

```

        adapters.put(INT_TO_STR_ADAPTER_NAME, integerTo-
StringConverter);
        recallAssembler.assembleDto(recallDto, entity, adapt-
ers, null);
        return recallDto;
    }

    @Override
    public DocumentDto fromDocument(Document entity) {
        DocumentDto documentDto = new DocumentDto();
        Map<String, Object> adapters = new HashMap<String,
Object>();
        adapters.put(INT_TO_STR_ADAPTER_NAME, integerTo-
StringConverter);
        adapters.put(URL_TO_STR_DOCUMENTS_ADAPTER_NAME, url-
ToLinksInDocumentsConverter);

        documentAssembler.assembleDto(documentDto, entity,
adapters, null);

        return documentDto;
    }

    @Override
    public DocumentsDto fromDocuments(List<Document> enti-
ties) {
        List<DocumentDto> documentDtos = new LinkedList<Docu-
mentDto>();
        for (Document document : entities) {
            documentDtos.add(fromDocument(document));
        }

        documentDtosPrepare(documentDtos);
        DocumentsDto result = new DocumentsDto();
        result.setDocuments(documentDtos);

        return result;
    }

    @Override
    public DomainDto fromDomain(Domain entity) {
        DomainDto domainDto = new DomainDto();
        Map<String, Object> adapters = new HashMap<String,
Object>();
        adapters.put(INT_TO_STR_ADAPTER_NAME, integerTo-
StringConverter);
        domainAssembler.assembleDto(domainDto, entity, adapt-
ers, null);
        return domainDto;
    }

    @Override

```

```

        public Map<String, Integer> getIdsAsInteger-
Map(TaskAssignmentDto dto) {
            int workerId = Integer.parseInt(dto.getWorkerId());
            int taskId = Integer.parseInt(dto.getTaskId());
            return ImmutableMap.of("workerId", workerId,
"taskId", taskId);
        }

        @Override
        public Map<String, Integer> getIdsAsIntegerMap(Com-
plaintDto dto) {
            int workerId= Integer.parseInt(dto.getWorkerId());
            int taskId = Integer.parseInt(dto.getTaskId());
            return ImmutableMap.of("workerId", workerId,
"taskId", taskId);
        }

        @Override
        public Map<String, Integer> getIdsAsIntegerMap(Post-
poneDocumentDto postponeDocumentDto) {
            int workerId = Integer.parseInt(postponeDocu-
mentDto.getWorkerId());
            int documentId = Integer.parseInt(postponeDocu-
mentDto.getDocumentId());
            return ImmutableMap.of("workerId", workerId, "docu-
mentId", documentId);
        }
    }

    package ru.kpfu.tools.cct.rest.security.auth;

    import org.springframework.security.core.GrantedAuthority;
    import org.springframework.security.web.authentica-
tion.preauth.PreAuthenticatedAuthenticationToken;

    import java.util.Collection;

    public class AuthenticationWithToken extends PreAuthenti-
catedAuthenticationToken {
        public AuthenticationWithToken(Object aPrincipal, Object
aCredentials) {
            super(aPrincipal, aCredentials);
        }

        public AuthenticationWithToken(Object aPrincipal, Object
aCredentials, Collection<? extends GrantedAuthority> anAuthori-
ties) {
            super(aPrincipal, aCredentials, anAuthorities);
        }

        public void setToken(String token) {
            setDetails(token);
        }
    }

```

```

        public String getToken() {
            return (String)getDetails();
        }
    }

package ru.kpfu.tools.cct.rest.security.auth;

import org.apache.http.HttpStatus;
import org.codehaus.jackson.map.ObjectMapper;
import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.security.authentication.*;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityCon-
textHolder;
import org.springframework.security.web.authentica-
tion.preauth.PreAuthenticatedAuthenticationToken;
import org.springframework.web.filter.GenericFilterBean;
import org.springframework.web.util.UrlPathHelper;
import ru.kpfu.tools.cct.core.services.models.User;
import ru.kpfu.tools.cct.rest.dto.ResponseObjectDto;
import ru.kpfu.tools.cct.rest.dto.TokenResponse;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class Filter extends GenericFilterBean {

    private AuthenticationManager authenticationManager;

    public Filter(AuthenticationManager authenticationMan-
ager) {
        this.authenticationManager = authenticationManager;
    }

    @Override
    public void doFilter(ServletRequest servletRequest,
        ServletResponse servletResponse, FilterChain filterChain) throws
        IOException, ServletException {
        HttpServletRequest httpRequest = (HttpS-
ervletRequest)servletRequest;
        HttpServletResponse httpResponse = (HttpS-
ervletResponse)servletResponse;

        String login = httpRequest.getHeader("X-Auth-
Username");

```

```

        String password = httpRequest.getHeader("X-Auth-Pass-
word");
        String token = httpRequest.getHeader("X-Auth-Token");

        String resourcePath = new UriPath-
Helper().getPathWithinApplication(httpRequest);

        try {
            if (postToAuthenticate(httpRequest, re-
sourcePath)) {
                try {
                    processUsernamePasswordAuthentica-
tion(httpResponse, login, password);
                } catch (Exception e) {
                    throw new IllegalArgumentException(e);
                }
                return;
            }
            if (token != null) {
                try {
                    processTokenAuthentication(token);
                } catch (BadCredentialsException e) {
                    throw new IllegalArgumentException();
                }
            }
            filterChain.doFilter(servletRequest,
servletResponse);
        } catch (InternalAuthenticationServiceException in-
ternalAuthenticationServiceException) {
            SecurityContextHolder.clearContext();
            httpResponse.sendError(HttpServletResponse.SC_IN-
TERNAL_SERVER_ERROR);
        }
    }

    private void processTokenAuthentication(String token) {
        Authentication resultOfAuthentication = tryToAuthen-
ticateWithToken(token);
        SecurityContextHolder.getContext().setAuthentica-
tion(resultOfAuthentication);
    }

    private Authentication tryToAuthenticateWithToken(String
token) {
        PreAuthenticatedAuthenticationToken requestAuthenti-
cation = new PreAuthenticatedAuthenticationToken(token, null);
        Authentication authentication = tryToAuthenticate(re-
questAuthentication);
        return authentication;
    }

    private void processUsernamePasswordAuthentication(HttpS-
ervletResponse httpResponse, String username, String password) {

```



```

        Authentication resultAuthentication = tryToAuthenticateWithUsernameAndPassword(username, password);
        SecurityContextHolder.getContext().setAuthentication(resultAuthentication);

        User principal = (User)resultAuthentication.getPrincipal();

        TokenResponse tokenResponse = new TokenResponse(resultAuthentication.getDetails().toString());
        ResponseObjectDto responseObjectDto = new ResponseObjectDto("201", "success", tokenResponse);
        ObjectMapper mapper = new ObjectMapper();
        httpResponse.setStatus(HttpStatus.SC_CREATED);
        httpResponse.addHeader("Content-Type", "application/json");
        try {
            String asString = mapper.writeValueAsString(responseObjectDto);
            httpResponse.getWriter().print(asString);
        } catch (IOException e) {
            throw new IllegalArgumentException(e);
        }
    }

    private Authentication tryToAuthenticateWithUsernameAndPassword(String username, String password) {
        UsernamePasswordAuthenticationToken requestAuthentication = new UsernamePasswordAuthenticationToken(username, password);
        return tryToAuthenticate(requestAuthentication);
    }

    private Authentication tryToAuthenticate(Authentication requestAuthentication) {
        Authentication responseAuthentication = authenticationManager.authenticate(requestAuthentication);
        if (responseAuthentication == null || !responseAuthentication.isAuthenticated()) {
            throw new IllegalArgumentException();
        }
        return responseAuthentication;
    }

    private boolean postToAuthenticate(HttpServletRequest request, String resourcePath) {
        return resourcePath.contains("/login") && request.getMethod().equals("POST");
    }
}

package ru.kpfu.tools.cct.rest.security.auth;

import org.codehaus.jackson.map.ObjectMapper;

```

```

import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.security.web.access.AccessDeniedHandler;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class RestAuthenticationEntryPoint implements AuthenticationEntryPoint, AccessDeniedHandler {
    private ObjectMapper msgMapper;

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException authException) throws IOException, ServletException {
        writeError(response);
    }

    private void writeError(HttpServletResponse response) {
        /** TODO
        ErrorResponse errorResponse = new ErrorResponse();
        errorResponse.setError("Not allowed for user");
        errorResponse.setMsg("not_allowed");
        WebUtil.objToResponse(errorResponse, response, HttpServletResponse.SC_METHOD_NOT_ALLOWED, msgMapper);
        */
    }

    public void setMsgMapper(ObjectMapper msgMapper) {
        this.msgMapper = msgMapper;
    }

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response, AccessDeniedException accessDeniedException) throws IOException, ServletException {
        writeError(response);
    }
}

package ru.kpfu.tools.cct.rest.security.auth;

import org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;

public class TokenAuthProvider implements AuthenticationProvider{

    @Autowired
    private TokenService tokenService;

    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationException {
        String token = (String)authentication.getPrincipal();
        return tokenService.retrieve(token);
    }

    @Override
    public boolean supports(Class<?> aClass) {
        return true;
    }
}

package ru.kpfu.tools.cct.rest.security.auth;

import com.sun.org.apache.xml.internal.security.utils.Base64;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.stereotype.Service;
import ru.kpfu.tools.cct.core.services.UserService;
import ru.kpfu.tools.cct.core.services.models.User;

import java.util.UUID;

import static java.util.Arrays.asList;

@Service
public class TokenService {

    @Autowired
    UserService userService;

    public String generateToken() {
        String token = UUID.randomUUID().toString();
        byte[] base64 = Base64.encode(token.getBytes()).getBytes();
        return new String(base64);
    }
}

```

```

        public Authentication retrieve(String token) {
            User user = userService.getUserByToken(token);
            return new AuthenticationWithToken(user,
null,asList(new SimpleGrantedAuthority(user.getRole())));
        }

        public void store(String newToken, Authentication authentication) {
            User user = (User)authentication.getPrincipal();
            userService.storeTokenForUser(user, newToken);
        }
    }

package ru.kpfu.tools.cct.rest.security.auth;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.authentication.encoding.ShaPasswordEncoder;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import ru.kpfu.tools.cct.core.services.UserService;
import ru.kpfu.tools.cct.core.services.models.User;

import java.util.List;

import static java.util.Arrays.asList;

@Component("domainAuthProvider")
public class UsernameWithPasswordAuthProvider implements AuthenticationProvider {

    @Autowired
    private ShaPasswordEncoder shaPasswordEncoder;

    @Autowired
    private UserService userService;

    @Autowired

```

```

        private TokenService tokenService;

        @Override
        public Authentication authenticate(Authentication authentication) throws AuthenticationException {
            Object passwordObject = authentication.getCredentials();

            if (passwordObject == null) {
                passwordObject = "";
            }

            String password = passwordObject.toString();
            String passwordHash = decodePassword(password);
            String login = authentication.getName();

            //TODO verify password and login
            User user = userService.getUser(login);
            if (passwordHash.equals(user.getPasswordHash())) {
                String newToken = tokenService.generateToken();
                List<SimpleGrantedAuthority> anAuthority =
asList(new SimpleGrantedAuthority("ROLE_" + user.getRole()));
                AuthenticationWithToken authenticationWithToken =
new AuthenticationWithToken(user, null, anAuthority);
                authenticationWithToken.setToken(newToken);
                SecurityContextHolder.getContext().setAuthentication(authenticationWithToken);
                tokenService.store(newToken, authenticationWithToken);

                return authenticationWithToken;
            } else throw new IllegalArgumentException();
        }

        private String decodePassword(String password) {
            return shaPasswordEncoder.encodePassword(password,
            "");
        }

        @Override
        public boolean supports(Class<?> aClass) {
            return aClass.equals(UsernamePasswordAuthenticationToken.class);
        }
    }

    package ru.kpfu.tools.cct.rest.utils;

    import org.springframework.http.HttpHeaders;
    import org.springframework.http.HttpStatus;
    import org.springframework.http.MediaType;
    import org.springframework.http.ResponseEntity;
    import org.springframework.web.bind.annotation.ControllerAdvice;

```

```

import org.springframework.web.servlet.mvc.method.annotation.
ResponseEntityExceptionHandler;
import ru.kpfu.tools.cct.core.services.exceptions.*;
import ru.kpfu.tools.cct.rest.dto.ErrorObjectDto;
import org.springframework.web.context.request.WebRequest;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@ControllerAdvice
public class ExceptionHandler extends ResponseEntityExcep-
tionHandler {

    HttpHeaders headers = createContentTypeHeaders();

    @org.springframework.web.bind.annotation.ExceptionHan-
dler({TaskNotFoundException.class, DocumentNotFoundExcep-
tion.class, UserNotFoundException.class,
        DomainNotFoundException.class})
    public ResponseEntity<Object> handleInvalidTaskRe-
quest(RuntimeException e, WebRequest request) {
        ErrorObjectDto errorObjectDto = createErrorOb-
jectDto("404", e);
        return handleExceptionInternal(e, errorObjectDto,
headers, HttpStatus.NOT_FOUND, request);
    }

    @org.springframework.web.bind.annotation.ExceptionHan-
dler({HasUncompletedTasksException.class, TaskNotAs-
signedToWorkerException.class})
    public ResponseEntity<Object> handleInvalidAssignmentRe-
quest(RuntimeException e, WebRequest request) {
        ErrorObjectDto errorObjectDto = createErrorOb-
jectDto("403", e);
        return handleExceptionInternal(e, errorObjectDto,
headers, HttpStatus.FORBIDDEN, request);
    }

    private ErrorObjectDto createErrorObjectDto(String code,
RuntimeException e) {
        return new ErrorObjectDto(code, "error", e.getMes-
sage(), e.getClass().getSimpleName());
    }

    private HttpHeaders createContentTypeHeaders() {
        HttpHeaders result = new HttpHeaders();
        result.setContentType(MediaType.APPLICATION_JSON);
        return result;
    }
}

package ru.kpfu.tools.cct.rest.utils;

```

```

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import ru.kpfu.tools.cct.rest.dto.DataTransferObject;
import ru.kpfu.tools.cct.rest.dto.ResponseObjectDto;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
public class ResponseBuilder {

    public static ResponseEntity<ResponseObjectDto> buildResponsePost(DataTransferObject data) {
        ResponseObjectDto body = new ResponseObjectDto("201", "success", data);
        ResponseEntity<ResponseObjectDto> response = new ResponseEntity<ResponseObjectDto>(body, HttpStatus.CREATED);
        return response;
    }

    public static ResponseEntity<ResponseObjectDto> buildResponseGet(DataTransferObject data) {
        ResponseObjectDto body = new ResponseObjectDto("200", "success", data);
        ResponseEntity<ResponseObjectDto> response = new ResponseEntity<ResponseObjectDto>(body, HttpStatus.OK);
        return response;
    }

}

package ru.kpfu.tools.cct.core.services.dao;
import ru.kpfu.tools.cct.core.services.models.Task;

import java.net.URL;

public class TaskDaoData {

    public static final Task TASK_1 = new Task(0, 0, "simple_task", 0, getURL("file://instructions.pdf"));
    public static final Task TASK_2 = new Task(1, 0, "simple_task_2", 0, getURL("file://instructions_2.pdf"));

    private static URL getURL(String url) {
        try {
            return new URL(url);
        } catch (Exception e) {
            throw new IllegalArgumentException();
        }
    }

}

package ru.kpfu.tools.cct.rest.controllers;

```

```

import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.http.HttpHeaders;
import org.springframework.security.web.FilterChainProxy;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.con-
text.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfigura-
tion;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.result.MockMvcRe-
sultMatchers;
import org.springframework.test.web.servlet.setup.Mock-
MvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import ru.kpfu.tools.cct.rest.config.WebAppContext;
import ru.kpfu.tools.cct.rest.config.WebSecurityConfig;

import javax.annotation.Resource;

import static java.util.Arrays.asList;
import static org.springframework.test.web.servlet.re-
quest.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.re-
quest.MockMvcRequestBuilders.post;

@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(classes = {WebSecurityConfig.class,
WebAppContext.class })
public class Integration {

    @Resource
    private FilterChainProxy springSecurityFilterChain;

    @Autowired
    private WebApplicationContext webApplicationContext;

    MockMvc mockMvc;

    @Before
    public void setUp() {
        mockMvc = MockMvcBuilders.webAppContextSetup(webAp-
plicationContext).addFilter(springSecurityFilterChain).build();
    }

    @Test
    public void test() throws Exception {
        HttpHeaders headers = new HttpHeaders();
        headers.put("X-Auth-Username", asList("mainAdmin"));
        headers.put("X-Auth-Password", asList("cct-admin"));
    }

```



```

        HttpHeaders headers1 = new HttpHeaders();
        headers1.put("X-Auth-Token", asList("YWE5ZjQ1NWMtND-
NjYi00ZGU3LWFjMDMtNTQ2NjglMmFjN2M4"));
        //mockMvc.perform(post("/login").headers(headers)).andExpect(MockMvcResultMatchers.status().isOk());
        mockMvc.perform(get("/tasks.json").headers(headers1)).andExpect(MockMvcResultMatchers.status().isOk());
    }
}

package ru.kpfu.tools.cct.rest.controllers;

import org.mockito.Mockito;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import ru.kpfu.tools.cct.core.services.WorkerServiceFacade;

@Configuration
public class TestWebApplicationContext {

}

package ru.kpfu.tools.cct.rest.controllers;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.common.collect.Lists;
import org.hamcrest.core.Is;
import org.junit.Before;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.mockito.Matchers;
import org.mockito.Mockito;
import org.springframework.beans.factory.annotation.Autowired;
wired;
import org.springframework.http.MediaType;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.con-
text.junit4.SpringJUnit4ClassRunner;
import org.springframework.test.context.web.WebAppConfigura-
tion;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvc-
MvcRequestBuilders;
import org.springframework.test.web.servlet.result.MockMvcRe-
sultMatchers;
import org.springframework.test.web.servlet.setup.MockMvc-
MvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import ru.kpfu.tools.cct.core.services.WorkerServiceFacade;
import ru.kpfu.tools.cct.core.services.exceptions.TaskNot-
FoundException;

```

```

import ru.kpfu.tools.cct.rest.config.WebApplicationContext;
import ru.kpfu.tools.cct.rest.dto.TaskAssignmentDto;

import static org.hamcrest.core.Is.is;
import static org.mockito.Mockito.doThrow;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.verify;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static ru.kpfu.tools.cct.core.services.dao.TaskDaoData.*;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes = {TestWebApplicationContext.class,
WebApplicationContext.class })
@WebAppConfiguration
public class WorkerActionControllerTest {

    private MockMvc mockMvc;

    @Autowired
    WorkerServiceFacade workerServiceFacade;

    @Autowired
    WebApplicationContext webApplicationContext;

    final ObjectMapper objectMapper = new ObjectMapper();
    @Before
    public void setUp() throws Exception {
        Mockito.reset(workerServiceFacade);
        mockMvc = MockMvcBuilders.webApplicationContextSetup(webAp-
plicationContext).build();
        Mockito.doThrow(TaskNotFoundException.class).when(workerServiceFacade).getTask(Match-
ers.anyInt());
        Mockito.doReturn(TASK_1).when(workerServiceFa-
cade).getTask(TASK_1.getId());
        Mockito.doReturn(Lists.newArrayList(TASK_1,
TASK_2)).when(workerServiceFacade).getAvailableTasks();
    }

    @Test
    public void testGetTask() throws Exception {
        mockMvc.perform(get("/tasks/{task-id}.json",
TASK_1.getId()).contentType(MediaType.APPLICATION_JSON))

```

```

        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.id", Is.is(String.valueOf(TASK_1.getId()))))
        .andExpect(MockMvcResultMatchers.jsonPath("$.authorId", Is.is(String.valueOf(TASK_1.getOwnerId()))))
        .andExpect(MockMvcResultMatchers.jsonPath("$.description", is(TASK_1.getDescription()))
        .andExpect(MockMvcResultMatchers.jsonPath("$.attributeDomainId", Is.is(String.valueOf(TASK_1.getAttributeDomainId()))))
        .andExpect(MockMvcResultMatchers.jsonPath("$.links.instructionsFile", is(TASK_1.getInstructions().toString())));
    }

    @Test
    public void testGetTaskForIncorrectTaskId() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/tasks/{task-id}.json", 1).contentType(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isNotFound())
        .andExpect(MockMvcResultMatchers.jsonPath("$.code", Is.is("404")))
        .andExpect(MockMvcResultMatchers.jsonPath("$.status", Is.is("error")))
        .andExpect(MockMvcResultMatchers.jsonPath("$.message", Is.is("Id of task is invalid")))
        .andExpect(MockMvcResultMatchers.jsonPath("$.data", Is.is("TaskNotFoundException")));
    }

    @Test
    public void testGetTasks() throws Exception {
        mockMvc.perform(MockMvcRequestBuilders.get("/tasks.json").contentType(MediaType.APPLICATION_JSON))
        .andExpect(MockMvcResultMatchers.status().isOk())
        .andExpect(MockMvcResultMatchers.jsonPath("$.tasks[0].id", Is.is(String.valueOf(TASK_1.getId()))))
        .andExpect(MockMvcResultMatchers.jsonPath("$.tasks[0].description", is(TASK_1.getDescription()))
        .andExpect(MockMvcResultMatchers.jsonPath("$.tasks[1].id", Is.is(String.valueOf(TASK_2.getId()))))
        .andExpect(MockMvcResultMatchers.jsonPath("$.tasks[1].description", is(TASK_2.getDescription())));
    }

    @Test
    public void testPostTaskAssignment() throws Exception {

```

```

        TaskAssignmentDto assignmentDto = new TaskAssign-
mentDto();
        assignmentDto.setTaskId("1");
        assignmentDto.setWorkerId("1");

        String json = objectMapper.writeValueAsString(assign-
mentDto);

        mockMvc.perform(post("/tasks/assignments").con-
tentType(MediaType.APPLICATION_JSON)
                .content(json.getBytes()))
                .andExpect(status().isCreated());
        verify(workerServiceFacade).taskAssign(1, 1);
    }
}

package ru.kpfu.tools.cct.core.services.dao.documents;

import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.stereotype.Repository;
import ru.kpfu.tools.cct.core.services.dao.utils.ver-
ify.DaoArgumentsVerifier;
import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.Params-
Mapper;
import ru.kpfu.tools.cct.core.ser-
vices.dao.utils.jdbc.SqlQueryExecutor;
import ru.kpfu.tools.cct.core.services.models.Document;

import org.springframework.jdbc.core.RowMapper;

import java.net.MalformedURLException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;

import static java.util.Arrays.asList;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Repository
public class DocumentsDaoJdbcImpl implements DocumentsDao {

    @Autowired
    DaoArgumentsVerifier verifier;

    @Autowired
    ParamsMapper paramsMapper;

    @Autowired
    SqlQueryExecutor sqlQueryExecutor;

```

```

        //language=SQL
        private static final String SQL_GET_DOCUMENTS_OF_TASK =
            "SELECT * FROM documents WHERE (task_id =
:taskId)";

        //language=SQL
        private static final String SQL_GET_DOCUMENT_BY_ID =
            "SELECT * FROM documents WHERE (id = :documen-
tId)";

        //language=SQL
        private static final String SQL_INSERT_POSTPONED_DOCUMENT
=
            "INSERT INTO postponed_documents VALUES
(:workerId, :documentId)";

        //language=SQL
        private static final String SQL_GET_POSTPONED_DOCU-
MENTS_BY_IDS =
            "SELECT * FROM documents d, tasks t, post-
poned_documents pd WHERE (pd.worker_id = :workerId AND d.id = "
+
                "pd.document_id AND t.id = :taskId)";

        private final RowMapper<Document> DOCUMENT_ROW_MAPPER =
new RowMapper<Document>() {
            @Override
            public Document mapRow(ResultSet resultSet, int i)
throws SQLException {
                try {
                    return new Document(resultSet.get-
Int("id"),resultSet.getInt("task_id"),re-
sultSet.getString("file_name"),
                        resultSet.getString("type"), re-
sultSet.getString("folder_name"), resultSet.getInt("size"),
                        new URL(resultSet.getString("url")));
                } catch (MalformedURLException e) {
                    throw new IllegalArgumentException(e);
                }
            }
        };

        @Override
        public List<Document> getDocuments(int taskId) {
            verifier.verifyTask(taskId);
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("taskId"), asList(taskId));
            return sqlQueryExecutor.queryForObjects(SQL_GET_DOCU-
MENTS_OF_TASK, paramMap, DOCUMENT_ROW_MAPPER);
        }

        @Override
        public Document getDocument(int workerId) {

```

```

        verifier.verifyDocument(workerId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("documentId"), asList(workerId));
        return sqlQueryExecutor.queryForObject(SQL_GET_DOCU-
MENT_BY_ID, paramMap, DOCUMENT_ROW_MAPPER);
    }

    @Override
    public void postponeDocument(int workerId, int documen-
tId) {
        verifier.verifyDocumentAndWorkerCompatibil-
ity(workerId, documentId);
        verifier.verifyAlreadyPostponedDocument(workerId,
documentId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "documentId"), asList(workerId,
documentId));
        sqlQueryExecutor.updateQuery(SQL_INSERT_POST-
PONED_DOCUMENT ,paramMap);
    }

    @Override
    public List<Document> getPostponedDocuments(int workerId,
int taskId) {
        verifier.verifyAssignment(workerId, taskId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "taskId"), asList(workerId, taskId));
        return sqlQueryExecutor.queryForObjects(SQL_GET_POST-
PONED_DOCUMENTS_BY_IDS, paramMap, DOCUMENT_ROW_MAPPER);
    }
}

package ru.kpfu.tools.cct.core.services.dao.domains;

import com.google.common.base.Splitter;
import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Service;
import ru.kpfu.tools.cct.core.services.dao.utils.ver-
ify.DaoArgumentsVerifier;
import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.Params-
Mapper;
import ru.kpfu.tools.cct.core.ser-
vices.dao.utils.jdbc.SqlQueryExecutor;
import ru.kpfu.tools.cct.core.services.models.Domain;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;

import static java.util.Arrays.asList;

```

```

@Service
public class DomainsDaoImpl implements DomainsDao {

    @Autowired
    DaoArgumentsVerifier verifier;

    @Autowired
    ParamsMapper paramsMapper;

    @Autowired
    SqlQueryExecutor sqlQueryExecutor;

    //language=SQL
    private final String SQL_GET_DOMAIN_BY_ID =
        "SELECT * FROM domains WHERE (id = :domainId)";

    private final RowMapper<Domain> DOMAIN_ROW_MAPPER = new
RowMapper<Domain>() {
        @Override
        public Domain mapRow(ResultSet resultSet, int i)
throws SQLException {
            int id = resultSet.getInt("id");
            String values = resultSet.getString("domain_val-
ues");

            Splitter splitter = Splitter.on(", ");
            List<String> parsedValues = splitter.splitTo-
List(values);

            Domain domain = new Domain(id, parsedValues);
            return domain;
        }
    };

    @Override
    public Domain getDomain(int domainId) {
        verifier.verifyDomain(domainId);
        Map<String, Object> paramsMap = paramsMapper.as-
Map(asList("domainId"), asList(domainId));
        return sqlQueryExecutor.queryForObject(SQL_GET_DO-
MAIN_BY_ID, paramsMap, DOMAIN_ROW_MAPPER);
    }
}

package ru.kpfu.tools.cct.core.services.dao.recalls;

import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Service;
import ru.kpfu.tools.cct.core.services.dao.utils.ver-
ify.DaoArgumentsVerifier;
import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.Params-
Mapper;
import ru.kpfu.tools.cct.core.ser-
vices.dao.utils.jdbc.SqlQueryExecutor;

```

```

import ru.kpfu.tools.cct.core.services.models.Recall;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;

import static java.util.Arrays.asList;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Service
public class RecallsDaoImpl implements RecallsDao {

    @Autowired
    DaoArgumentsVerifier verifier;

    @Autowired
    ParamsMapper paramsMapper;

    @Autowired
    SqlQueryExecutor sqlQueryExecutor;

    //language=SQL
    private final String SQL_GET_RECALLS_BY_USER_ID =
        "SELECT * FROM recalls WHERE (user_id =
:userId)";

    private final RowMapper<Recall> RECALL_ROW_MAPPER = new
RowMapper<Recall>() {
        @Override
        public Recall mapRow(ResultSet resultSet, int i)
throws SQLException {
            return new Recall(resultSet.getInt("user_id"),
resultSet.getInt("author_id"), resultSet.getInt("mark"),
resultSet.getString("recall_text"));
        }
    };

    @Override
    public List<Recall> getRecalls(int userId) {
        verifier.verifyUser(userId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("userId"), asList(userId));
        return sqlQueryExecutor.queryForObjects(SQL_GET_RE-
CALLS_BY_USER_ID,paramMap, RECALL_ROW_MAPPER);
    }

}

package ru.kpfu.tools.cct.core.services.dao.tasks;

```



```

import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.stereotype.Repository;
import org.springframework.jdbc.core.RowMapper;
import ru.kpfu.tools.cct.core.services.dao.utils.ver-
ify.DaoArgumentsVerifier;
import ru.kpfu.tools.cct.core.ser-
vices.dao.utils.jdbc.ParamsMapper;
import ru.kpfu.tools.cct.core.ser-
vices.dao.utils.jdbc.SqlQueryExecutor;
import ru.kpfu.tools.cct.core.services.models.Task;

import java.net.MalformedURLException;
import java.net.URL;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;
import static java.util.Arrays.asList;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Repository
public class TasksDaoJdbcImpl implements TasksDao {

    @Autowired
    private DaoArgumentsVerifier verifier;

    @Autowired
    private ParamsMapper paramsMapper;

    @Autowired
    private SqlQueryExecutor sqlQueryExecutor;

    static TasksDaoJdbcImpl build(DaoArgumentsVerifier veri-
fier, ParamsMapper paramsMapper, SqlQueryExecutor
sqlQueryExecutor) {
        TasksDaoJdbcImpl result = new TasksDaoJdbcImpl();

        result.verifier = verifier;
        result.paramsMapper = paramsMapper;
        result.sqlQueryExecutor = sqlQueryExecutor;

        return result;
    }

    //language=SQL
    static final String SQL_GET_TASKS =
        "SELECT * FROM tasks";

    //language=SQL
    static final String SQL_GET_TASK_BY_ID =

```

```

        "SELECT * FROM tasks WHERE (id = :taskId)";

//language=SQL
static final String SQL_INSERT_TASK_ASSIGNMENT =
    "INSERT INTO task_assignments VALUES (:workerId,
:taskId)";

//language=SQL
static final String SQL_INSERT_INTO_COMPLAINTS =
    "INSERT INTO complaints VALUES (:workerId,
:taskId, :description)";

//language=SQL
static final String SQL_INSERT_INTO_COMPLETED =
    "INSERT INTO completed_tasks VALUES (:taskId,
:workerId)";

//language=SQL
static final String SQL_DELETE_FROM_ASSIGNMENTS_BY_IDS =
    "DELETE FROM task_assignments WHERE (worker_id =
:workerId AND task_id = :taskId)";

static final RowMapper<Task> TASK_ROW_MAPPER = new Row-
Mapper<Task>() {
    @Override
    public Task mapRow(ResultSet resultSet, int i) throws
SQLException {
        try {
            return new Task(resultSet.getInt("id"), re-
sultSet.getInt("owner_id"), resultSet.getString
                ("description"), resultSet.get-
Int("attribute_domain_id"), new URL(resultSet.getString
                ("instructions_file_url")));
        } catch (MalformedURLException e) {
            throw new IllegalArgumentException(e);
        }
    }
};

@Override
public List<Task> getAvailableTasks() {
    return sqlQueryExecutor.queryForOb-
jects(SQL_GET_TASKS, TASK_ROW_MAPPER);
}

@Override
public Task getTask(int taskId) {
    verifier.verifyTask(taskId);
    Map<String, Object> paramMap = paramsMapper.as-
Map(asList("taskId"), asList(taskId));
    return sqlQueryExecutor.queryForOb-
ject(SQL_GET_TASK_BY_ID, paramMap, TASK_ROW_MAPPER);
}

```

```

    }

    @Override
    public void taskAssign(int workerId, int taskId) {
        verifier.verifyUncompletedTasksByWorker(workerId);
        verifier.verifyTask(taskId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "taskId"), asList(workerId, taskId));
        sqlQueryExecutor.updateQuery(SQL_INSERT_TASK_ASSIGN-
MENT, paramMap);
    }

    @Override
    public void complaintToTask(int workerId, int taskId,
String description) {
        verifier.verifyAssignment(workerId, taskId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "taskId", "description"),
asList(workerId,
            taskId, description));
        sqlQueryExecutor.updateQuery(SQL_INSERT_INTO_COM-
PLAINTS, paramMap);
    }

    @Override
    public void finishTask(int workerId, int taskId) {
        verifier.verifyAlreadyCompletedTask(workerId,
taskId);
        verifier.verifyAssignment(workerId, taskId);
        Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "taskId"), asList(workerId, taskId));
        sqlQueryExecutor.updateQuery(SQL_INSERT_INTO_COM-
PLETED, paramMap);
        sqlQueryExecutor.updateQuery(SQL_DELETE_FROM_ASSIGN-
MENTS_BY_IDS, paramMap);
    }

}

package ru.kpfu.tools.cct.core.services.dao.utils.jdbc;

import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Component
public class ParamsMapperImpl implements ParamsMapper {

```

```

        public Map<String, Object> asMap(List<String> keys,
List<?> values) {
            if (keys.size() != values.size()) {
                throw new IllegalArgumentException();
            } else {
                Map<String, Object> result = new HashMap<String,
Object>();
                for (int i = 0; i < keys.size(); i++) {
                    result.put(keys.get(i), values.get(i));
                }
                return result;
            }
        }
    }

package ru.kpfu.tools.cct.core.services.dao.utils.jdbc;

import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.NamedParamete-
rJdbcDaoSupport;
import org.springframework.jdbc.core.namedparam.NamedParamete-
rJdbcTemplate;
import org.springframework.stereotype.Component;

import javax.sql.DataSource;
import java.util.List;
import java.util.Map;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Component
public class SqlQueryExecutorImpl implements SqlQueryExecutor
{

    JdbcTemplate jdbcTemplate;
    NamedParameterJdbcTemplate namedParameterJdbcTemplate;

    @Autowired
    public SqlQueryExecutorImpl(DataSource dataSource) {
        NamedParameterJdbcDaoSupport jdbcDaoSupport = new
NamedParameterJdbcDaoSupport();
        jdbcDaoSupport.setDataSource(dataSource);
        this.jdbcTemplate = jdbcDaoSupport.getJdbcTemplate();
        this.namedParameterJdbcTemplate = jdbcDaoSupport.get-
NamedParameterJdbcTemplate();
    }

    public <T> List<T> queryForObjects(String sql, RowMap-
per<T> rowMapper) {
        return jdbcTemplate.query(sql, rowMapper);
    }
}

```

```

        }

        @Override
        public <T> List<T> queryForObjects(String sql,
            Map<String, Object> paramMap, RowMapper<T> rowMapper) {
            return namedParameterJdbcTemplate.query(sql, pa-
            ramMap, rowMapper);
        }

        public <T> T queryForObject(String sql, Map<String, Ob-
            ject> paramMap, RowMapper<T> rowMapper) {
            return namedParameterJdbcTemplate.queryForObject(sql,
            paramMap, rowMapper);
        }

        public void updateQuery(String sql, Map<String, Object>
            paramMap) {
            namedParameterJdbcTemplate.update(sql, paramMap);
        }

        @Override
        public int queryForInt(String sql, Map<String, Object>
            paramMap) {
            return namedParameterJdbcTemplate.queryForInt(sql,
            paramMap);
        }

        @Override
        public int queryForInt(String sql) {
            return jdbcTemplate.queryForInt(sql);
        }
    }

    package ru.kpfu.tools.cct.core.services.dao.utils.verify;

    import org.springframework.beans.factory.annotation.Auto-
    wired;
    import org.springframework.stereotype.Component;
    import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.Params-
    Mapper;
    import ru.kpfu.tools.cct.core.ser-
    vices.dao.utils.jdbc.SqlQueryExecutor;
    import ru.kpfu.tools.cct.core.services.exceptions.*;

    import java.util.Map;
    import static java.util.Arrays.asList;

    /**
     * @author Sidikov Marsel (Kazan Federal University)
     */
    @Component
    public class DaoArgumentsVerifierImpl implements DaoArgu-
    mentsVerifier {

```

```

        //language=SQL
        private static final String SQL_COUNT_TASKS_BY_ID =
            "SELECT COUNT (*) FROM tasks WHERE (id =
:taskId)";

        //language=SQL
        private static final String SQL_COUNT_USERS_BY_ID =
            "SELECT COUNT (*) FROM users WHERE (id =
:userId)";

        //language=SQL
        private static final String SQL_COUNT_OF_ASSIGN-
MENTS_OF_WORKER =
            "SELECT COUNT (*) FROM task_assignments WHERE
(worker_id = :workerId)";

        //language=SQL
        private static final String SQL_COUNT_OF_ASSIGN-
MENTS_BY_IDS =
            "SELECT COUNT (*) FROM task_assignments WHERE
(worker_id = :workerId AND task_id = :taskId)";

        //language=SQL
        private static final String SQL_COUNT_OF_COMPLETED_BY_IDS
=
            "SELECT COUNT (*) FROM completed_tasks WHERE
(worker_id = :workerId AND task_id = :taskId)";

        //language=SQL
        private static final String SQL_COUNT_OF_DOCUMENTS_BY_ID
=
            "SELECT COUNT (*) FROM documents WHERE (id =
:documentId)";

        //language=SQL
        private static final String SQL_COUNT_OF_DOMAINS_BY_ID =
            "SELECT COUNT(*) FROM domains WHERE (id = :do-
mainId)";

        //language=SQL
        private static final String SQL_GET_TASK_ID_OF_DOCUMENT =
            "SELECT (task_id) FROM documents WHERE (id =
:documentId)";

        //language=SQL
        private static final String SQL_COUNT_OF_POSTPONED_BY_IDS
=
            "SELECT COUNT(*) FROM postponed_documents WHERE
(worker_id = :workerId AND document_id = :documentId)";

        @Autowired
        private SqlQueryExecutor sqlQueryExecutor;

```

```

        @Autowired
        private ParamsMapper paramsMapper;

        @Override
        public void verifyUser(int userId) {
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("userId"), asList(userId));
            int usersCount = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_USERS_BY_ID, paramMap);
            if (usersCount != 1) {
                throw new UserNotFoundException(userId);
            }
        }

        @Override
        public void verifyTask(int taskId) {
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("taskId"), asList(taskId));
            int tasksCount = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_TASKS_BY_ID, paramMap);
            if (tasksCount != 1) {
                throw new TaskNotFoundException(taskId);
            }
        }

        @Override
        public void verifyAssignment(int workerId, int taskId) {
            verifyUser(workerId);
            verifyTask(taskId);
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "taskId"), asList(workerId, taskId));
            int assignmentsCount = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_OF_ASSIGNMENTS_BY_IDS, paramMap);
            if (assignmentsCount != 1) {
                throw new TaskNotAssignedToWorkerExcep-
tion(workerId, taskId);
            }
        }

        @Override
        public void verifyUncompletedTasksByWorker(int workerId)
        {
            verifyUser(workerId);
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId"), asList(workerId));
            int assignmentsCount = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_OF_ASSIGNMENTS_OF_WORKER, paramMap);
            if (assignmentsCount >= 1) {
                throw new HasUncompletedTasksException(workerId);
            }
        }
    }

```

```

        @Override
        public void verifyAlreadyCompletedTask(int workerId, int
taskId) {
            verifyUser(workerId);
            verifyTask(taskId);
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "taskId"), asList(workerId, taskId));
            int completedTask = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_OF_COMPLETED_BY_IDS, paramMap);
            if (completedTask != 0) {
                throw new TaskAlreadyCompletedByWorkerExcep-
tion(workerId, taskId);
            }
        }

        @Override
        public void verifyDocument(int documentId) {
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("documentId"), asList(documentId));
            int documentsCount = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_OF_DOCUMENTS_BY_ID, paramMap);
            if (documentsCount != 1) {
                throw new DocumentNotFoundException(documentId);
            }
        }

        @Override
        public void verifyAlreadyPostponedDocument(int workerId,
int documentId) {
            verifyUser(workerId);
            verifyDocument(documentId);
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("workerId", "documentId"), asList(workerId,
documentId));
            int postpones = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_OF_POSTPONES_BY_IDS, paramMap);
            if (postpones != 0) {
                throw new WorkerAlreadyPostponeDocument(workerId,
documentId);
            }
        }

        @Override
        public void verifyDocumentAndWorkerCompatibility(int
workerId, int documentId) {
            verifyUser(workerId);
            verifyDocument(documentId);
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("documentId"), asList(documentId));
            int taskId = sqlQueryExecutor.queryFor-
Int(SQL_GET_TASK_ID_OF_DOCUMENT, paramMap);
            verifyAssignment(workerId, taskId);
        }

```



```

        @Override
        public void verifyDomain(int domainId) {
            Map<String, Object> paramMap = paramsMapper.as-
Map(asList("domainId"), asList(domainId));
            int domainsCount = sqlQueryExecutor.queryFor-
Int(SQL_COUNT_OF_DOMAINS_BY_ID, paramMap);
            if (domainsCount != 1) {
                throw new DomainNotFoundException(domainId);
            }
        }
    }
}

package ru.kpfu.tools.cct.core.services;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Configuration
@ComponentScan("ru.kpfu.tools.cct.core.services.dao")
public class ApplicationContext {

}

package ru.kpfu.tools.cct.core.services;

import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;
import org.springframework.stereotype.Service;
import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.Params-
Mapper;
import ru.kpfu.tools.cct.core.ser-
vices.dao.utils.jdbc.SqlQueryExecutor;
import ru.kpfu.tools.cct.core.services.dao.utils.ver-
ify.DaoArgumentsVerifier;
import ru.kpfu.tools.cct.core.services.models.User;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Map;

import static java.util.Arrays.asList;

@Repository
public class UsersServiceImpl implements UsersService {

    @Autowired
    DaoArgumentsVerifier verifier;

```

```

@Autowired
ParamsMapper paramsMapper;

@Autowired
SqlQueryExecutor sqlQueryExecutor;

//language=SQL
private static final String SQL_GET_USER_BY_LOGIN =
    "SELECT * FROM users WHERE login =:login";

//language=SQL
private static final String SQL_GET_USER_BY_TOKEN =
    "SELECT * FROM users WHERE token =:token";

//language=SQL
private static final String SQL_UPDATE_TOKEN =
    "UPDATE users SET token = :token WHERE id = :id";

//language=SQL
private static final String SQL_USER_INSERT =
    "INSERT INTO users (organization, role, rating,
login, password_hash, token) " +
    "VALUES (:organization, :role, :rating,
:login, :passwordHash, :token)";

private final RowMapper<User> USER_ROW_MAPPER = new Row-
Mapper<User>() {
    @Override
    public User mapRow(ResultSet resultSet, int i) throws
SQLException {
        return new User(
            resultSet.getInt("id"),
            resultSet.getString("organization"),
            resultSet.getString("role"),
            resultSet.getInt("rating"),
            resultSet.getString("login"),
            resultSet.getString("password_hash"),
            resultSet.getString("token"));
    }
};

@Override
public User getUser(String login) {
    Map<String, Object> paramMap = paramsMapper.as-
Map(asList("login"), asList(login));
    return sqlQueryExecutor.queryForOb-
ject(SQL_GET_USER_BY_LOGIN, paramMap, USER_ROW_MAPPER);
}

@Override
public User getUserByToken(String token) {

```

```

        Map<String, Object> paramMap = paramsMapper.asMap(
            Map(asList("token"), asList(token)));
        return sqlQueryExecutor.queryForObject(
            SQL_GET_USER_BY_TOKEN, paramMap, USER_ROW_MAPPER);
    }

    @Override
    public void storeTokenForUser(User user, String token) {
        Map<String, Object> paramMap = paramsMapper.asMap(
            Map(asList("id", "token"), asList(user.getId(), token)));
        sqlQueryExecutor.updateQuery(SQL_UPDATE_TOKEN, paramMap);
    }

    @Override
    public void addUser(User user) {
        Map<String, Object> paramMap = paramsMapper.asMap(
            Map(
                asList("organization", "role", "rating",
                    "login", "passwordHash", "token"),
                asList(user.getOrganization(),
                    user.getRole(), 0, user.getLogin(), user.getPasswordHash(), user
                        .getToken())));
        sqlQueryExecutor.updateQuery(SQL_USER_INSERT, paramMap);
    }
}

package ru.kpfu.tools.cct.core.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ru.kpfu.tools.cct.core.services.dao.domains.DomainsDao;
import ru.kpfu.tools.cct.core.services.dao.recalls.RecallsDao;
import ru.kpfu.tools.cct.core.services.dao.tasks.TasksDao;
import ru.kpfu.tools.cct.core.services.models.*;
import ru.kpfu.tools.cct.core.services.dao.documents.DocumentsDao;
import ru.kpfu.tools.cct.core.services.dao.PrecedentsDao;

import java.util.List;

/**
 * @author Sidikov Marsel (Kazan Federal University)
 */
@Service
public class WorkerServiceFacadeImpl implements WorkerServiceFacade {

    @Autowired
    TasksDao tasksDao;

```

```

@Autowired
DocumentsDao documentsDao;

@Autowired
PrecedentsDao precedentsDao;

@Autowired
DomainsDao domainsDao;

@Autowired
RecallsDao recallsDao;

@Override
public List<Task> getAvailableTasks() {
    return tasksDao.getAvailableTasks();
}

@Override
public List<Recall> getRecalls(int userId) {
    return recallsDao.getRecalls(userId);
}

@Override
public Task getTask(int taskId) {
    return tasksDao.getTask(taskId);
}

@Override
public void taskAssign(int workerId, int taskId) {
    tasksDao.taskAssign(workerId, taskId);
}

@Override
public Domain getDomain(int domainId) {
    return domainsDao.getDomain(domainId);
}

@Override
public List<Document> getDocuments(int taskId) {
    return documentsDao.getDocuments(taskId);
}

@Override
public Document getDocument(int documentId) {
    return documentsDao.getDocument(documentId);
}

@Override
public void complaintToTask(int workerId, int taskId,
String description) {
    tasksDao.complaintToTask(workerId, taskId, descrip-
tion);
}

```

```

        @Override
        public void postponeDocument(int workerId, int document-
tId) {
            documentsDao.postponeDocument(workerId, documentId);
        }

        @Override
        public void sendPrecedent(int workerId, int taskId, Prece-
dent precedent) {
            precedentsDao.sendPrecedent(workerId, taskId, prece-
dent);
        }

        @Override
        public List<Document> getPostponedDocuments(int workerId,
int taskId) {
            return documentsDao.getPostponedDocuments(workerId,
taskId);
        }

        @Override
        public void finishTask(int workerId, int taskId) {
            tasksDao.finishTask(workerId, taskId);
        }
    }
}

package ru.kpfu.tools.cct.core.services.dao.tasks;

import com.google.common.collect.Lists;
import org.springframework.test.con-
text.junit4.SpringJUnit4ClassRunner;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Auto-
wired;
import org.springframework.test.context.ContextConfiguration;

import ru.kpfu.tools.cct.core.services.AppContext;
import ru.kpfu.tools.cct.core.services.dao.Integration-
TestAppContext;
import ru.kpfu.tools.cct.core.services.exceptions.*;
import ru.kpfu.tools.cct.core.services.models.Task;

import static org.junit.Assert.assertThat;
import static org.hamcrest.CoreMatchers.*;

import java.util.List;

import static org.junit.Assert.assertEquals;
import static ru.kpfu.tools.cct.core.ser-
vices.dao.tasks.TasksDaoData.*;

```

```

    @RunWith(SpringJUnit4ClassRunner.class)
    @ContextConfiguration(classes = {ApplicationContext.class, IntegrationTestApplicationContext.class})
    public class TasksDaoIntegrationImplTest {
        /*
            @Autowired
            private TasksDaoJdbcImpl tasksDao;

            @Test
            public void testGetAvailableTasks() {
                List<Task> actual = tasksDao.getAvailableTasks();
                List<Task> expected = Lists.newArrayList(TASK_1,
TASK_2);
                assertThat(actual, is(expected));
            }

            @Test
            public void testGetTask() throws Exception {
                Task actual = tasksDao.getTask(0);
                Task expected = TASK_1;
                assertEquals(actual, expected);
            }

            @Test (expected = TaskNotFoundException.class)
            public void testGetTaskForIncorrectTaskId() throws Exception {
                tasksDao.getTask(2);
            }

            @Test
            public void testTaskAssign() {
                tasksDao.taskAssign(0, 1);
                int actual = tasksDao.sqlQueryExecutor.queryForInt(
                    "SELECT COUNT(*) FROM task_assignments WHERE
(user_id = 0 AND task_id = 1)");
                assertThat(actual, is(1));
            }

            @Test (expected = UserNotFoundException.class)
            public void testTaskAssignForIncorrectUserId() {
                tasksDao.taskAssign(4, 1);
            }

            @Test (expected = TaskNotFoundException.class)
            public void testTaskAssignForIncorrectTaskId() {
                tasksDao.taskAssign(0, 10);
            }

            @Test (expected = HasUncompletedTasksException.class)
            public void testTaskAssignForIncorrectAssignment() {
                tasksDao.taskAssign(0, 1);
                tasksDao.taskAssign(0, 1);
            }
        */
    }

```

```

    }

    @Test
    public void testComplaintToTask() {
        tasksDao.complaintToTask(1,1,"bad task");
        int actual = tasksDao.sqlQueryExecutor.queryForInt(
            "SELECT COUNT(*) FROM complaints WHERE
(user_id = 1 AND task_id = 1 AND description = 'bad task')");

        assertThat(actual, is(1));
    }

    @Test(expected = TaskNotFoundException.class)
    public void testComplaintToTaskForIncorrectTaskId() {
        tasksDao.complaintToTask(1,10, "bad task");
    }

    @Test (expected = UserNotFoundException.class)
    public void testComplaintToTaskForIncorrectUserId() {
        tasksDao.complaintToTask(10, 1, "bad task");
    }

    @Test (expected = TaskNotAssignedToUserException.class)
    public void testComplaintToTaskForIncorrectAssignment() {
        tasksDao.complaintToTask(0, 1, "bad task");
    }

    @Test
    public void testFinishTask() {
        tasksDao.finishTask(1,1);
        int actual = tasksDao.sqlQueryExecutor.queryForInt(
            "SELECT COUNT (*) FROM completed_tasks WHERE
(user_id = 1 AND task_id = 1)");
        assertThat(actual, is(1));
    }

    @Test (expected = UserNotFoundException.class)
    public void testFinishTaskForIncorrectUserId() {
        tasksDao.finishTask(10,1);
    }

    @Test (expected = TaskNotFoundException.class)
    public void testFinishTaskForIncorrectTaskId() {
        tasksDao.finishTask(1,10);
    }

    @Test (expected = TaskNotAssignedToUserException.class)
    public void testFinishTaskForIncorrectAssignments() {
        tasksDao.finishTask(0,1);
    }

    @Test (expected = TaskAlreadyCompletedByUserException.class)

```

```

        public void testFinishTaskSecond() {
            tasksDao.finishTask(1,1);
            tasksDao.finishTask(1,1);
        }
    */
}
package ru.kpfu.tools.cct.core.services.dao.tasks;

import org.junit.Before;
import org.junit.Test;
import org.mockito.Mock;
import ru.kpfu.tools.cct.core.services.dao.utils.verify.DaoArgumentsVerifier;
import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.ParamsMapper;
import ru.kpfu.tools.cct.core.services.dao.utils.jdbc.SqlQueryExecutor;
import ru.kpfu.tools.cct.core.services.exceptions.TaskNotAssignedToWorkerException;
import ru.kpfu.tools.cct.core.services.exceptions.TaskNotFoundException;
import ru.kpfu.tools.cct.core.services.exceptions.UserNotFoundException;
import ru.kpfu.tools.cct.core.services.models.Task;

import java.util.List;

import static java.util.Arrays.asList;
import static org.hamcrest.core.Is.is;
import static org.junit.Assert.*;
import static org.mockito.Mockito.*;
import static org.mockito.MockitoAnnotations.initMocks;
import static ru.kpfu.tools.cct.core.services.dao.tasks.TasksDaoJdbcImpl.*;
import static ru.kpfu.tools.cct.core.services.dao.tasks.TasksDaoData.*;

public class TasksDaoJdbcImplTest {

    private TasksDaoJdbcImpl tasksDao;

    @Mock
    DaoArgumentsVerifier verifier;

    @Mock
    SqlQueryExecutor sqlQueryExecutor;

    @Mock
    ParamsMapper paramsMapper;

    @Before
    public void setUp() throws Exception {
        initMocks(this);

```



```

        stubbing();
        buildTaskDao();
    }

    private void stubbing() {
        stubbingSqlQueryExecutor();
        stubbingParamsMapper();
        stubbingVerifier();
    }

    private void stubbingSqlQueryExecutor() {
        doReturn(TASKS).when(sqlQueryExecutor).queryForObject(
            SQL_GET_TASKS, TASK_ROW_MAPPER);
        doReturn(TASK_1).when(sqlQueryExecutor).queryForObject(
            SQL_GET_TASK_BY_ID, PARAMS_TASK, TASK_ROW_MAPPER);
    }

    private void stubbingParamsMapper() {
        doReturn(PARAMS_TASK).when(paramsMapper).asMap(
            asList("taskId"), asList(TASK_1.getId()));
        doReturn(PARAMS_TASK_AND_USER).when(paramsMapper).asMap(
            asList("userId", "taskId"), asList(USER_1_ID,
            TASK_1.getId()));
    }

    private void stubbingVerifier() {
        doThrow(TaskNotFoundException.class).when(verifier).verifyTask(
            anyInt());
        doThrow(TaskNotAssignedToWorkerException.class).when(verifier).verifyAssignment(
            anyInt(), anyInt());
        doThrow(UserNotFoundException.class).when(verifier).verifyUncompletedTasksByWorker(
            anyInt());
        doThrow(TaskNotAssignedToWorkerException.class).when(verifier).verifyUncompletedTasksByWorker(
            USER_2_ID);
        doNothing().when(verifier).verifyTask(TASK_1.getId());
        doNothing().when(verifier).verifyUncompletedTasksByWorker(USER_1_ID);
    }

    private void buildTaskDao() {
        tasksDao = TasksDaoJdbcImpl.build(verifier, paramsMapper, sqlQueryExecutor);
    }

    @Test
    public void testGetAvailableTasks() throws Exception {
        List<Task> actual = tasksDao.getAvailableTasks();
        List<Task> expected = TASKS;

        assertThat(actual, is(expected));
    }

```

```

@Test
public void testGetTask() throws Exception {
    Task actual = tasksDao.getTask(TASK_1.getId());
    Task expected = TASK_1;

    assertEquals(expected, actual);
}

@Test (expected = TaskNotFoundException.class)
public void testGetTaskForIncorrectId() throws Exception
{
    tasksDao.getTask(2);
}

@Test
public void testTaskAssign() {
    tasksDao.taskAssign(USER_1_ID, TASK_1.getId());

    verify(verifier).verifyUncompletedTasksBy-
Worker(USER_1_ID);
    verify(verifier).verifyTask(TASK_1.getId());
}

@Test (expected = UserNotFoundException.class)
public void testTaskAssignForIncorrectUserId() {
    tasksDao.taskAssign(2, TASK_1.getId());
}

@Test (expected = TaskNotAssignedToWorkerException.class)
public void testTaskAssignForUnassignedUser() {
    tasksDao.taskAssign(USER_2_ID, TASK_1.getId());
}

@Test (expected = TaskNotFoundException.class)
public void testTaskAssignForIncorrectTask() {
    tasksDao.taskAssign(USER_1_ID, 3);
}
}

CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    organization VARCHAR(100),
    role VARCHAR (100),
    rating INTEGER,
    login VARCHAR(100),
    password_hash VARCHAR(100),
    token VARCHAR(500)
);

CREATE TABLE domains (
    id SERIAL PRIMARY KEY,
    domain_values VARCHAR(500)

```

```

);

CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    owner_id INTEGER,
    description VARCHAR(500),
    attribute_domain_id INTEGER,
    instructions_file_url VARCHAR(100),

    FOREIGN KEY (owner_id) REFERENCES users(id),
    FOREIGN KEY (attribute_domain_id) REFERENCES domains(id)
);

CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    task_id INTEGER,
    file_name VARCHAR(100),
    type VARCHAR(100),
    folder_name VARCHAR(200),
    size INTEGER,
    url VARCHAR(100),

    FOREIGN KEY (task_id) REFERENCES tasks(id)
);

CREATE TABLE precedents (
    worker_id INTEGER,
    task_id INTEGER,
    document_id INTEGER,
    fragment VARCHAR(500),
    label_value VARCHAR(100),

    FOREIGN KEY (worker_id) REFERENCES users(id),
    FOREIGN KEY (task_id) REFERENCES tasks(id),
    FOREIGN KEY (document_id) REFERENCES documents(id)
);

CREATE TABLE task_assignments (
    worker_id INTEGER,
    task_id INTEGER,

    FOREIGN KEY (worker_id) REFERENCES users(id),
    FOREIGN KEY (task_id) REFERENCES tasks(id)
);

CREATE TABLE complaints (
    worker_id INTEGER,
    task_id INTEGER,
    description VARCHAR(500),

    FOREIGN KEY (worker_id) REFERENCES users(id),
    FOREIGN KEY (task_id) REFERENCES tasks(id)
);

```

```

CREATE TABLE completed_tasks (
    task_id INTEGER,
    worker_id INTEGER,

    FOREIGN KEY (task_id) REFERENCES tasks(id),
    FOREIGN KEY (worker_id) REFERENCES users(id)
);

CREATE TABLE accepted_decisions (
    worker_id INTEGER,
    task_id INTEGER,

    FOREIGN KEY (worker_id) REFERENCES users(id),
    FOREIGN KEY (task_id) REFERENCES tasks(id)
);

CREATE TABLE recalls (
    user_id INTEGER,
    author_id INTEGER,
    mark INTEGER,
    recall_text VARCHAR(500),

    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (author_id) REFERENCES users(id)
);

CREATE TABLE banned_users (
    user_id INTEGER,
    reason VARCHAR(500),

    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE postponed_documents (
    worker_id INTEGER,
    document_id INTEGER,

    FOREIGN KEY (worker_id) REFERENCES users(id),
    FOREIGN KEY (document_id) REFERENCES documents(id)
);

INSERT INTO users (organization, role, rating, login, password_hash, token)
VALUES ('KPFU', 'WORKER', 90, 'sidikov.marsel@gmail.com', 'qwerty007', 'sqwerty007');

INSERT INTO users (organization, role, rating, login, password_hash, token)
VALUES ('KPFU', 'WORKER', 100, 'irinoise@gmail.com', 'irish-ka', 'iirishka');

```

```

INSERT INTO users (organization, role, rating, login, password_hash, token)
VALUES ('KPFU', 'TASK_OWNER', 100, 'ivanov@gmail.com', 'vladivan', 'ivanvladivan');

INSERT INTO domains (domain_values)
VALUES ('животное, человек');

INSERT INTO domains (domain_values)
VALUES ('существительное');

INSERT INTO tasks (owner_id, description, attribute_domain_id, instructions_file_url)
VALUES (3, 'Определить местонахождение человека и животного на фотографии', 1,
        'https://bitbucket.org/marsel_sidikov/cctool/wiki/');

INSERT INTO tasks (owner_id, description, attribute_domain_id, instructions_file_url)
VALUES (3, 'Найти в документах существительные', 2,
        'https://bitbucket.org/marsel_sidikov/cctool/wiki/');

INSERT INTO documents (task_id, file_name, type, folder_name, size, url)
VALUES (1, 'imag1.jpg', 'imag', 'task_1_images', 0,
        'http://img.izismile.com/img/img5/20120314/640/animals_and_humans_being_too_cute_46tAq_640_06.jpg');

INSERT INTO documents (task_id, file_name, type, folder_name, size, url)
VALUES (1, 'imag2.jpg', 'imag', 'task_1_images', 0,
        'http://img.izismile.com/img/img5/20120314/640/animals_and_humans_being_too_cute_jVsCq_640_14.jpg');

INSERT INTO documents (task_id, file_name, type, folder_name, size, url)
VALUES (1, 'imag3.jpg', 'imag', 'task_1_images', 0,
        'http://photoity.com/wp-content/uploads/2014/04/Real-Animals-Pose-with-Humans-3.jpg');

INSERT INTO recalls (user_id, author_id, mark, recall_text)
VALUES (3, 1, 10, 'Интересные задачи, веселье - гарантирую');

INSERT INTO recalls (user_id, author_id, mark, recall_text)
VALUES (3, 2, 15, 'Спасибо за предоставленную возможность поработать!');

```