

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования «Ярославский государственный университет им.  
П.Г. Демидова»

Кафедра информационных и сетевых технологий

Сдано на кафедру

«\_\_\_\_\_» \_\_\_\_\_ 2020 г.

Заведующий кафедрой,  
к. ф.-м. н.

\_\_\_\_\_ Д. Ю. Чалый

Выпускная квалификационная работа

**Разработка клиентской части системы для  
автоматизации процесса рекрутирования  
сотрудников**

по направлению  
09.03.03 Прикладная информатика

Научный руководитель  
стар. преподаватель

\_\_\_\_\_ Н. В. Легков

«\_\_\_\_\_» \_\_\_\_\_ 2020 г.

Студент группы ПИЭ-41БО

\_\_\_\_\_ О. С. Гаршина

«\_\_\_\_\_» \_\_\_\_\_ 2020 г.

Ярославль, 2020

## Реферат

Объем 39 с., 4 гл., 23 рис., 0 табл., 8 источников, 0 прил.

Ключевые слова и выражения: **react, рекрутер, автоматизация, HR, резюме, front-end, JavaScript**

Целью данной работы является разработка клиентской части приложения - HR-CV Portal, который оптимизирует работу рекрутеров при создании резюме.

В работе проведён анализ потребностей клиента. Также сформированы требования к приложению, определены технологии для разработки, отвечающие поставленным требованиям. В результате работы был получен опыт в сфере front-end разработки, а конечный продукт передан клиенту для использования и получил положительные отзывы и обратную связь для наращивания функционала в дальнейшем.

# Содержание

<b>Введение</b>	<b>5</b>
<b>1. Теоретические принципы для создания одностраничного реактивно-го приложения</b>	<b>6</b>
1.1. Одностраничное приложение . . . . .	6
1.2. Разработка с JSX . . . . .	7
1.3. Компонента React . . . . .	7
1.3.1. React Props . . . . .	7
1.3.2. React State . . . . .	8
1.3.3. Жизненный цикл в React . . . . .	8
1.3.4. События в React . . . . .	10
1.3.5. Обработка форм в React . . . . .	11
1.4. Material-UI для React . . . . .	11
1.5. Стилизация с LESS . . . . .	12
1.6. Axios для более приятной работы с запросами . . . . .	12
1.7. Управление состоянием с помощью MobX . . . . .	12
1.8. Авторизация с помощью JWT-токенов . . . . .	13
1.8.1. Авторизация . . . . .	14
1.8.2. Обмен информацией . . . . .	14
1.8.3. Структура JSON Web Token . . . . .	14
1.8.4. Access Token и Refresh Token . . . . .	15
<b>2. О задаче</b>	<b>17</b>
2.1. Постановка задачи . . . . .	17
2.2. Требуемый функционал . . . . .	17
2.3. Используемые программные средства . . . . .	18
<b>3. Решение задачи</b>	<b>19</b>
3.1. Создание базовой архитектуры . . . . .	19
3.2. Создание сервиса запросов, авторизация . . . . .	20
3.3. Регистрация, валидация и вход на сервис . . . . .	21
3.4. Сервис уведомлений пользователя, отслеживание прогресса загрузки . . . . .	23
3.5. Страница создания резюме взглядом пользователя . . . . .	25
3.6. Таблица всех резюме взглядом пользователя . . . . .	31
3.7. Интерфейс резюме взглядом администратора . . . . .	31
<b>4. Результаты решения задачи</b>	<b>37</b>

<b>Заключение</b>	<b>38</b>
<b>Список литературы</b>	<b>39</b>

## Введение

Несмотря на то, что мы живем в век технологий и автоматизации есть еще много аспектов, требующих алгоритмов, которые не сможет имитировать машина. Такие вещи обычно требуют психологических навыков, индивидуального подхода и нажитого опыта.

В дипломной работе рассматриваются проблемы рекрутеров компании Akvelon при бюрократической деятельности, а именно проблемы при работе с огромным количеством резюме, которые надо создавать с нуля, редактировать и поддерживать в актуальном состоянии.

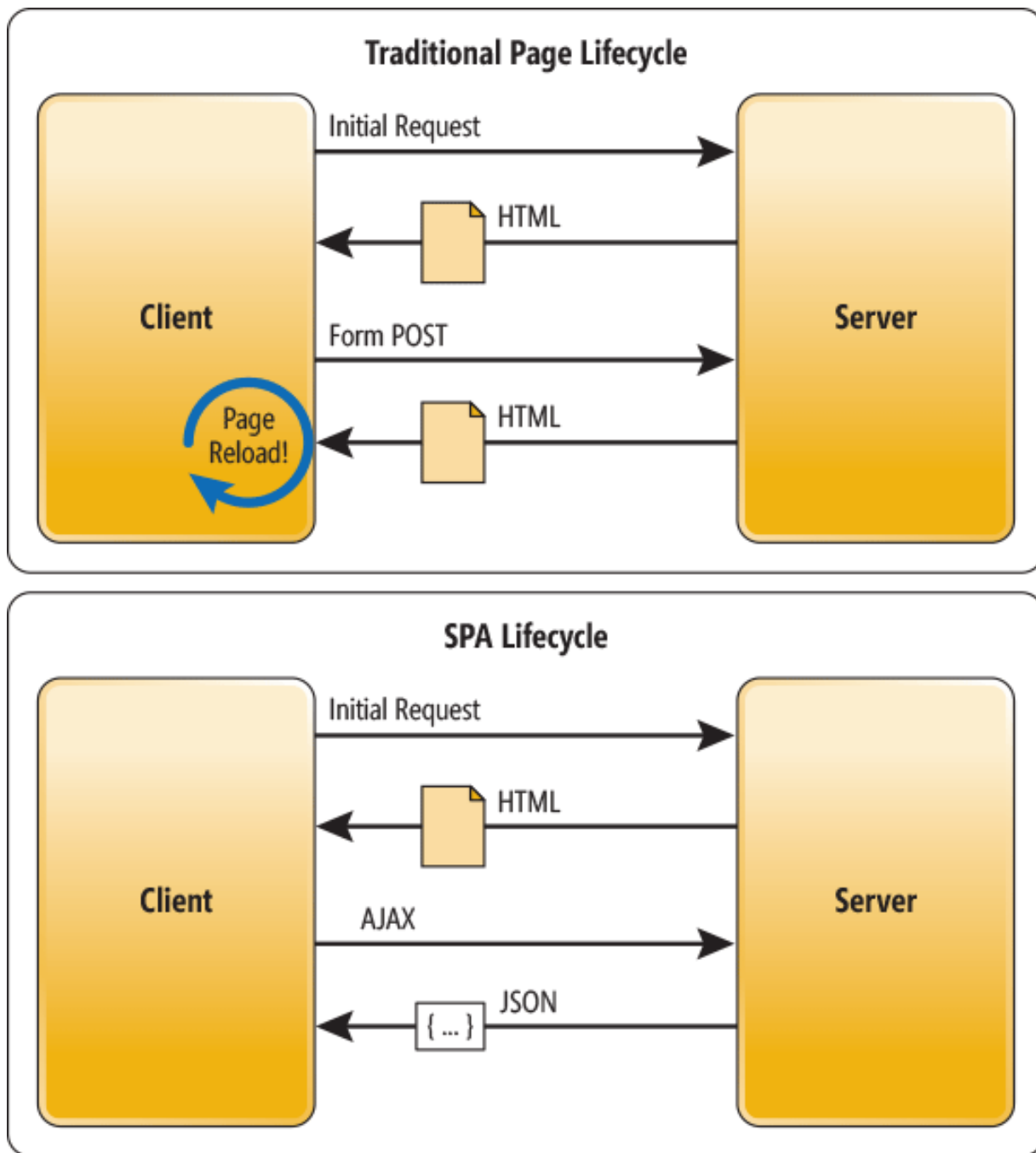
Заполнение резюме формата компании Akvelon может занимать у сотрудника от часа до четырех часов. Как показал опрос клиента, наибольшей проблемой является время, потраченное на копирование информации из одного места в другое, орфографические ошибки кандидатов, правки съехавшей разметки в word-документе.

В связи с этим, было поставлена задача создать web-приложение, которое бы являлось централизованным хранилищем всех резюме компании и цель которого — сделать процесс заполнения данного документа менее рутинным и медленным.

# 1. Теоретические принципы для создания одностраничного реактивного приложения

## 1.1. Одностраничное приложение

Single Page Application – одностраничное приложение – что это такое?



**Рис. 1** — Разница в жизненных циклах двух концепций

Для ответа на этот вопрос вспомним, что раньше традиционный жизненный

цикл страницы работал следующим образом (рис. 1): сервер генерирует много html кода и возвращает его в браузер. Браузер (он же клиент), отображает его. Потом, если мы ходим увидеть изменения на интерфейсе, мы обновляем страницу целиком, а сервер возвращает полностью практически идентичный предыдущему html код с небольшими различиями.

Одностраничные приложения концептуально отличается тем, оно загружает страничку html единожды. С сервера приходит мало html, но много java-script кода, который и генерирует разметку и посылает AJAX запросы с клиента на сервер, на что сервер возвращает данные в формате JSON.

При этом, чтобы узнать, не поменялось ли что-либо на интерфейсе не нужно перезагружать страницу. Может существовать какой-нибудь отдельно взятый блок с java-script, который будет периодически отправлять запросы на сервер, чтобы узнать, не появилось ли новых данных для отображения на клиенте. Если они есть, то возвращаются данные и рендерится маленький кусочек разметки, отвечающий за это, за счет чего сильно экономится трафик. React[1], Angular и Vue работают именно по принципу Single Page Application, динамически отрисовывая интерфейс приложений с помощью изменения состояния отдельно взятых компонентов.

## 1.2. Разработка с JSX

JSX[2], созданный разработчиками Facebook, фактически является синтаксическим сахаром JavaScript-а.

JSX позволяет нам писать html-элементы в JavaScript и помещать их в DOM без каких-либо createElement() и / или appendChild() методов. Можно не использовать JSX, но JSX облегчает написание приложений React.

1. `<h1>Я-JSX разметка!</h1>`
2. `React.createElement('h1', , 'Я не JSX разметка!');`

Оба варианта отобразят одинаковый интерфейс, но код, написанный с JSX выглядит гораздо понятнее и структурированнее. С ним вы можете писать выражения внутри фигурных скобок, которое может быть переменной React, свойством или любым другим допустимым выражением JavaScript.

Html-код должен быть обязательно заключен в один элемент верхнего уровня. Поэтому, если необходимо вернуть два тега, разработчик должен поместить их в родительский элемент, например div или React.Fragment.

## 1.3. Компонента React

### 1.3.1. React Props

Компонента – это независимая и многократно используемая JSX-разметка. Компоненты служат той же цели, что и функции JavaScript, но работают изолиро-

ванно и возвращают html через функцию рендеринга. Они принимают произвольные входные данные (так называемые «пропсы») и возвращают React-элементы, описывающие, что мы хотим увидеть на экране. Компоненты бывают двух типов: компоненты класса и компоненты функции, автор в своей работе использовала только первый вариант.

Компоненты могут ссылаться на другие компоненты в возвращённой ими разметке. Это позволяет нам использовать одну и ту же абстракцию — компоненты — на любом уровне нашего приложения. Неважно, разрабатываем ли мы особый заголовок, поле для заполнения или большую страницу: все они представляют собой компоненты в React-приложениях.

Props - это аргументы, передаваемые в компоненты React. Props передаются компонентам через атрибуты html, используется тот же самый синтаксис. Каждый компонент получает аргумент в виде props-объекта, а также позволяет передавать данные из одного компонента в другой в качестве параметров. React Props доступны только для чтения. Программист получит ошибку, если попытается изменить их значение.

### **1.3.2. React State**

У компонентов React есть встроенный state-объект. В этом state-объекте хранятся значения свойств, которые принадлежат компоненту. Когда он изменяется, компонент перерисовывается (перерендеривается).

State-Объект инициализируется в конструкторе (рис. 2):

Обратиться к state-объекту можно в любом месте компонента, используя синтаксис `this.state.<имя-состояния>`. Чтобы изменить значение в объекте состояния, используется `this.setState()` метод. Когда значение в state-объекте изменяется, компонент будет повторно перерендериваться.

### **1.3.3. Жизненный цикл в React**

Каждый компонент в React имеет жизненный цикл, который можно отслеживать и манипулировать им на трех основных этапах:

- Монтирование
- Обновление
- Размонтирование

Монтирование означает помещение react-элементов в DOM-дерево. В React есть четыре встроенных метода, которые вызываются в следующем порядке при монтировании компонента:

- `constructor()`
- `getDerivedStateFromProps()`



```

class Car extends React.Component {
  constructor(props) {
    super(props);
    this.state = {brand: "Ford"};
  }
  render() {
    return (
      <div>
        <h1>My Car</h1>
      </div>
    );
  }
}

```

**Рис. 2** — Инициализация состояния

- render()
- componentDidMount()

Метод render() является обязательным и будет вызываться всегда, остальные являются необязательными и будут вызываться, если их работу определить самостоятельно.

Метод constructor() вызывается раньше всего, когда компонент инициализируется, и это же является местом для установки начального состояния и других начальных значений. Данный метод вызывается с props в качестве аргументов, и вы всегда должны начинаться с вызова функции super(props), так как это запускает метод конструктора его родителя и позволяет компоненту наследовать методы от React.Component.

Метод getDerivedStateFromProps() вызывается непосредственно перед отображением элемента(ов) в DOM-дереве. Обычно в данном методе устанавливают значения состояния на основе props-ов. Он принимает state в качестве аргумента и возвращает объект уже с изменённым state.

Метод componentDidMount() вызывается после первоначального рендера компонента. Здесь можно запустить код, который требует, чтобы компонент уже

был помещен в DOM-дерево.

Следующий этап жизненного цикла — это когда компонент обновляется. Это происходит всякий раз, когда происходит изменение состояния или props-а компонента.

React имеет пять встроенных методов, которые вызываются в следующем порядке при обновлении компонента:

- `getDerivedStateFromProps()`
- `shouldComponentUpdate()`
- `render()`
- `getSnapshotBeforeUpdate()`
- `componentDidUpdate()`

Метод `getDerivedStateFromProps` также срабатывает и при обновлении. Это первый метод, который при этой вызывается. Выполняет те же функции, что и при монтировании.

В методе `shouldComponentUpdate()` можно вернуть логическое значение, которое указывает, должен ли React продолжать процесс рендеринга или нет. Значение по умолчанию равняется `true`.

Логично, что `render()` вызывается при обновлении компонента, ведь он должен повторно визуализировать html-код в DOM-дерево с новыми изменениями.

В методе `getSnapshotBeforeUpdate()` есть доступ к props-ам и состоянию до обновления. Это означает, что даже после обновления можно проверить, какие значения были до него.

`componentDidUpdate()` вызывается после обновления компонента в DOM-дерево. Если пытаться менять состояние в данном методе, то можно получить в результате бесконечный цикл, так как сразу после изменения `state` произойдет обновление компонента и вновь вызовется `componentDidUpdate()`.

Следующий этап жизненного цикла — это когда компонент удаляется из DOM или размонтируется, как это принято называть в React.

В React есть только один встроенный метод, который вызывается при размонтировании компонента: `componentWillUnmount()`

#### **1.3.4. События в React**

Как и HTML, React может выполнять действия на основе пользовательских событий. React имеет те же события, что и HTML: щелчок, изменение, наведение мыши и т.д.

События React записываются в синтаксисе camelCase: `onClick` вместо `onclick`. Обработчики событий React написаны в фигурных скобках: `onClick=show` вместо `onClick="show()"`. Хорошей практикой является размещение обработчика событий как метод в классе компонента.

Для методов в React ключевое слово "this" должно представлять компонент, которому принадлежит метод. Вот почему стоит использовать стрелочные функции. С ними this всегда будет иметь контекст объекта, который определил стрелочную функцию.

### 1.3.5. Обработка форм в React

Как и в HTML, React использует формы, позволяющие пользователям взаимодействовать с web-страницей. Обработка форм — это то, как обрабатываются данные, когда они изменяют значение или передаются. В html данные формы обычно обрабатываются DOM-деревом. В React они обрабатываются компонентами.

Когда данные обрабатываются компонентами, все данные сохраняются в их состоянии. Тем самым можно контролировать изменение состояния, добавив обработчики событий в onChange атрибут.

Если разработчик не хочет отображать элемент (например button) до тех пор, пока пользователь не сделает какой-либо ввод, он сможет добавить оператор if прямо в метод рендера. Контролировать действие отправки можно, добавив обработчик события в атрибуте onSubmit.

Очень часто приходится управлять значениями более чем одного поля ввода. Для этого можно добавить name-атрибут к каждому элементу. Во время инициализации состояний в конструкторе просто нужно использовать имена полей для ввода. Чтобы получить доступ к этим полям в методе обработчика используется синтаксис event.target.name и event.target.value. Обновление состояния обычно делают по принципу: this.setState({[event.target.name] : event.target.value}).

## 1.4. Material-UI для React

Material UI[3] это библиотека на основе Material Design - вида дизайна, разработанного в 2014 году Google и очень популярного для web и мобильных приложений. Он вдохновлен физическим миром и его текстурами, включая то, как они отражают свет и отбрасывают тени. Минималистичен, не перегружен деталями.

Данная библиотека позволяет не тратить часы на стилизацию CSS с нуля, а позволяет пользоваться готовыми компонентами. Чтобы начать, нужно иметь знания в React и разобраться в документации на официальном сайте, где описано, как подключить все зависимости, настроить конфигурацию особого файла muitheme.js.

Разработчики хоть и обновляют библиотеку достаточно часто, но их компоненты все равно не исчерпывают все потребности в разработке интерфейса, но являются отличной базой для собственных.

## 1.5. Стилизация с LESS

LESS[4] - язык динамических таблиц стилей, а также препроцессор CSS. LESS расширяет CSS динамическим поведением, таким как переменные, миксины, операции и функции. LESS работает как на стороне клиента (IE 6+, Webkit, Firefox), так и на стороне сервера, с Node.js.

Проще говоря, LESS позволяет писать CSS более умным способом, комбинируя функции, миксины, операции и многое другое. Это означает, что вы пишете более краткую информацию о стилях и можете более легко использовать такие вещи, как цвета и стили. Препроцессор CSS - это язык сценариев, который является расширением CSS. Он компилируется в обычный синтаксис CSS, а затем CSS читается web-браузером. Less очень похож на CSS.

Миксины позволяют встраивать все свойства класса в другой класс, включая имя класса в качестве одного из его свойств, и таким образом это работает в своем роде как константа или переменная. Они также могут вести себя как функции и принимать аргументы.

CSS не поддерживает миксины, поэтому файлы формата .css содержат так много повторяющегося кода. Миксины обеспечивают более эффективный и чистый код, а также облегчают его изменение. LESS позволяет использовать операции и функции. Операции позволяют добавлять, вычитать, делить и умножать значения и цвета свойств, которые можно использовать для создания сложных отношений между свойствами, а функции позволяют манипулировать разными значениями прямо внутри LESS файла.

## 1.6. Axios для более приятной работы с запросами

Axios[?] является одним из самых популярных HTTP-клиентов на основе промисов как для браузеров, так и для Node.js.

По умолчанию он защищает от подделки межсайтовых запросов (XSRF).

При работе с Axios у нас есть прямой доступ к JSON-результату в свойстве data объекта ответа. Также данная библиотека выдаёт сведения о сетевых ошибках и умеет следить за ходом загрузки данных. Это может быть полезным тем, кто разрабатывает приложение, позволяющее пользователям загружать на сервер фотографии или видеофайлы.

## 1.7. Управление состоянием с помощью MobX

MobX[5] — это библиотека, делающая управление состоянием приложения простым и масштабируемым, применяя функционально-реактивное программирование. Философия MobX очень проста: “Все, что может быть получено из состояния приложения, должно быть получено. Автоматически.”.

React отрисовывает состояние приложения, предоставляя механизмы для перевода его в DOM-дерево. MobX предоставляет механизм хранения и обновления состояния приложения, которое затем может использовать React.

Состояние — это данные, которые управляют реактивным приложением. MobX добавляет возможность наблюдать за существующими структурами данных, такими как объекты, массивы и экземпляры классов. Чтобы это сделать, нужно обернуть необходимое свойство класса с помощью декоратора `@observable`.

Наблюдаемыми значениями могут быть JavaScript-примитивы, ссылки, объекты, экземпляры классов, массивы и т.д. Если значением является массив или объект, при выводе переменной будет возвращен Observable Array или Observable Object соответственно.

При использовании React, можно сделать компоненты реактивными, добавив декоратор `@observer` из пакета `mobx-react`.

`@observer` говорит React-компоненту реагировать на изменение данных. MobX гарантирует, что компоненты будут перерисовываться, когда это необходимо, но так же не больше, чем нужно.

Для правильного реагирования компонента-наблюдателя, ему нужно передавать не внутренние свойства наблюдаемых объектов, а весь объект целиком, т.к. в первом случае будет передано лишь значения, изменение которого ни на что не повлияет.

`observable`-переменные можно определять не только внутри специального файла Store, но и внутри React-компонента, обернутого наблюдателем `observer`. В данном случае, как и при классическом использовании, изменение значения переменной не будет отражено в классических методах жизненного цикла React, кроме `componentDidUpdate` и `componentWillUpdate`. При необходимости их использования, придется отказаться от `mobx`-обертки в пользу обычного `state`.

`@actions` в `mobx` – это все, что изменяет состояние. С MobX можно явно указать в своем коде, где прописаны ваши actions, пометив их. Actions помогут лучше структурировать код.

Рекомендуется использовать декоратор `@action` для любой функции, которая изменяет `observable` данные. `@action` также предоставляет полезную информацию об отладке в сочетании с `devtools`.

## 1.8. Авторизация с помощью JWT-токенов

JSON Web Token (JWT)[6] - это открытый стандарт ( RFC 7519 ), который определяет компактный и автономный способ безопасной передачи информации между клиентом и сервером в виде объекта JSON. JWT могут быть подписаны с использованием секретного ключа (с помощью алгоритма HMAC ) или пары открытого/закрытого ключей с использованием RSA или ECDSA.

### 1.8.1. Авторизация

Это наиболее распространенный сценарий использования JWT. Как только пользователь вошел в систему, каждый последующий запрос будет включать JWT, позволяющий пользователю получать доступ к маршрутам, службам и ресурсам, которые разрешены с этим токеном.

### 1.8.2. Обмен информацией

Web-токены являются хорошим способом безопасной передачи информации между клиентом и сервером. Поскольку JWT могут быть подписаны можно быть уверенным, что отправители запросов - это именно те, кто ожидается, а не злоумышленники. Кроме того, поскольку подпись рассчитывается с использованием header-а и payload-а также можно убедиться, что содержимое запроса не было подделано.

### 1.8.3. Структура JSON Web Token

В своей компактной форме веб-токены JSON состоят из трех частей, разделенных точками.

- Header (Заголовок)
- Payload (Полезная нагрузка)
- Signature (Сигнатура/Подпись)

Заголовок обычно состоит из двух частей: типа токена, и используемого алгоритма подписи, такого как HMAC SHA256 или RSA. Затем этот JSON кодируется Base64Url для формирования первой части JWT.

Payload — это любые данные, которые нужно передать в токене. Но стандарт предусматривает несколько зарезервированных полей:

- iss — (issuer) издатель токена
- sub — (subject) "тема назначение токена
- aud — (audience) аудитория, получатели токена
- exp — (expire time) срок действия токена
- nbf — (not before) срок, до которого токен не действителен
- iat — (issued at) время создания токена
- jti — (JWT id) идентификатор токена

Payload не шифруется при использовании токена, поэтому не стоит передавать в нем данные, которые не должны попасть в открытый доступ.

Подпись вычисляется на основе заголовка и нагрузки. Таким образом, если кто-то попытается изменить данные в токене, он не сможет изменить подпись, не

зная закрытого ключа. При аутентификации закрытым ключом может выступать пароль пользователя (или хеш от пароля).

Сначала header и payload приводятся к формату JSON, а затем переводятся в base64 – стандарт кодирования двоичных данных при помощи только 64 символов ASCII (рис. 3).

---

```
Header: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
Payload: eyJpc3MiOiJDb2RleCBUZWFtIiwic3ViIjoieYXV0aCI9ImV4cCI6MTUwNTQ2Nzc1Njg2OSwiaWF0IjoxNTA1NDY3MTUyMDY5LCJ1c2VyIjoxfQ
```

**Рис. 3** — Заголовок и полезная нагрузка в base64

Затем, две эти строки соединяются через точку и хэшируются указанным в header алгоритмом.

После первого входа на сайт, клиенту возвращается сгенерированный сервером JWT. При каждом следующем запросе, клиент должен передавать JWT установленным API способом (например, через заголовок или как параметр запроса). Сервер декодирует header и payload и проверяет зарезервированные поля. Если все в порядке, по указанному в header алгоритму составляется подпись. Если полученная подпись совпадает с переданной, пользователя авторизуют.

#### 1.8.4. Access Token и Refresh Token

Зачем нужен refresh-token если есть access-token? Access-token используется при запросах к серверу (например, при логине). У него есть два свойства: он многоразовый и короткоживущий. Время выбирается на основании того, как используется сервис. Refresh-token, используется для обновления пары access и refresh токенов. У него тоже есть два свойства, обратные первому токenu: он одноразовый и долгоживущий.

Схема использования у токенов следующая:

- Пользователь логинится в приложении, передавая логин и пароль на сервер. Они не сохраняются на устройстве, а сервер возвращает два токена и время их жизни
- Приложение сохраняет токены и использует access token для последующих запросов
- Когда время жизни access token подходит к концу (приложение может само проверять время жизни, или дождаться пока во время очередного использования сервер ответит «ой, всё»), приложение использует refresh token, чтобы обновить оба токена и продолжить использовать новый access token

Представим ситуацию, где злоумышленник узнал оба ваших токена, но не воспользовался refresh.

В этом случае он получит доступ к сервису на время жизни access token. Как только оно истечет и приложение, которым вы пользуетесь, воспользуется refresh token, сервер вернет новую пару токенов, а те, что узнал злоумышленник, станут невалидными.

В случае, если взломщик воспользовался refresh, оба ваши токена истекают, приложение предлагает вам авторизоваться логином и паролем, сервер возвращает новую пару токенов, а те, что узнал злоумышленник, снова станут невалидными.

Таким образом, схема refresh + access token ограничивает время, на которое атакующий может получить доступ к сервису. По сравнению с одним токеном, которым хакер может пользоваться неделями и никто об этом не узнает.



## **2. О задаче**

### **2.1. Постановка задачи**

Требуется создать web-приложение, которое упрощает создание и обновление резюме кандидатами и работниками компании Akvelon, а также решает многие проблемы рекрутеров, оптимизируя их работу и тем самым сокращая время, проведенное над редактированием документов.

### **2.2. Требуемый функционал**

1. Возможность создания, копирования, редактирования и архивирования резюме;
2. Наличие базы данных, в которой бы хранились названия компаний, институтов, проектов, навыков, персональных результатов и сфер ответственности;
3. Автоматическое заполнение перечисленных данных в поля резюме - всплывающие подсказки и поиск по ним;
4. Возможность пополнения этой базы данных как обычными пользователями так и администраторами сайта;
5. Модерация добавленных данных администраторами в один клик;
6. Подobie папок с проектами, на которые можно назначить кандидатов и производить поиск по имени и позиции;
7. Скачивание резюме в формате .docx, стилизованное под стандартное резюме Akvelon;
8. Заполнение общей информации о кандидате с помощью подсказок с логическими выделенными словосочетаниями; которые превращаются в подобие шаблона при их выборе. Подсказки должны предлагаться в случайном порядке, чтобы повысить уникальность текста в резюме;
9. Пользователь приложения должен иметь возможность указать свою роль на проекте, для которого создается резюме; Смена этой роли должна вызвать автоматическую пересортировку данных, чтобы наиболее актуальные для позиции умения находились выше остальных;
10. Сайт нужно создать в стиле Akvelon, придерживаясь дизайна других сервисов данной компании;
11. Возможность дать другим пользователям права модератора;
12. Блокировка и удаление пользователя;
13. Всплывающие уведомления об ошибках и прочей информации для пользователя;

14. Интерфейс для отслеживания прогресса работы приложения.

### **2.3. Используемые программные средства**

Исходя из того, что требуется написать клиентскую часть приложения, для разработки были выбраны следующие программные средства:

- VSCode[7] для разработки и отладки приложения;
- JavaScript в качестве основного языка программирования;
- GitLab для управления репозиторием кода для Git;
- MobX для управления состоянием приложения;
- Axios для взаимодействия с API;
- React.js для создания интерфейса;
- Material UI для создания единого стиля компонентов;
- LESS для корректировок стиля Matreial и для создания собственного;

## 3. Решение задачи

### 3.1. Создание базовой архитектуры

В компании автору работы предоставили готовый шаблон со структурой, где уже подключен Webpack, настроено несколько правил ESLint для поддержания кода чистым и более приятным глазу. Для начала разработки была реализована следующая структура проекта в директории src:

```
/
├── components
├── containers
├── services
│   ├── action.notify
│   ├── toast.notify
│   ├── stores
│   ├── request.services
│   └── validator
├── app.js
├── app.less
├── constants.js
├── index.html
├── muitheme.js
├── router.js
└── variables.less
```

- Папка components предназначена для react-компонентов для многоразового использования, непривязанных к какому-либо контексту, желательно максимально абстрактных.
- Containers - каталог для логически разделенных папок, содержащих в себе компоненты конкретных страниц.
- Services - папка для сервисов, которые отвечают за реализацию кода, независимого от внешнего окружения. В данном приложении понадобились сервисы для логики полос загрузки данных, появления уведомлений, взаимодействия с API, валидации, и действий с observable-состояниями MobX-a.
- index.html - точка входа приложения, в нем описываются подключения стилей и скрипта для рендера.
- index.js указывает, в какую область html документа будет проецироваться DOM-дерево и рендерит app.js.
- app.js содержит компоненты-провайдеры, отвечающие за авторизацию, инициализацию MobX stores, стилей-muitheme и перенаправления на страницы.

- `app.less` - в этом файле написаны общие стили, которые используются практически во всех компонентах.
- `constants.js` - переменные, которые используются в разных местах программы по типу предложений, коэффициентов, регулярных выражений.
- `muitheme.js` - файл, позволяющий задать конфигурации Material UI стилей.
- `router.js` - компонент-маршрутизатор, определяет какой обработчик надо вызвать для конкретного маршрута.
- `variables.less` - содержит палитру именных основных цветов сайта. Файл служит для удобства, чтобы было проще ориентироваться на название переменной, а не на HEX или RGB коды.

### 3.2. Создание сервиса запросов, авторизация

В данном разделе приводится пример только один из многочисленных сервисов для работы с запросами в HRCV-Portal.

Первым делом предстояло как-то идентифицировать пользователей, для чего был создан первый сервис в папке `services` - `request.service` со следующей структурой:

```
request.service
├── api
│   ├── auth.js
│   └── index.js
```

`index.js` содержит логику неявной авторизации, а также функцию `sendRequest`, отвечающую за то, чтобы во все заголовки запросов подкладывался валидный `token`.

`auth.js` – один из классов с асинхронными функциями, которые формируют `axios`-конфигурацию для отправки запроса на сервер. Для авторизации понадобились следующие функции:

1. `signUp` – выполняет POST-запрос, в котором отправляются данные для инициализации нового пользователя;
2. `signIn` – POST-запрос, который принимает логин и пароль, а в ответ мы получаем информацию о пользователе, его `refresh` и `access token`;
3. `refresh` – POST-запрос, в котором отправляется `refresh-token`, чтобы обратно получить `access-token`. `Refresh` хранится в `locale storage` – постоянном хранилище данных – и его действие истекает через месяц полного бездействия аккаунта, в то время как `access-token` "живет" пол часа и обновляется каждый раз, когда пользователь перезагружает страницу, при условии, что `refresh` еще актуален; Все это позволяет не хранить в целях безопасности `access-token` в локальном хранилище. И даже если злоумышленник вдруг перехватит `access-token`, то у него будет лишь пол часа. Также эта идея позволяет пользователю сэкономить время на вбивание данных для входа.

Страницу со входом можно будет наблюдать в случае, если человек выйдет из аккаунта сам или если пройдет месяц и его refresh-token станет невалидным;

4. getPermissions – GET-запрос, который возвращает информацию о правах, чтобы различать админа от рядового пользователя.

Теперь, когда запросы описаны, автор работы написала компонент-обертку над app.js, в которой происходит логика неявной авторизации, если пользователь перезагружает страницу.

В классе AuthorizaionProvider в конструкторе задается начальное состояние компонента – isAuthorized, равное false. Пока оно false, на интерфейсе отображается только белый фон с полосой загрузки. При каждом первоначальном рендеринге (монтировании) этого компонента вызовется функция silentAuthorization, обернутая в try/catch. Она выполняет следующие действия:

1. Из локального хранилища достается refresh-token;
2. Этот refresh отправляется на сервер, ожидая получить новый access-token;
3. Если с момента входа в аккаунт прошел месяц и refresh истек, то возвращается ошибка, срабатывает catch и пользователя перенаправляет на страницу входа, чтобы он зашел вновь;
4. Если нет, то внутри программного кода сохраняется access-token, а в локальное хранилище записывается новый продленный refresh;
5. Определяются права пользователя;
6. isAuthorized становится true и пользователю доступен интерфейс приложения.

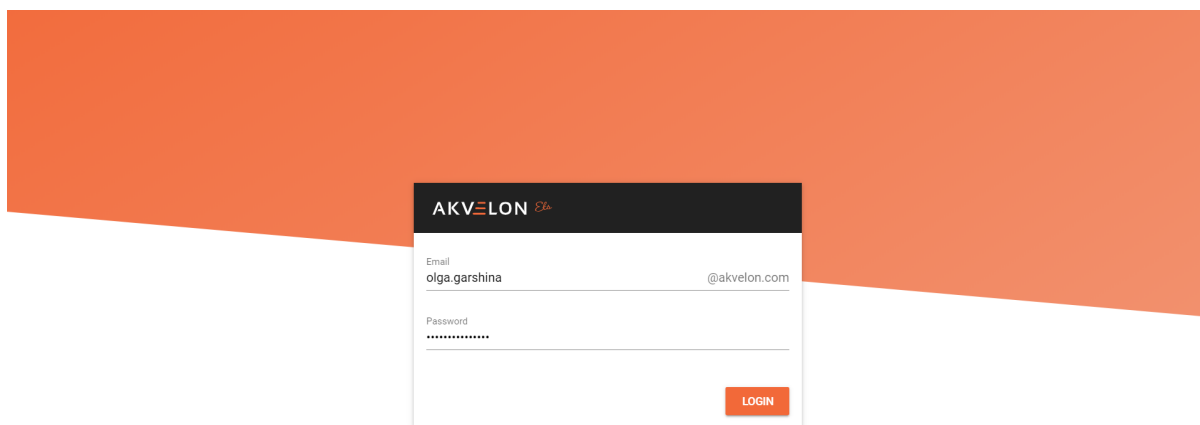
### 3.3. Регистрация, валидация и вход на сервис

Клиент поставил условие – страницы, связанные с авторизацией должны быть выполнены в таком же стиле, что и сайт компании для оценки рабочего времени, написанный на Vue.js. Но должна быть возможность входа с любой почтой, а не с доменным именем.

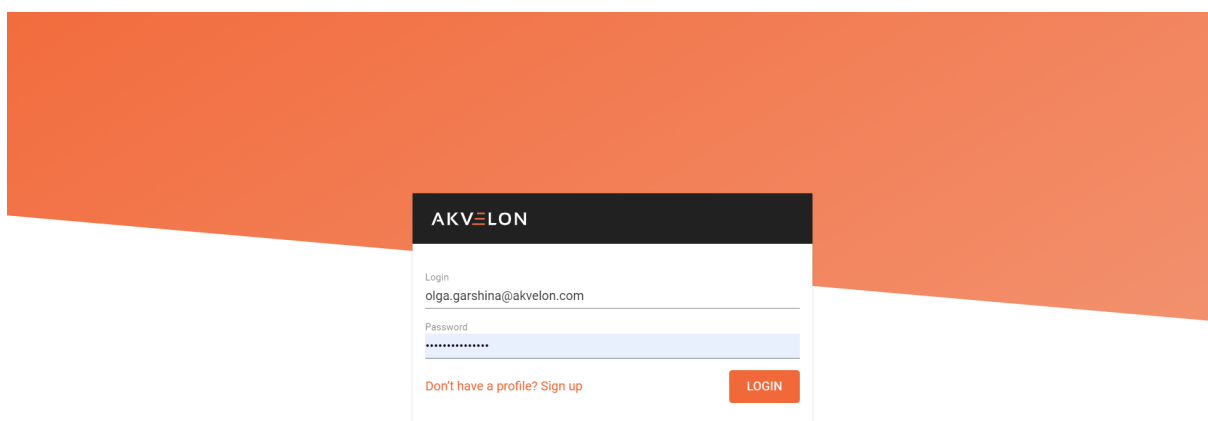
В результате получилось застилизовать достаточно похоже, но с помощью React, LESS и Material-UI компонентов (рис. 5):

Следующий шаг — сделать валидацию на странице регистрации. Для этого был сделан еще один сервис со следующей архитектурой.

```
validator
├── isValidPassword
├── isValidEmail.js
├── isEmpty.js
├── isValidLength.js
├── isValidName.js
```



**Рис. 4** — Страница входа на ets.akvelon.com



**Рис. 5** — Страница входа на hrcv.inyar.ru

В результате работы этих функций возвращается булевый результат и сообщение об ошибке, если он false. Если ошибки могут быть разного типа для одного поля ввода, то сначала покажется одна, а уже при ее исправлении – следующая.

- isEmpty проверяют строку на пустоту. С помощью функции JavaScript trim() удаляются пробелы, а длина того, что осталось сравнивается с нулем.
- isValidLength проверяет строку на соответствие минимальной и максимальной длине строки, переданных в функцию.
- isValidPassword и isValidName проверяют строку на соответствие регулярному выражению из файла constants, вызывает внутри isEmpty и isValidLength.
- isValidEmail валидирует строку на корректное написание e-mail, условия успешной проверки хранятся в constants.

Также в целях интереса и знакомства с технологией добавлена ReCAPTCHA[8] — система, разработанная в университете Карнеги-Меллон для защиты web-сайтов от интернет-ботов.

В SignUp react-компоненте заведены состояния для каждого условия успешной регистрации на сайте. Пока все условия не пройдут проверку, кнопка login будет заблокирована, а рядом с полями, в которых допущена ошибка будут отображаться наглядные красные сообщения.

AKVELON

First name  
Olga

Last name  
Гаршина  
Field must contain only Latin letters and hyphen

Email  
qwerty@mail.ru

Password  
\*\*\*\*\*

Confirm password  
\*\*\*\*  
This password does not match that entered in the password field

Время проверки истекло. Установите флажок и повторите попытку.

☐ Я не робот

reCAPTCHA  
Конфиденциальность - Условия использования

Already have a profile? Login

SIGN UP

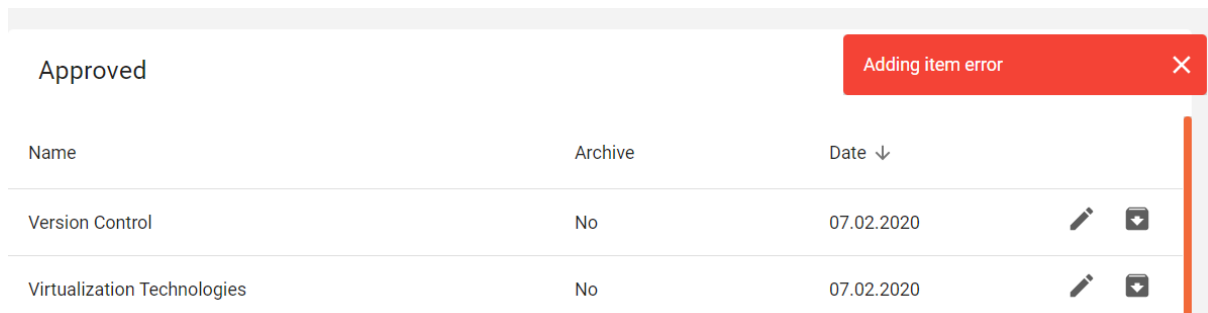
Рис. 6 — Страница входа на ets.akvelon.com

### 3.4. Сервис уведомлений пользователя, отслеживание прогресса загрузки

На данном этапе разработки понадобилось оповещать пользователя об ошибках со стороны сервера и другой полезной информации. Для этого автор работы создала сервис toast.notify и ToastTrigger компонент.

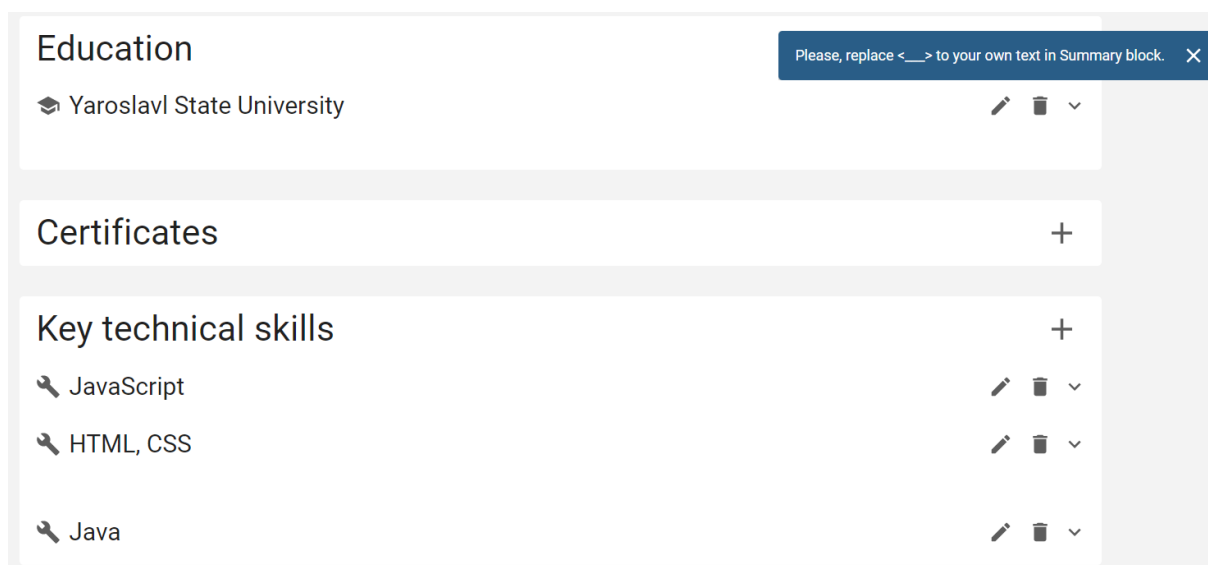
В этом компоненте инициализируются состояния, отвечающие за отслеживание того, отображается плашка с оповещениями или нет, за текст сообщения и

за тип уведомления. При монтировании ToastTrigger как бы говорит toast.notify сервису, что если в коде в каком-нибудь месте вызовется функция-уведомитель, в параметрах которой будут переданы сообщение и тип уведомления – состояние компонента должно поменяться соответственно. На рисунке 7 отражен вызов функции в месте кода, где сервер вернул ошибку добавления новой категории.



**Рис. 7** — Пример уведомления об ошибке

Автор дипломной работы поделила оповещения на 4 типа: предупреждение, ошибка, информация и успех. С помощью LESS очень удобно переопределять стиль плашки в зависимости от параметров, переданных в функцию. Вот, например, плашка с информацией, которая вызывается при попытке сохранения резюме, если пользователь забыл ввести обязательную информацию (рис. 8).



**Рис. 8** — Пример информационного уведомления

Что касается сервиса, показывающего прогресс, то он сделан похоже, за исключением того, что action.notify хранит массив из подписок на события, которые должны вызывать полосы загрузки. Таким образом на странице может отображаться сразу несколько полос прогресса получения данных в разных местах. Достаточно обернуть их компонентом LoaderTrigger, передав имя события, на которое подпишется сервис.



Данный пример иллюстрирует отображение полос прогресса, пока с сервера подгружаются коллекции компаний и обрабатывается запрос на перенос компании из списка предложенных в список одобренных (рис. 9). На самом деле это делается мгновенно и полоса загрузки является больше дружественным интерфейсом для пользователей с медленным интернетом. Для отображения прогресса достаточно вызвать две функции: перед началом асинхронного запроса на сервер, передав параметр "start" и после его окончания, передав "finish".

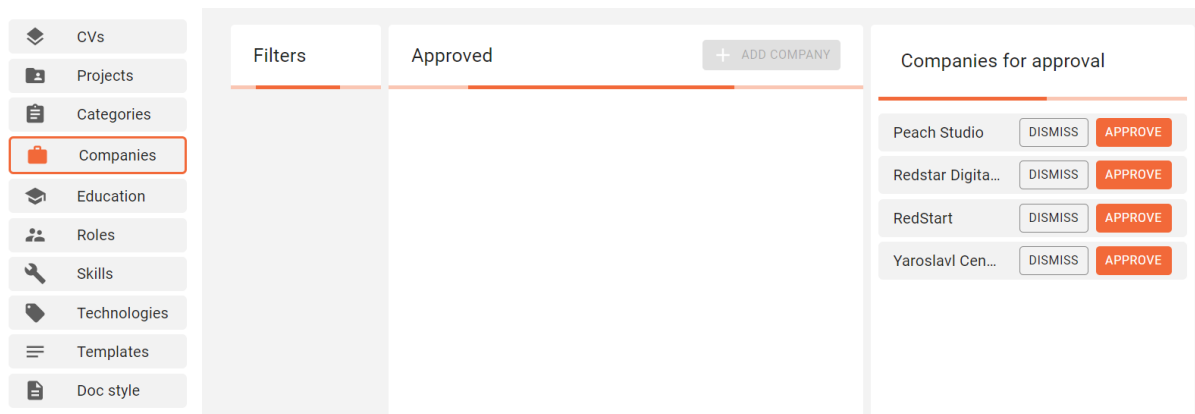


Рис. 9 — Интерфейс полос загрузки

### 3.5. Страница создания резюме взглядом пользователя

Создание резюме проходит в несколько этапов и создается для отправки клиенту Akvelon на определенный проект. Для начала выбирается роль, в зависимости от которой будут подсказываться автоматические предложения для заполнения и сортироваться технологии, которые знает кандидат. Это значит, что тестировщику, например, будет подсказываться текст, связанный с написанием UNIT-тестов, а специалисту DevOps - с настройкой CI/CD процессов.

Клиент пожелал, чтобы данные подсказки предлагались в случайном порядке – это сделает резюме более уникальным, если его будет заполнять не специально обученный рекрутер, а кандидат на позицию. Далее заполняется поле Summary - краткая выжимка об умениях и качествах работника. В данном поле при нажатии на пункт меню конкретные словосочетания превращаются в шаблоны, выделенные жирным – такое поведение также попросил заказчик.

Как можно заметить, все это (рис. 10) не похоже на типичное поведение обычного компонента текстового поля из Material UI. Все потому, что данный интерфейс – это замаскированный под material design с помощью LESS фреймворк Draft.js – текстовый редактор, переделанный под нужды автора работы. Ею было потрачено немало времени, чтобы разобраться в работе библиотеки, убрать лишние детали и добавить нужные.

Поле роль — созданный автором дипломной работы компонент. Изначально, в Material UI не было достойного решения для выбора пункта из списка, с полем

Summary

Full-stack software development engineer with more than 3 years of experience of designing and developing either web services using C#, Java and Python or high-quality UI with a variety of tools such as React.js, jQuery, Electron.js and JavaScript. Hands-on experience in UI performance analysis and optimization. Thorough experience of working with Azure services and API. Expert in hybrid app development using React Native, Flutter, Cordova. Strong focus on <\_>, but also has experience with <\_>. Hands-on experience in <\_>.

Pattern search

- Strong focus on **Ruby/Rails**, but also has experience with **Elixir, Solidity, Java, JavaScript, Golang**. Hands-on experience in **Java and Kotlin**.
- Deep knowledge in **fintech and banking domain**.
- Have a passion for **high-quality**.
- Proficiency / Well-developed skills / in **developing high performance software in C/C++ using object oriented programming**.
- Passionate about **UI, UX, design systems**.

**Рис. 10** — Интерфейс заполнения поля Summary

для поиска по нему. Автору предлагались только сторонние библиотеки, которые имели в себе много лишних зависимостей и строк кода, котор неприятно читать и сложно в них разбираться. После чего наступил переломный момент и автор работы решила написать свой компонент с тем поведением, который идеально бы подошел. Несмотря на то, что создание простого казалось бы списка со встроенным поиском выглядело тривиальной задачей, в процессе написания автор столкнулась со многими подводными камнями, связанными с обработкой событий в React. Она искала ответы в интернет сообществах по программированию, но встречала только некрасивые решения.

В итоге автору удалось сделать компонент (рис. 11), отвечающий желаемому функционалу и хранящий в состоянии не строку, а целый объект, что не тратит ресурсы программы на дополнительные преобразования перед отправкой запроса на сервер.

Assign role

\$

- SDET
- System Administrator
- Data Analyst
- Data Scientist
- DevOps
- SDE

**Рис. 11** — Компонент для выбора из списка с поиском

После кандидат заполняет информацию об опыте работы в компаниях в блоке professional experience. Автор дипломной работы вдохновлялась дизайном

заполнения информации на сайте linkedin при разработке данного блока (рис. 12). На его примере показан функционал подсказок для автозаполнения и принцип пополнения базы данных.

## Add the company

Company \*

Company description

Position

Start date

June 2020

End date

June 2020

☐ To the present

Projects

+

If you can't specify certain projects, please fill overall information about your experience in this company:

Responsibilities

? Add responsibility

+

Personal results

? Add personal result

+

Technologies

Add technology

+

ADD COMPANY

CANCEL

**Рис. 12** — Диалоговое окно добавления компании

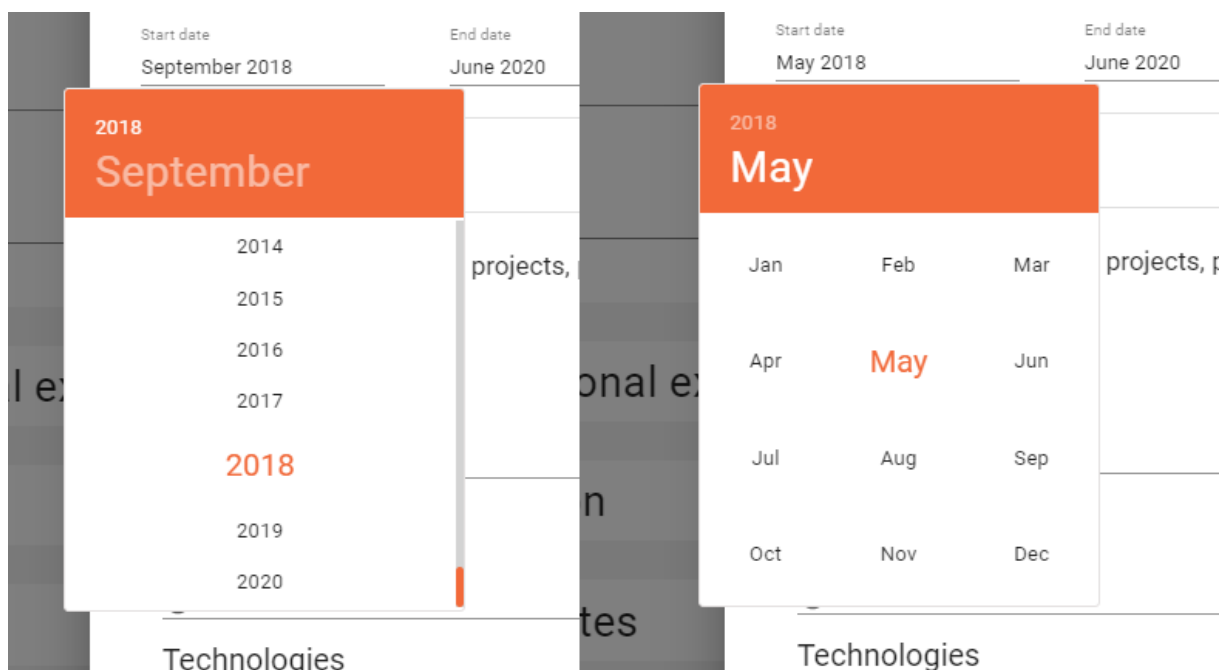
Как только пользователь начинает набирать текст в поле Company, под ним раскрывается список тех компаний для заполнения, которые включают в себя набранное буквосочетание. Если база не включает в себя название компании, которое хочет вбить кандидат, то после нажатия кнопки "Add" она отправляется на рассмотрение админами в часть интерфейса, скрытого от обычного пользователя.

Далее идут обычные текстовые поля для заполнения краткой сводки о компании и о занимаемой позиции в ней.

После кандидату необходимо указать время работы в компании. При нажа-

27

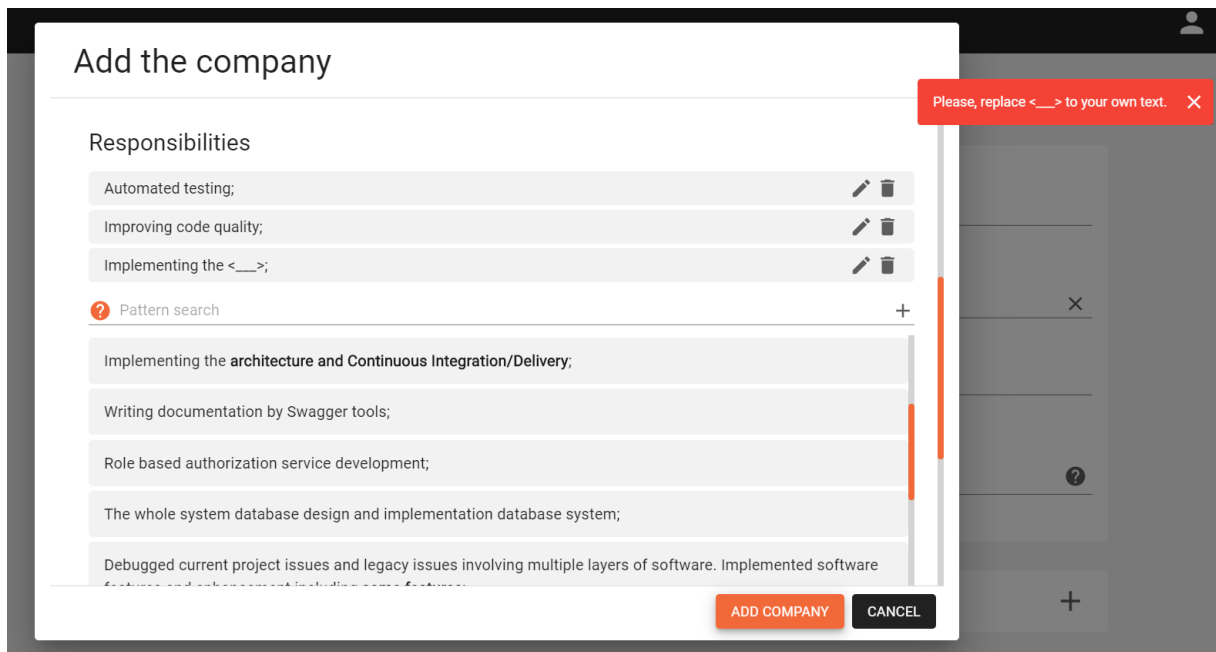
тии на поля start date/end date поочередно открывается интерфейс для выбора года и месяца (рис. 13). В случае, если кандидат все еще является сотрудником описываемой компании, он может нажать галочку рядом с "To the present" тем самым автоматически скроется возможность выбора даты окончания работы.



**Рис. 13** — Интерфейс выбора даты

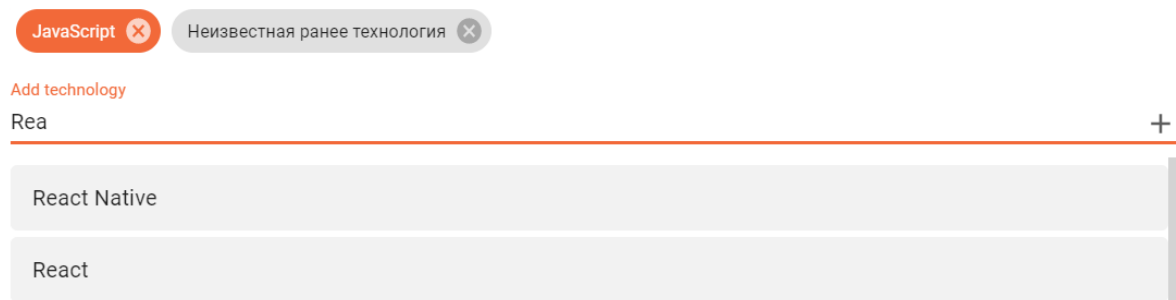
Потом подразумевается заполнение информации о проектах. Если кандидат не может выделить каких-то конкретных, то требуется заполнить общую информацию. Предлагается внести данные о персональных результатах, выполненных задачах и используемых технологиях. Некоторые люди сталкиваются с проблемой того, что они не могут сразу быстро вспомнить все свои достижения. Для этого автор работы добавила возможность раскрыть по кнопке со знаком вопроса меню с подсказками шаблонных фраз, которые как и в поле Summary относятся к выбранной в начале роли. Если кандидат оставил место для шаблона незаполненным – это валидируется. Описанный выше функционал можно наблюдать на рисунке 14.

Заполнение блока заканчивается указанием используемых в компании технологий, которые автозаполняются и попадают в базу данных на рассмотрение админами. Так как технологии являются коллекцией с самой большой базой, их модерация достойна отдельного внимания. Чтобы глаз рекрутера мог зацепиться за потенциальное некорректное слово, было принято решение сделать цветовой акцент на интерфейсе. Таким образом, одобренные технологии окрашены в оранжевый, а остальные в серый (рис. 15). Чтобы пользователь не засорял базу альтернативными названиями технологий было к ним привязывается список синонимов. Так, введя в поле JS и нажав ввод, кандидат получит корректное названия языка на выходе – JavaScript.



**Рис. 14** — Интерфейс заполнения информацией поля responsibility и её валидация

## Technologies



**Рис. 15** — Заполнение инфо-блока технологиями

Если же проекты к заполнению все же есть, то по нажатию на плюс всплывает еще одно диалоговое окно, где нужно указать название проекта, его описание и те же пункты, что и в блоке добавления после выбора времени работы на позиции.

Всё доступно для редактирования и полного удаления, все информационные списки можно сворачивать и разворачивать. Чтобы ненужная на момент заполнения информация не мешала на фоне в развернутом состоянии, в режиме редактирования CV находится только один логический блок.

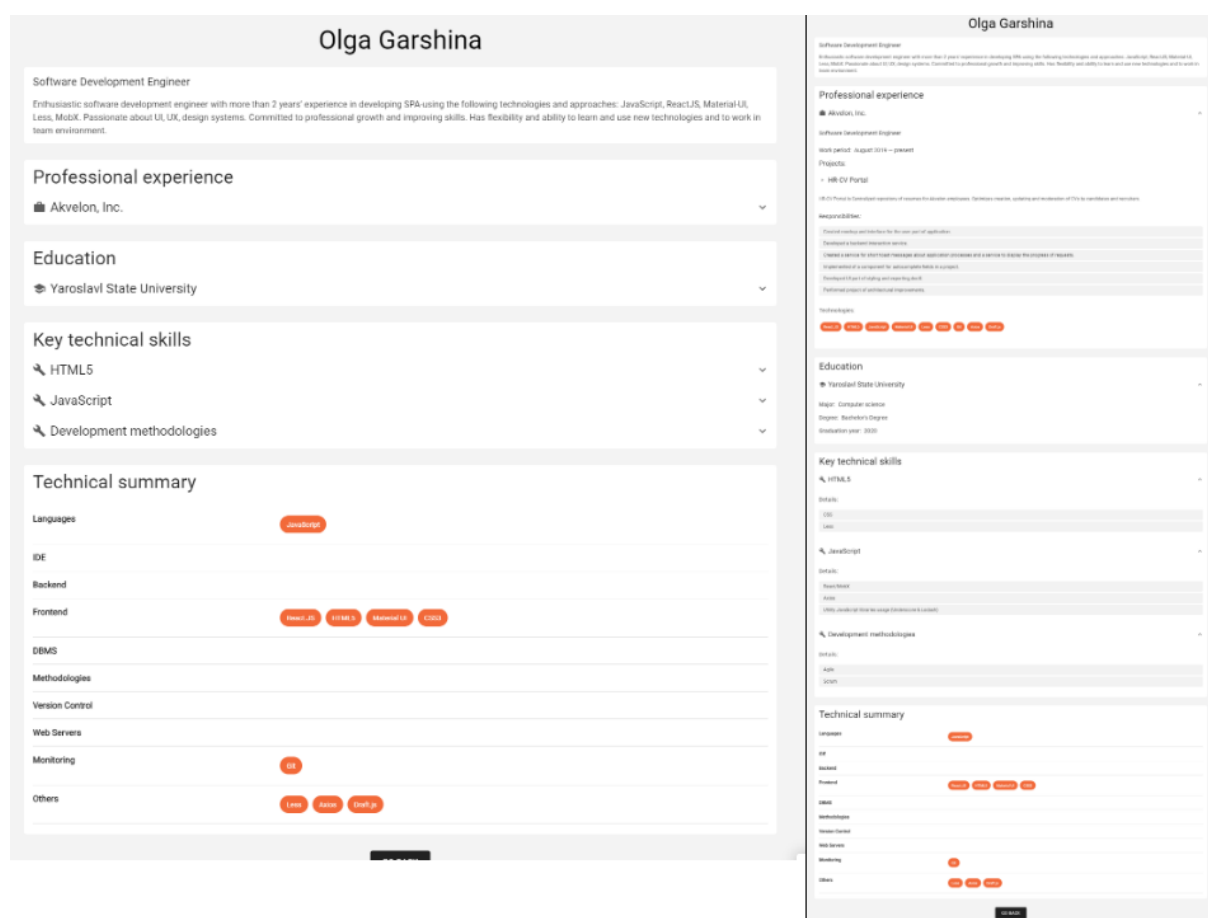
Преимущество данного UX-решения особенно хорошо наблюдается в режиме просмотра резюме. В нем можно развернуть всю структуру документа для полноты восприятия, страница приложения станет действительно длинной (рис. 16), особенно если речь идет об опыте работы программиста-сеньора. Также в данном режиме полностью отсутствуют отвлекающие кнопки для какого-либо взаимодействия, он отлично подходит для того, чтобы рекрутер мог просто поде-

литься ссылкой на страницу с голой информацией о резюме.

Блоки education, certificates и key technical skills по своей структуре похожи на блок professional experience, а вот technical summary хочется уделить особое внимание. В нем автоматически сгенерирован и категоризирован список всех технологий, который пользователь затрагивал, заполняя резюме. Блок меняет свой вид при изменении роли, потому что у программиста с большим стеком он громаден, а клиент хочет сэкономить свое время и увидеть приоритетную для него информацию в первых же строках. Если технологию указывают в резюме на различных проектах и различных компаниях, написанный автором работы алгоритм валидирует хранение в данном списке только уникальных наименований технологий.

Также это то место, где кандидат может вбить изученные им вне работы технологии. Так как это тоже играло роль, когда клиенты Akvelon-а выбирали, на какой стек поставить работника на проекте.

Технологии, категории которых не определены, всегда попадают в Others по регламенту стандартного Akvelon-резюме. Список категорий у каждой роли свой и фиксирован, поэтому если кандидат не знает ни одной технологии из категории, ее строка остается пустой (рис. 16).

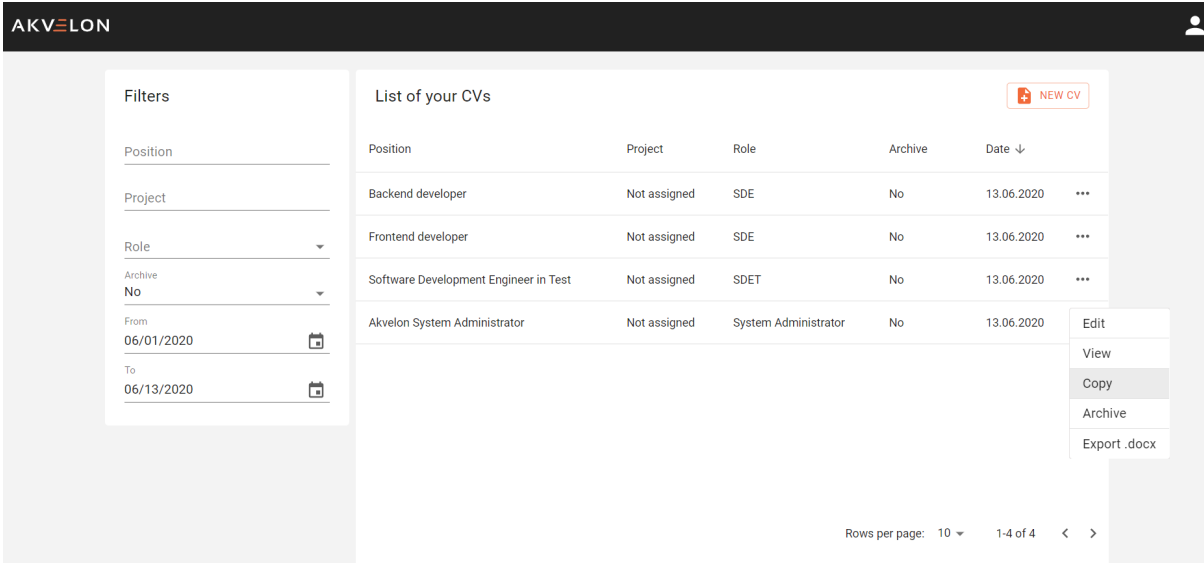


**Рис. 16** — Слева отображен режим просмотра с примером со свернутыми инфо-блоками, справа с развернутыми

### 3.6. Таблица всех резюме взглядом пользователя

Для удобства поиска резюме и фильтрации была реализована идея с таблицей (рис. 17). Фильтрацию можно воспроизводить по имени, роли, позиции, проекту, по дате и по типу резюме исходя из того, скрыт он от глаз или нет. В зависимости от нужд пользователя, количество строк в таблице можно менять от 5 до 50 на странице. Нажимая на кнопку с троеточием кандидат может заархивировать (спрятать) резюме, копировать, перейти в режим редактирования или просмотра, а также экспортировать стилизованный документ в формате .docx. Пользователь может знать информацию о проекте, к которому привязано его резюме, но не может изменить ее, так как эта привилегия рекрутеров, которые подразумеваются администраторами сайта.

Для экспорта файла была использована библиотека FileSaver.js. Она является решением для сохранения файлов на стороне клиента и идеально подходит для web-приложений, которым необходимо создавать файлы, или для сохранения конфиденциальной информации, которую не следует отправлять на внешний сервер.



The screenshot shows a web application interface for a user. At the top, there is a header with the logo 'AKVELON' and a user profile icon. Below the header, the main content area is divided into two sections. On the left, there is a 'Filters' sidebar with dropdown menus for 'Position', 'Project', 'Role', and 'Archive' (set to 'No'). There are also date pickers for 'From' (06/01/2020) and 'To' (06/13/2020). On the right, there is a table titled 'List of your CVs' with a 'NEW CV' button. The table has columns for 'Position', 'Project', 'Role', 'Archive', 'Date', and a three-dot menu. The table contains four rows of data. A context menu is open for the last row, showing options: 'Edit', 'View', 'Copy', 'Archive', and 'Export .docx'. At the bottom right, there is a pagination control showing 'Rows per page: 10', '1-4 of 4', and navigation arrows.

Position	Project	Role	Archive	Date	
Backend developer	Not assigned	SDE	No	13.06.2020	...
Frontend developer	Not assigned	SDE	No	13.06.2020	...
Software Development Engineer in Test	Not assigned	SDET	No	13.06.2020	...
Akvelon System Administrator	Not assigned	System Administrator	No	13.06.2020	Edit View Copy Archive Export .docx

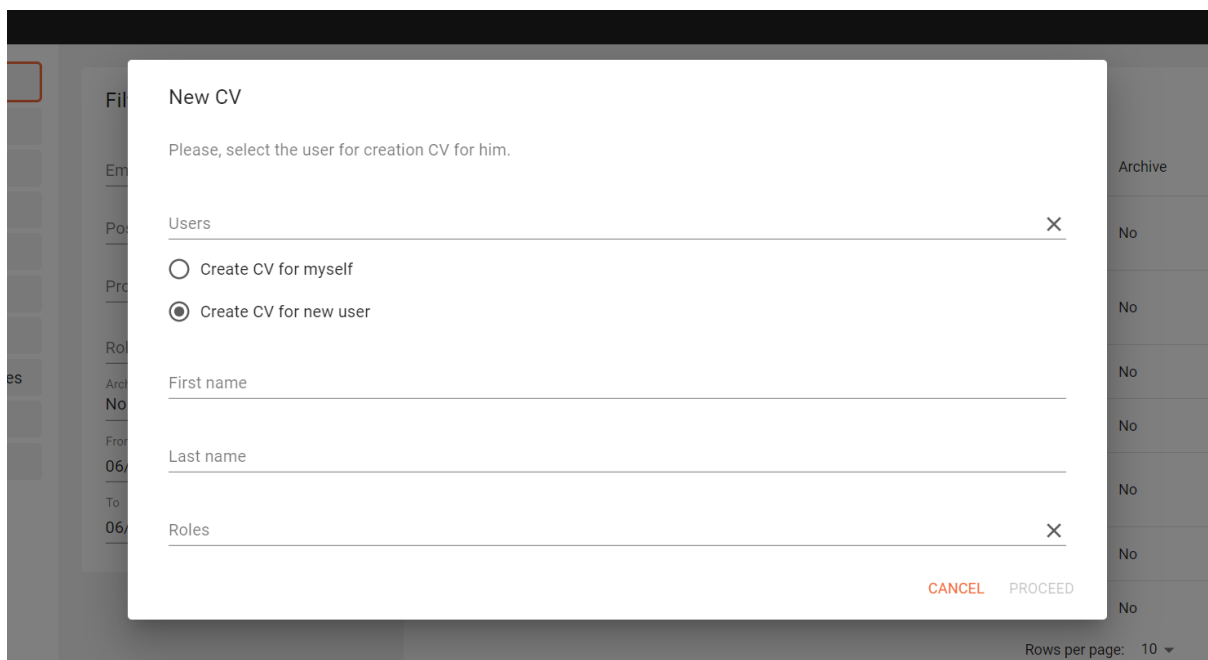
Рис. 17 — Таблица со списком пользовательских резюме

### 3.7. Интерфейс резюме взглядом администратора

Приложение со стороны администратора имеет весь набор функционала, доступный обычному пользователю. Но прав и возможностей для модерирования гораздо больше.

Что касается создания резюме, то администратор может создать его для себя (если он по совместительству кандидат) и для другого пользователя (рис. 18). В планах расширить функционал в сторону создания резюме для гостей (незаре-

гистрированных пользователей), чтобы они смогли заполнить его по временной ссылке. В случае желания зарегистрировать аккаунт, CV бы прикрепилось к пользователю.



**Рис. 18** — Диалог перед созданием резюме на интерфейсе администратора

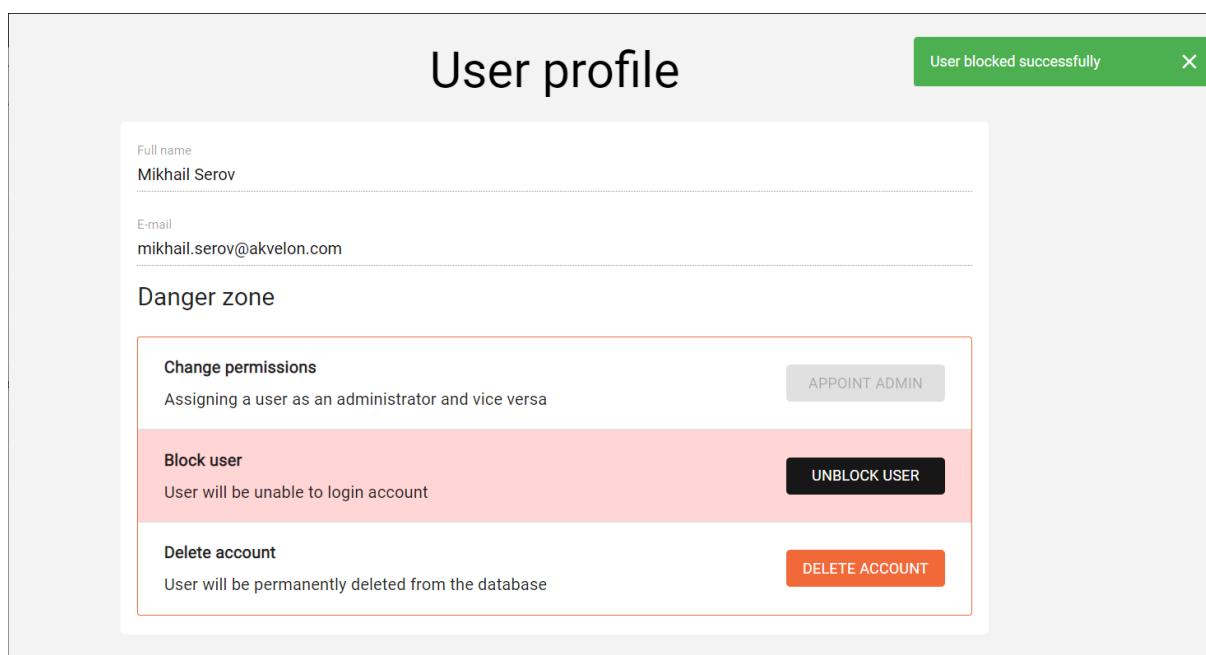
На странице создания резюме рекрутер может назначать его на определенный проект. На работе они специально создают папки на компьютере, чтобы хранить резюме для разных проектов отдельно. С HRCV не нужно создавать каталоги под каждый признак, а достаточно произвести фильтрацию на главной странице.

В верхнем правом углу находится вкладка с настройками. По нажатию администратор перейдет к списку всех пользователей ресурса. Он может произвести поиск по имени и выбрав необходимое кликнуть по пункту меню и перейти на профиль данного человека.

В профиле (рис. 19) рекрутер может дать права администратора или отнять их. Он может заблокировать пользователя и тогда он не зайдет в свой аккаунт. Также присутствует кнопка удаления аккаунта. При нажатии на нее, конечно же, несколько раз уточняется, правда ли админ хочет это сделать, поскольку это необратимый процесс, который повлечет удаление из базы все резюме, привязанных к данному профилю.

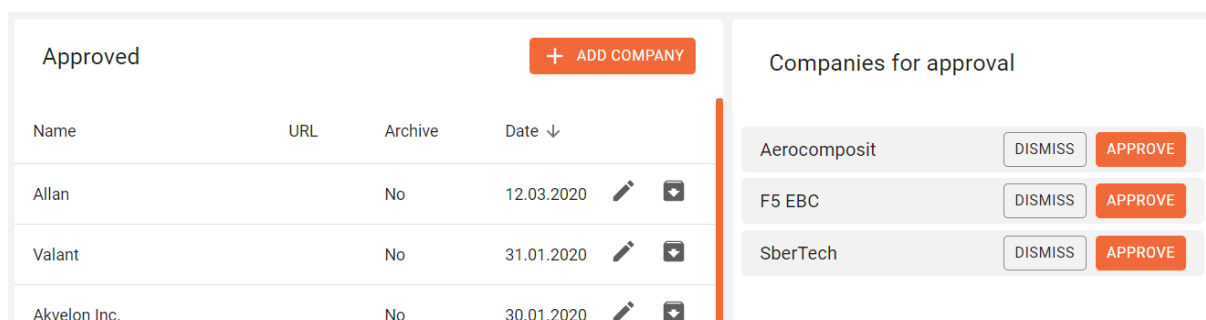
Самым главным отличием админского интерфейса от пользовательского является наличие меню слева страницы, в которых модерируются коллекции базы данных и изменяется конфигурация стилей для экспортируемого .docx документа. Во всех коллекциях можно производить фильтрацию, добавлять новые экземпляры, редактировать их и отправлять в архив.





**Рис. 19** — Пример блокировки пользователя

На данный момент разработки существуют списки компаний, категорий технологий, самих технологий, названий учебных учреждений, ролей, навыков и шаблонных строк для автозаполнения. Компании, технологии и учебные заведения отличаются тем, что администратор их может как одобрить (после чего элемент отправляется в базу данных и в таблицу), так и отклонить в специальном меню на странице их коллекции (рис. 20).



**Рис. 20** — Меню для утверждения или отклонения новых компаний

После списка CV идет таблица, отображающая коллекцию проектов. Это самое подобие папок с проектами, которые требовал клиент. Сначала нужно отправить POST-запрос на добавление в коллекцию путем нажатия на кнопку "Add project" и прописать название. После чего можно назначить резюме на проекты, их список появится в разделе проектов при нажатии на кнопку редактирования (рис. 21).

Далее идет список категорий, который имеет уже описанный выше функционал, впрочем, как и коллекция учебных учреждений.

После следует список компаний, к описанию конкретной компании опцио-

нально прилагается ссылка на ее сайт.

Filters Projects

Edit project

Project name  
HR+CV project

Search by name

Search by position

Sergey Konovalov, DevOps Engineer		
Alex Reilly, Software Developer Engineer		
Olga Garshina, Software Development Engineer		

CANCEL SAVE

**Рис. 21** — Подobie папки проекта

Коллекция ролей имеет важную роль для создания резюме. Сами роли — это фиксированное количество терминов около 20-ти штук, от которых зависит заполнение резюме. Так, например, на рисунке показана роль разработчика-тестировщика, для которого важно сначала отображать технологии из разряда языков программирования, после из сферы тестирования и так далее. Приоритет категорий можно менять местами, для этого реализован так называемый drag-and-drop компонент (рис. 22).

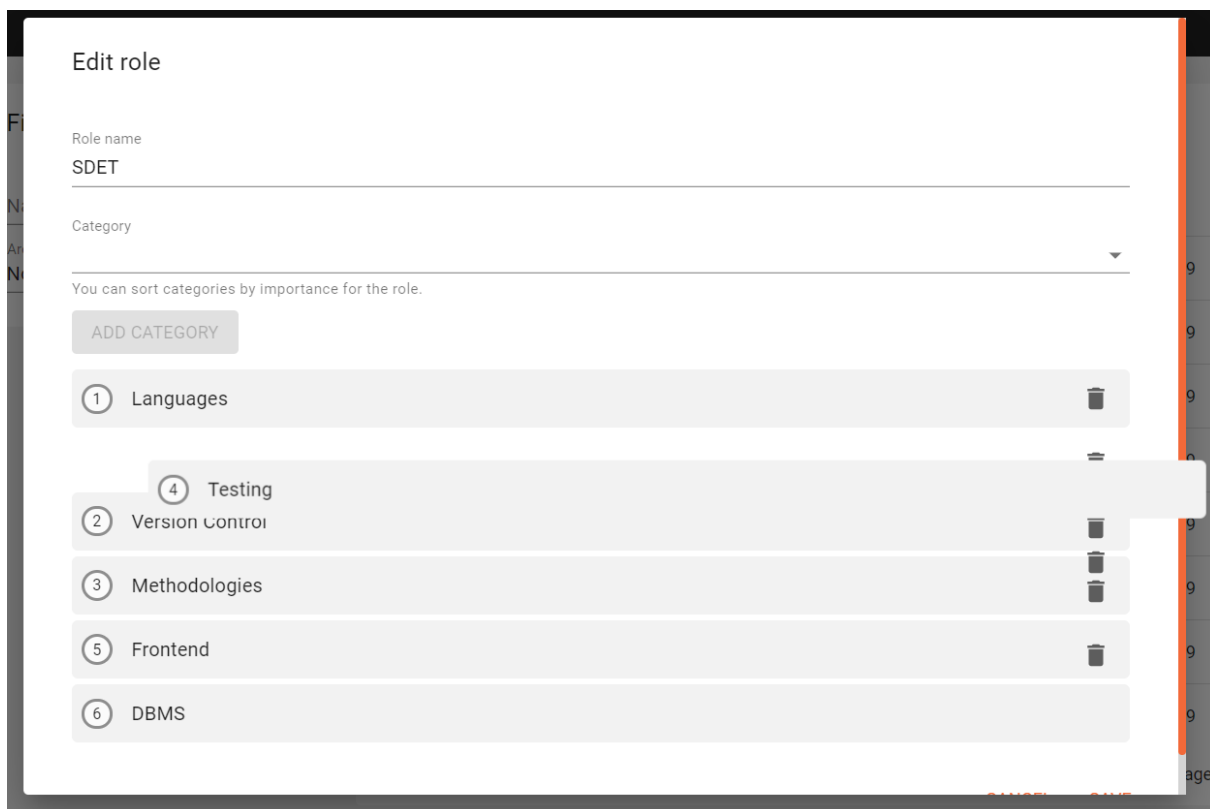
Drag'n'Drop – это возможность захватить мышью элемент и перенести его. В своё время это было замечательным открытием в области интерфейсов, которое позволило упростить большое количество операций.

Перенос мышкой может заменить целую последовательность кликов. И, самое главное, он упрощает внешний вид интерфейса: функции, реализуемые через Drag'n'Drop, в ином случае потребовали бы дополнительных полей, виджетов и т.п.

В коллекции skills к каждому навыку привязан массив из "деталей". Это короткие предложения, которые раскрывают суть навыка и предлагаются пользователю при выборе конкретного умения.

Технологии отличаются тем, что они должны быть привязаны к определенной категории, иначе они автоматически будут попадать в категорию "others".

Коллекция с шаблонами выполняет одну из самых важных ролей. В ней хранятся стандартные фразы для заполнения полей типа summary, personal results и responsibilities. Чтобы словосочетание имело в себе место, выделенное жирным текстом, которое потом превратится в шаблон, рекрутеру надо просто обернуть его в угловые скобки. Также нужно выбрать один из трех типов фразы и закрепить

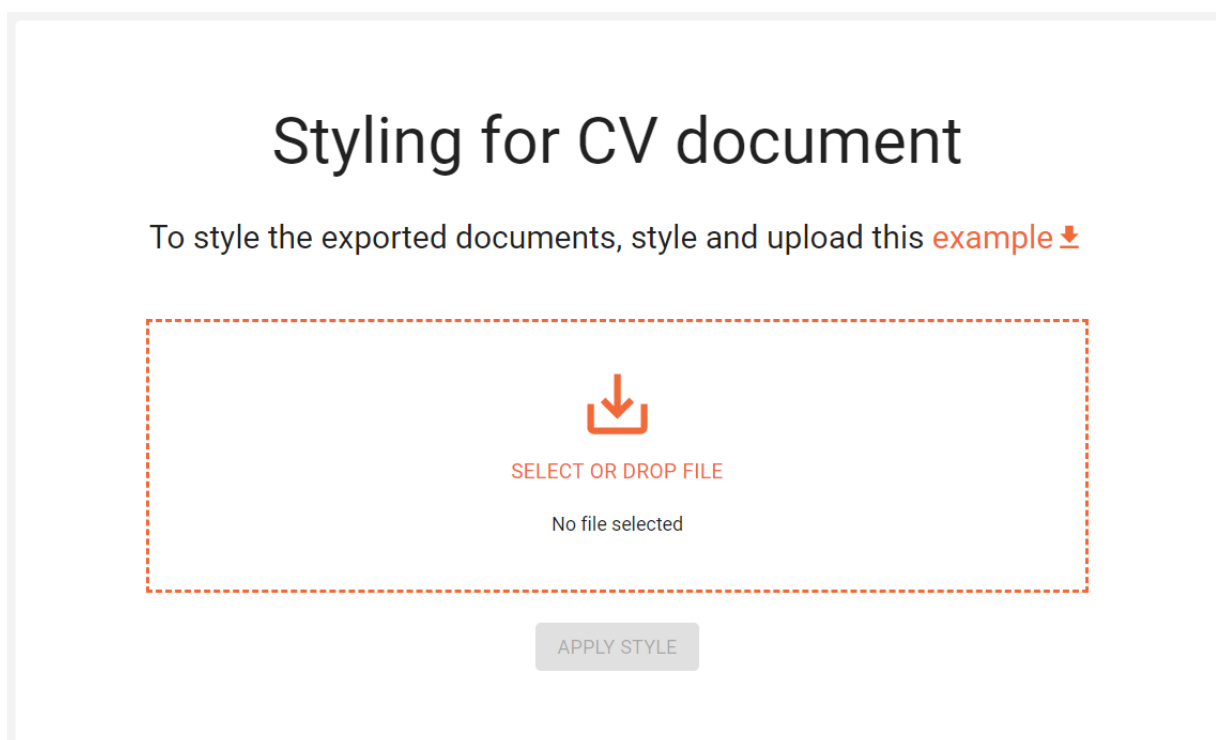


**Рис. 22** — Перетаскивание категории testing на вторую позицию

ее за определенной ролью.

Последняя вкладка меню отвечает за две вещи: выгрузку .docx файла, в котором содержится последняя конфигурация, описывающая все стили резюме word документа и загрузку файла с новой настроенной конфигурацией. Было решено снова написать собственный компонент и застилизовать анимацию при Drag-and-Drop с помощью LESS.

Таким образом автор дипломной работы с нуля написала всем знакомый интерфейс (рис. 23), когда документ можно загрузить, просто опуская его из директории в выделенную область или кликнув по ней, чтобы открылся проводник на компьютере. Была написана валидация на формат загружаемого документа – если пользователь пытается загрузить что-то, кроме .docx, то его предупреждает мой сервис оповещений.



**Рис. 23** — Интерфейс изменения конфигурации выгружаемого документа с резюме

## **4. Результаты решения задачи**

Проделанная работа была продемонстрирована и оценена работниками Ярославского офиса компании Akvelon, получила как теплые отзывы, так и пищу для размышлений для дальнейшего развития данного проекта.

Сейчас HRCV-Portal тестируется рекрутерами, а база данных пополняется сотрудниками компании.

В результате работы был получен колоссальный опыт в сфере front-end разработки и был задан вектор для понимания направления профессионального развития в будущем.

## **Заключение**

В ходе работы автором была создана клиентская часть веб-приложения для оптимизации заполнения резюме рекрутерами компании Akvelon.

Автор разобралась в поставленной перед ней задаче и выполнила проект, удовлетворяющий требованиям клиента в лице команды рекрутеров. Результатом работы стал ресурс, содержащий автоматически заполняемую базу необходимых для ускорения создания резюме данных. По совместительству работа является централизованным хранилищем всех резюме кандидатов в сети Интернет. Реализован весь интерфейс приложения на основе исследования пользовательского опыта и поведения. Создан сервис для взаимодействия с сервером. Внедрены react-компоненты для автозаполнения данных, проверка их корректности. Сделан интерфейс для информации о прогрессе получения информации для отображения и обработчики, отвечающие за взаимодействие пользователя с элементами страниц на сайте.

Разработки автора внедрены в систему рабочего процесса рекрутеров компании Akvelon.

## Список литературы

- [1] React - JavaScript-библиотека для создания пользовательских интерфейсов URL: <https://ru.reactjs.org/> (дата доступа: 10.06.2020)
- [2] Знакомство с JSX в React URL: <https://ru.reactjs.org/docs/introducing-jsx.html> (дата доступа: 10.06.2020)
- [3] Material UI - React компоненты для легкой и быстрой разработки веб приложений URL: <https://material-ui.com/> (дата доступа: 10.06.2020)
- [4] Less - CSS, который просто немного лучше URL: <http://lesscss.org/> (дата доступа: 10.06.2020)
- [5] MobX - простое масштабируемое управление состоянием URL: <https://mobx.js.org/README.html> (дата доступа: 10.06.2020)
- [6] JWT - это открытый промышленный стандарт RFC 7519 для безопасного представления заявок между двумя сторонами. URL: <https://jwt.io/> (дата доступа: 10.06.2020)
- [7] VSCode - редактирование кода бесплатно. Построен на открытом исходном коде. Работает везде. URL: <https://code.visualstudio.com/> (дата доступа: 10.06.2020)
- [8] reCAPTCHA - новый способ остановить ботов URL: <https://www.google.com/recaptcha/intro/v3.html> (дата доступа: 10.06.2020)