

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 24 с., 4 гл., 15 рис., 0 табл., 1 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>4</b>
<b>1. Теория</b>	<b>5</b>
<b>2. О задаче</b>	<b>6</b>
2.1. Постановка задачи . . . . .	6
2.2. Требуемый функционал . . . . .	6
2.3. Используемые программные средства . . . . .	7
<b>3. Решение задачи</b>	<b>8</b>
3.1. Создание базовой архитектуры . . . . .	8
3.2. Создание сервиса запросов, авторизация . . . . .	9
3.3. Регистрация, валидация и вход на сервис . . . . .	10
3.4. Сервис уведомлений пользователя, отслеживание прогресса загрузки	12
3.5. Страница создания резюме взглядом пользователя . . . . .	14
3.6. Таблица всех резюме взглядом пользователя . . . . .	19
3.7. Отладка умных контрактов . . . . .	20
3.8. Визуальный интерфейс для взаимодействия с развернутыми умными контрактами . . . . .	20
<b>4. Результаты решения задачи</b>	<b>22</b>
<b>Заключение</b>	<b>23</b>
<b>Список литературы</b>	<b>24</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. Теория**

## **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 Аквелона;
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 Аквелона, 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 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;
- Jest и Enzyme для написания unit-тестов.

## 3. Решение задачи

### 3.1. Создание базовой архитектуры

В компании мне 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 релизована 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 многократного 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 этой 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.
- `mui-theme.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

Первым делом предстояло как-то идентифицировать пользователей, для чего я создала 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, я 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

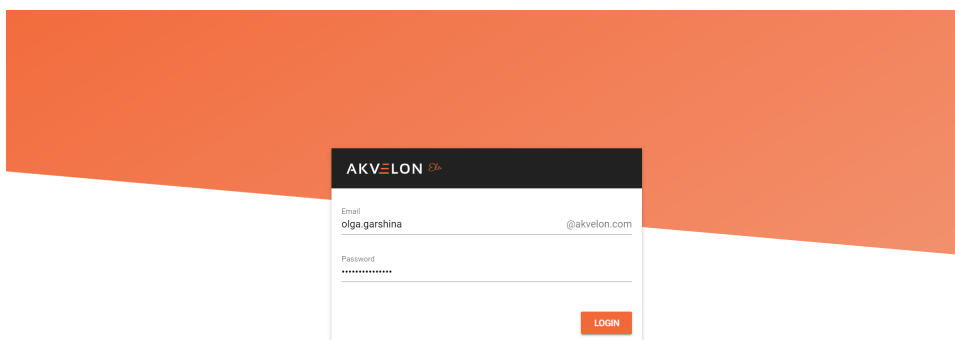
Клиент поставил условие – страницы, связанные с авторизацией должны быть выполнены в таком же стиле, что и сайт компании для оценки рабочего времени, написанный на Vue.js. Но должна быть возможность входа с любой почтой, а не с доменным именем.

По итогу было сделано максимально возможно похоже, но с помощью React, Less и Material-UI компонентов (рис. 2):

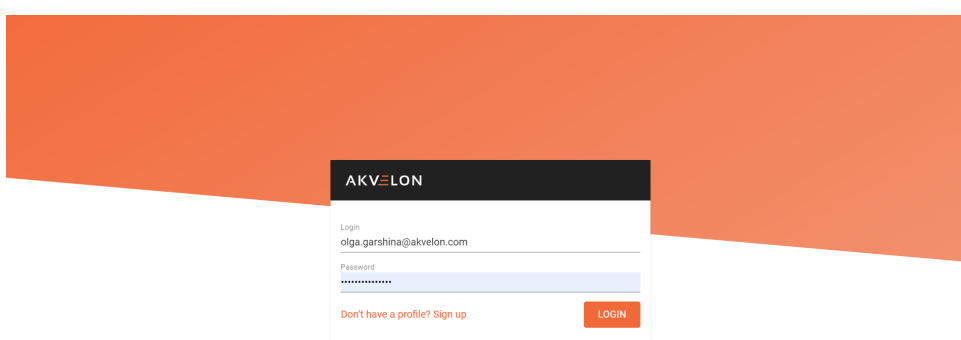
Следующий шаг - сделать валидацию на странице регистрации. Для этого я создала еще один сервис со следующей архитектурой.

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

В результате работы этих функций возвращается булевый результат и сообщение об ошибке, если он false. Если ошибки могут быть разного типа для одного



**Рис. 1** — Страница входа на ets.akvelon.com



**Рис. 2** — Страница входа на hrcv.inyar.ru

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 — система, разработанная в университете Карнеги — Меллон для защиты веб-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

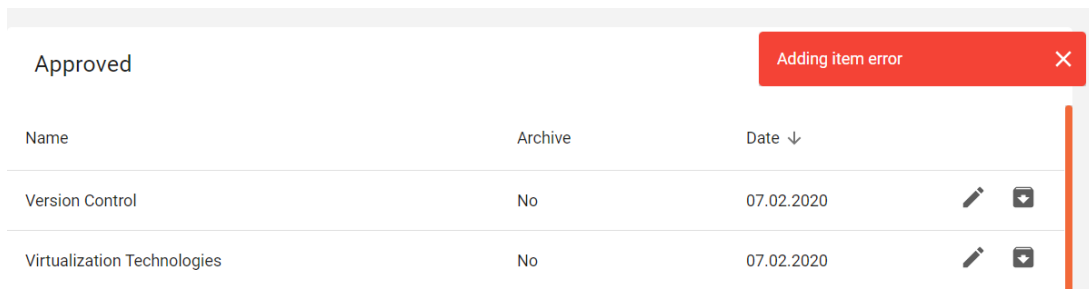
SIGN UP

Рис. 3 — Страница входа на ets.akvelon.com

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

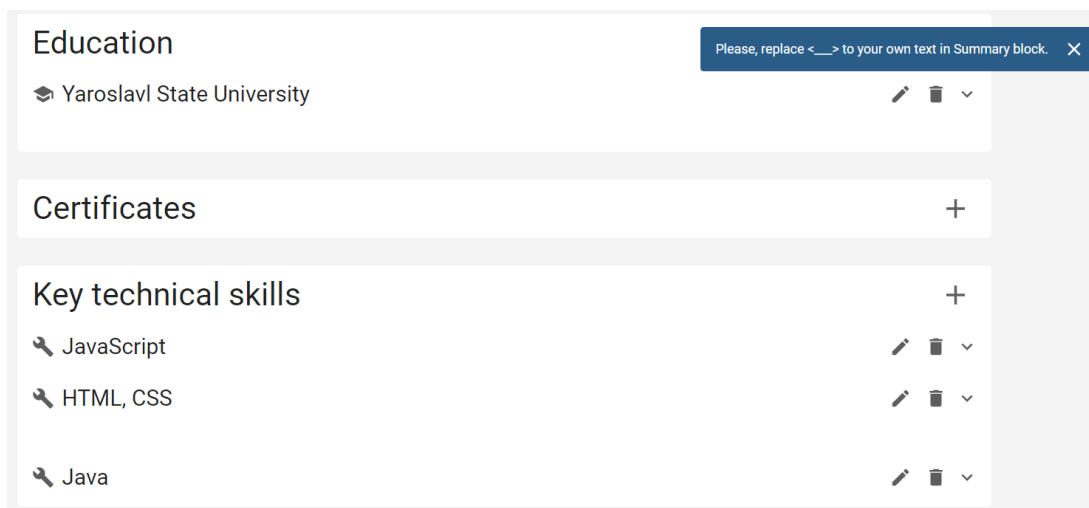
На данном этапе разработки понадобилось оповещать пользователя об ошибках со стороны сервера и другой полезной информации. Для этого я создала сервис toast.notify и ToastTrigger компонент.

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 соответствующее. На рисунке 4 отражен вызов функции в месте кода, где сервер вернул ошибку добавления новой категории.



**Рис. 4** — Пример уведомления об ошибке

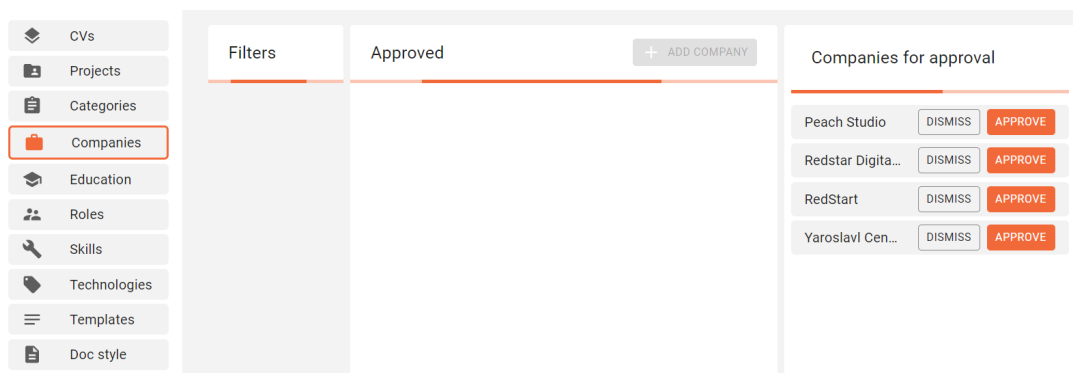
Я поделила оповещения на 4 типа: предупреждение, ошибка, информация и успех. С помощью Less очень удобно переопределять стиль плашки в зависимости от параметров, переданных в функцию. Вот, например, плашка с информацией, которая вызывается при попытке сохранения резюме, если пользователь забыл ввести обязательную информацию (рис. 5).



**Рис. 5** — 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. 6). 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. 6** — Интерфейс полос загрузки

### 3.5. Страница создания резюме взглядом пользователя

Создание резюме проходит в несколько этапов и создается для отправки клиенту Аквелона 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. 7) 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 мои нужды. Было потрачено немало времени, чтобы разобраться в работе библиотеки, убрать лишние детали и добавить нужные.

Поле роль - созданный мною компонент. Изначально, в Material UI не было достойного решения для выбора пункта из списка, с полем для поиска по нему. Мне предлагали только сторонние библиотеки, которые имели в себе много лишних зависимостей и много строк кода, которые неприятно читать и сложно в них разбираться. После чего наступил переломный момент и я решила просто написать свой компонент с тем поведением, который идеально мне подошел. Несмотря на то, что создание простого казалось бы списка со встроенным поиском выглядело простым, в процессе написания я столкнулась со многими подводными камнями, связанными с обработкой событий в React. Я искала ответы в интернет сообществах по программированию, но встречала только некрасивые решения.

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. 7** — Интерфейс заполнения поля Summary

В итоге мне удалось сделать компонент (рис. 8), 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

s

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

**Рис. 8** — Компонент для выбора из списка с поиском

После кандидат заполняет информацию об опыте работы в компаниях в блоке professional experience. Я вдохновлялась дизайном заполнения информации на сайте linkedin при разработке данного блока (рис. 9). На его примере я хочу показать функционал подсказок для автозаполнения и принцип пополнения базы данных.

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 карточки сводки о компании и о занимаемой позиции в ней.

## 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

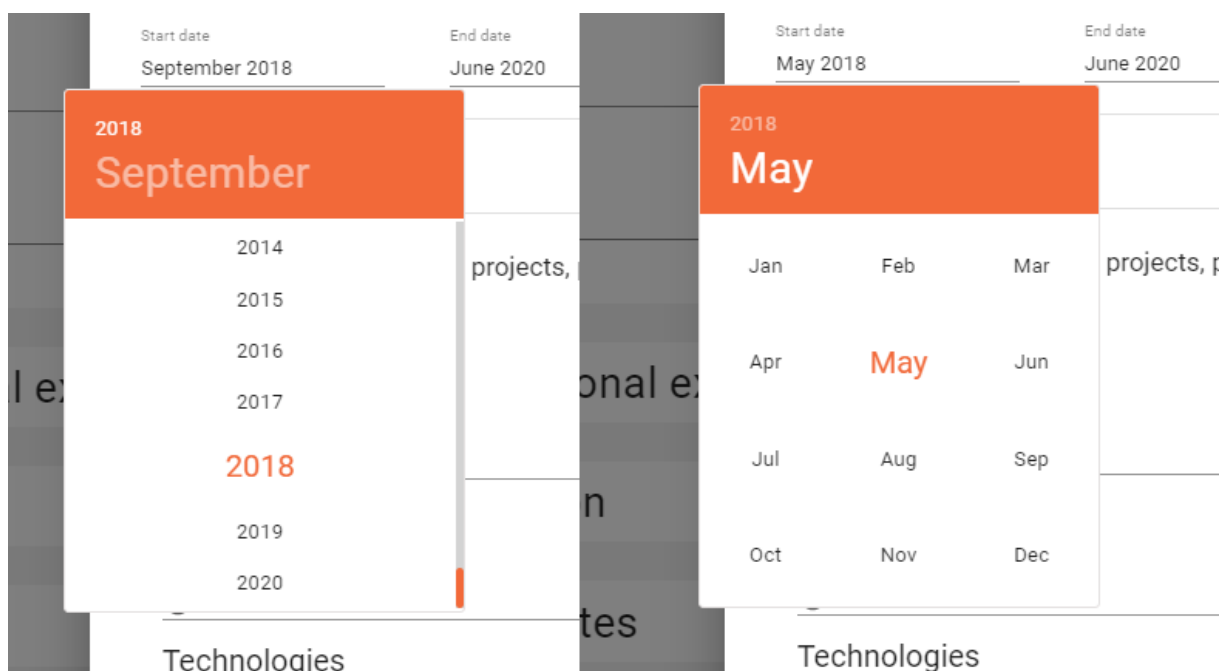
ADD COMPANY CANCEL

**Рис. 9** — Диалоговое окно добавления компании

После кандидату необходимо указать время работы в компании. При нажатии на поля start date/end date поочередно открывается интерфейс для выбора года и месяца (рис. 10). 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.

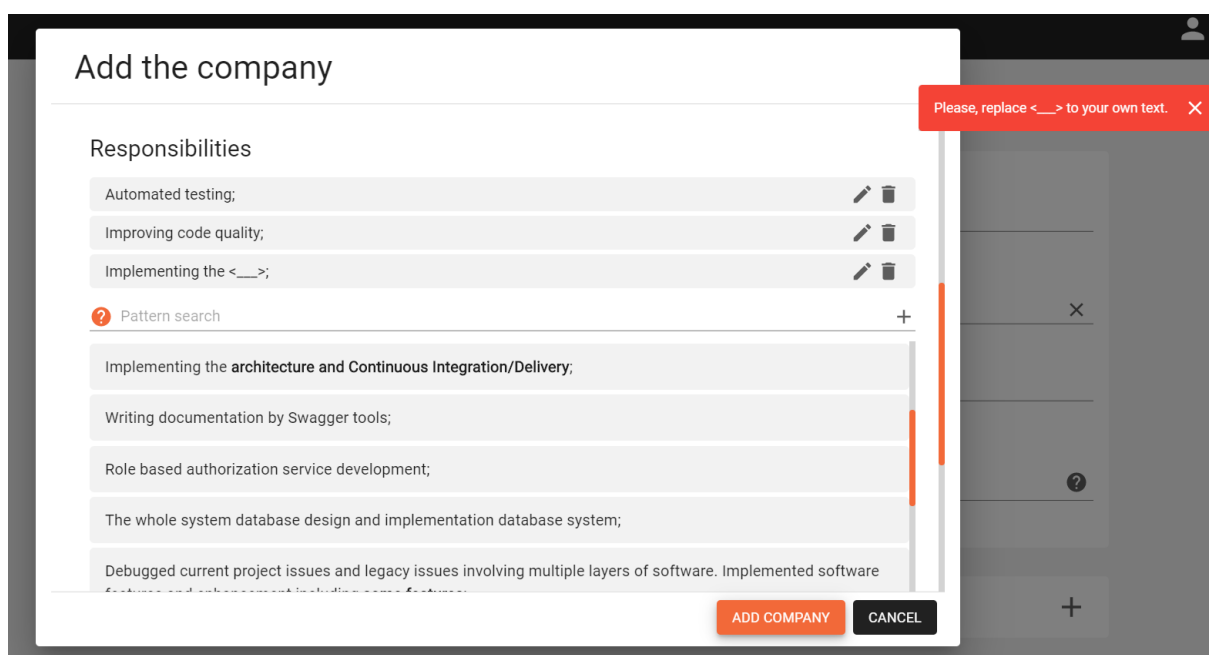
Потом подразумевается заполнение информации о проектах. Если кандидат не может выделить каких-то конкретных, то требуется заполнить общую информацию. Предлагается внести данные о персональных результатах, выполненных задачах и используемых технологиях. Некоторые люди сталкиваются с проблемой того, что они не могут сразу быстро вспомнить все свои достижения. Для этого я добавила возможность раскрыть по кнопке со знаком вопроса меню с подсказ-





**Рис. 10** — Интерфейс выбора даты

ками 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 11.

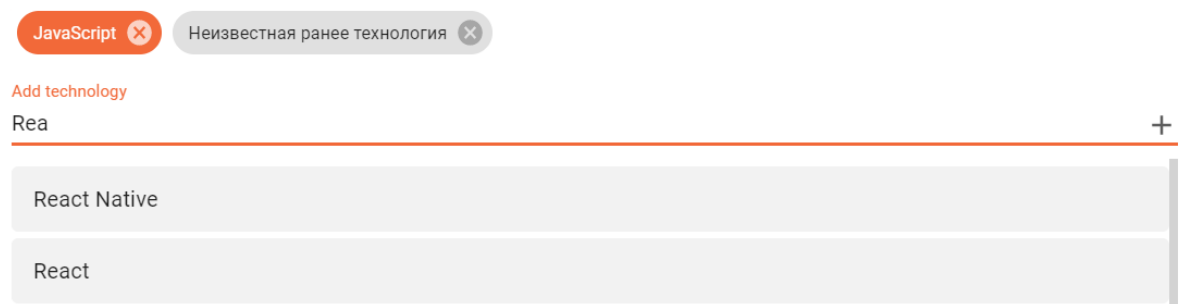


**Рис. 11** — Интерфейс заполнения информацией поля responsibility и её валидация

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, я сделала цветовой акцент на интерфейсе. Таким образом, одобренные технологии окрашены в оранжевый, а остальные в серый (рис. 12). Чтобы пользователь не засорял базу альтернативными названиями технологий было решено привязывать к ним список синонимов. Так, введя в поле JS и нажав ввод, кандидат получит корректное названия языка на выходе – JavaScript.

## Technologies



The screenshot shows a user interface for adding technologies. At the top, there's a header 'Technologies'. Below it, there are two buttons: 'JavaScript' (orange) and 'Неизвестная ранее технология' (grey). Underneath these buttons is a section titled 'Add technology' with a search bar. The search bar contains the text 'Rea'. Below the search bar, there are two suggestions: 'React Native' and 'React'.

**Рис. 12** — 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 сиви находится только один логический блок.

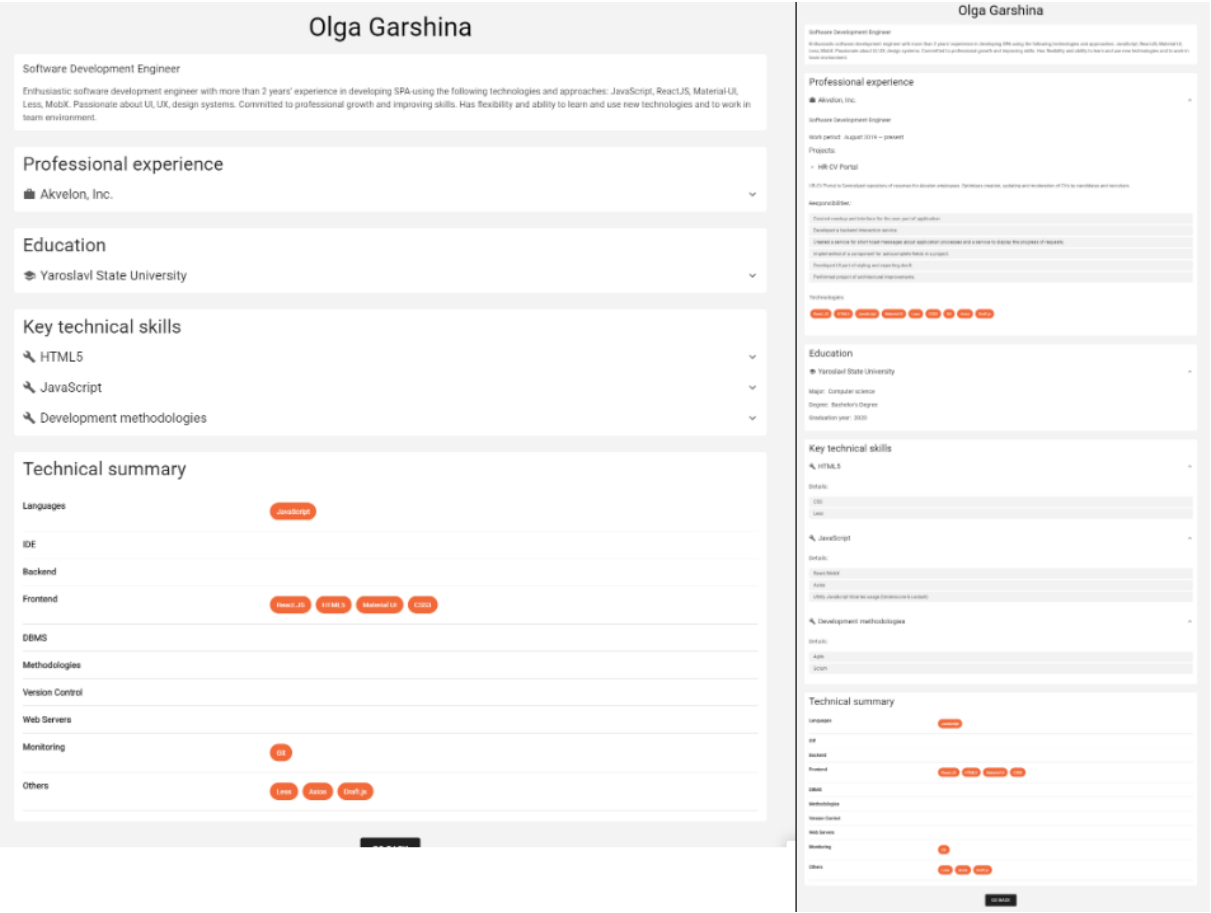
Преимущество данного UX-решения особенно хорошо наблюдается в режиме просмотра резюме. В нем можно развернуть всю структуру документа для полноты восприятия, страница приложения станет действительно длинной (рис. 13), особенно если речь идет о опыте работа программиста-сеньора. Также в данном режиме полностью отсутствуют отвлекающие кнопки для какого-либо взаимодействия, он отлично подходит для того, чтобы рекрутер мог просто поделиться 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 указываю в резюме на различных проектах и различных компаниях, написанный мною алгоритм валидирует хранение в данном списке только уникальных наименований технологий.

Также это то место, где кандидат может вбить изученные им вне работы

технологии. Иногда бывало, что 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. 14).



**Рис. 13** — 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. 15). 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 привелегия рекрутеров, которые подразумеваются



AKVELON

Filters

Position

Project

Role

Archive

No

From

06/01/2020

To

06/13/2020

List of your CVs

NEW CV

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	<div>Edit</div> <div>View</div> <div>Copy</div> <div>Archive</div> <div>Export .docx</div>

Rows per page: 10 1-4 of 4 < >

**Рис. 15** — Таблица со списком пользовательских резюме

#### **4. Результаты решения задачи**

В результате решения задачи было получено расширение для VSCode

## **Заключение**

## **Список литературы**

- [1] Visual Studio Code - Code Editing. Redefined URL: <https://code.visualstudio.com>  
(дата доступа: 09.06.2020)