Софиийски университет "Св. Климент Охридски"
Факултет по математика и информатика

Проект

по

Разработка на клиент-сървър (fullstack) приложения с Node.js + Express.js + React.js

спец. Информатика, 3 курс, летен семестър,

учебна година 2018/19

IVAS-TECH Online Ordering system

3 септември 2019 г.

Изготвил:

Иво Алексеев Стратев

Фак. номер: 45342

София

Съдържание

Списък на фигурите

1 Увод

Целта на проекта е да се разбработи приложение, което да работи в облачна среда. Разработваното приложение цели да модернизира процеса по поръчки от страна на клиентите на една българска компания и да улесни процеса по приемане и обработка на направените поръчки от страна на служителите на компанията. Дейността на фирмата е уникална за България и за това се налага проектирането и разработката на специфично приложение.

Компанията се занимава с изработката на лазерно рязани стенсили. Те се използват в процеса на монтиране на електорнни компоненти върху печатни платки.

Разработваното приложение трябва да осигурява възможност на клиентите на компанията да правят своите поръчки чрез интернет. Като за целта използват web browser-а на своя работен компютър.

2 Изисквания към разработваното приложение

2.1 Функционални изисквания предназначени за клиентите на компанията

Приложението трябва да осигурява възможност на ръководното лице на всяка компания да създаде виртуална организация съответстваща на компанията, която ръководи. В тази виртуална организация трябва да се съхранява информацията за всички потребители участващи в организацията, както и за всяка една поръчка направена от участник в организацията. Съответно трябва да се осигури възможостта за правенето на поръчки, което се състоѝ в попълването на информация за поръчвания стенсил и изпращането (прикачането) на архив от необходимите файлове за изработката. Всеки потребител на приложението участва само в една организация.

При създаването на организация се създава и потребителски профил на създателя на организацията. Всеки създател на организация има възможност да кани нови участници в организацията. За целта създателя на организацията тряба да има възможост за генериране на линк за ре-

гистрация на нов потребител. Този линк трябва да дава възможност за автоматичното попълване на адреса на електорнна поща на евентуалния нов участник на организацията. Когато бъде достъпен този линк от web browser трябва да се дава възможност за регистрация на нов потребител. Като при успешна регистрация потребителя става член на организацията, от която е бил създаен линка.

Приложението трябва да дава възможост за вход в системата. След успешен вход в системата на потребителя трябва да се зарежда информация за активните поръчки, ако такива има или да се съобщава, че такива няма. Прочъките, които са в състояние готови за доставка, трябва да са отделени от останлите и да са над тях. Съответно от списъка с готовите за доставка поръчки трябва да могат да се избират, тези които да бъдат изпратени. Тоест всяка поръчка за стенсил се прави индивидуално, но с една доставка могат да се доставят няколко поръчки.

Всеки стенсил се използва многократно, но след определен брой използвания той се изхабява. За това приложение трябва да дава възможност всяка поръчка, да бъде повторена, при което автоматично във формата за поръчка се попълва информацията от поръчката, която трябва да бъде повторена, но трябва да се предоставя възможността за редактирането на информацията, от формата или промяна на избрания архив от файлове.

Всеки клиентски потребител трябва да има възможност за преглед на всички поръчки в рамките на виртуалната организация. За целта трябва да се предоставя възможност за гъвкавко филтриране, на направените поръчки.

Всеки създадетел на организация трябва да има възможност да деактивира или замразява потребителски профил на участник в организацията, различен от себе си.

2.2 Функционални изисквания предназначени за служителите на компанията

Приложението трябва да осигурява възможност на служителите на компанията да преглеждат направените поръчки и да им помага в тяхната обработка. Като дава възможност за промяна на състоянието на една поръчка. Възможните състояния са: изчакваща обработка, одобрена, от-

хвърлена, отказана, изпълнява се, готова за доставка, изпратена и доставена. Съответно при промяна на статуса на отхвърлена приложението трябва да дава възможност за подробо описание на причината за отхвърлянето на поръчката.

Приложението трябва още да предоставя инфромация за направените заявки за доставка. Като предоставя информация за избраните поръчки и попълнената от клиента информация за доставка. Съотвено всеки служител трябва да има възможност да отбелязва, че дадена доставка е изпратена, при което състоянието на всяка поръчка трябва автоматично да се променя от готова за доставка на изпратена.

2.3 Функционални изисквания общи за всички потребители

Приложеието трябва да осигурява възможост за избор на език на всеки потребител независимо дали е регистриран или не е и независимо дали е влязал в системата или не е.

Приложението трябва дава осигурява достъп до вход в системата, на всеки регистриран потребител. Един потребител трябва да има възможност да влезе в системата от няколко устройства едновременно. При правенето на заявка за вход в системата трябва да се дава възможност за избор дали потребителя да бъде запомнен.
Приложението трябва още да дава възможност на всеки потребител на приложението да променя информацията за себе си, като адрес на електорнна поща, потребителско име, телефон за контакт и парола. След промяна на информацията за потребител при достъп до приложението от устройство различно от използваното за промяна на информацията, трябва да се показва съобщение, че информация за потребителския профил е била промена и той ще бъде автоматично изхвърлен от системата. При опит за продължаване на изпозване на приложението потребителя, трябва да бъде изхвърлен от системата и да бъде насочен към формата за вход в системата.

3 Използвани технологии и библиотеки

Приложението е разделено на две части: клиентска и сървърна. Клиентската част е разработена като Single page application (SPA). Сървърната

част е разработена като Service API и е разбработена с цел опериране от някой от публичните доставчици на облачни услуги.

3.1 Използвани технологии и библиотеки за сървърната част

За разработката на сървърната част като основна технология бе избрана мулти-платформената среда за изпълнение на сървърни и мрежови приложения Node.js [1].

За база данни бе избрана документната база данни [2] MongoDB [3].

За връзка с MongoDB бе избран официалния пакет mongodb [4] за използване от Node.js.

С цел верифициране на чувствителната информация изпращана от клиента към сървъра бе избрана технологията JSON Web Token (JWT) [5].

За използването на JWT технологията бе избрана Node.js библиотеката jsonwebtoken[6].

За кодирането и декодирането на base64url[7] кодирани символни низове бе израбна Node.js библиотеката base64url[8].

За хеширане на пароли бе избран алгоритъма bcrypt [9].

За използване на bcrypt алгоритъма бе избрана Node.js библиотеката bcrypt [10].

За валидация на данните бе избрана JavaScript библиотеката validator [11].

За генерирането на UUID v4 [12] символни низове бе избрана библиотеката uuid [13].

Като софтуерна рамка за разработка на сървърната част бе избран пакетния модул Express [14].

Като добавка към Express за оработка на тялото на получените заявки бяха избрани пакетите: body-parse[15] и multer[16].

3.2 Използвани технологии и библиотеки за клиентската част

JavaScript е доминиращия език [17] за разработка на приложения работещи в web browser. Въпреки, че съществуват технологии като WebAssembly [18], JavaScript все още продължава да е най-предпочитания език [19], до който се компилират други езици [20]. Това е така, защото JavaScript си има свойте проблеми [21]. За това за език на клиентската част бе избран езика TypeScript[22].

В днешно време голяма част от новите клиентски приложения, които се разработват като Single Page Application (SPA) са базирани на идеята за компоненти [23], която е една от основните идеи на библиотеката React [24]. React също така е доминантната технология за разработка на SPA [25].

За библиотека за User Interface (UI) (Потребителски интерфейс) компоненти бе избрана MATERIAL-UI [26], която имплементира така наречения Material Design [27], която е дизайн система, с React компоненти.

За менажиране на състоянието на приложението бе избрана библиотеката Redux [28]. Като под състояние се разбира цялостното състояние, в което се намира клиентското приложение. От това на коя от всички (виртуални) страници е, кои от компонентите на съответната страници са видими, кои от диалозите са отворени и кои не, какви са стойностите на полетата на всяка форма, какви са данните на приложението и тн.

За рутиране на вритуалните страници бе избрана библиотеката Redux-First Router [29].

За управление на страничните ефекти на приложението (като правенето на заявки до сървъра или използването на маханизми за съхранение на данни в web browser-а) бе избрана библиотеката Redux-Saga [30].

За по-лесно ползване на маханизми за съхранение на данни в web browser-а, известни под общото наименование (Offline) Browser Storage бе избрана библиотеката localForage [31].

4 Service API

4.1 Потребител

4.1.1 POST /api/user/organizationManager

Създава нова организация и потребителски профил на създателя.
Очаква тялото на заявката да съдържа:

- user (Потребителско име): Символен низ с минимална дължина поне 2 символа и максимална дължина 23.
- password (Парола): Символен низ с минимална дължина 8 символа, максимална дължина 32, състоящ се само от големи латински букви, малки латински букви, цифри и - и _, трябва да съдържа поне 1 главна латинска буква, поне 1 малка латинска буква и поне 1 цифра.
- confirmPassword (Потвърди парола): Символен низ, чиято стойност трябва да съвпада със стойността на password.
- email (Адрес на електронна поща): Символен низ с максимална дължина 255 символа, трябва да представлява валиден адрес на електорнна поща.
- phone (Телефон): Символен низ с максимална дължина 19 символа, трябва да предоставя валиден телефоннен номер.
- organization (Име на създаваната организация): Символен низ с минимална дължина 3 символа и максимална дължина 99 символа.

Проверява дали съществува организация с име съвпадащо със стойността на полето organization, ако съществува връща информация за това, ако не съществува организация проверява дали съществува потребител с адрес на електорнна поща съвпадащ със стойността на полето email. Ако съществува връща информация за това, в противен случай създава нова организация с подаденото име с управител ново създадеия потребителски профил, с роля управител на организация и организация създадената организация. Ако се случи грешка при създаване на документите в базата се връща информация за това. Ако не е възникнала грешка се връща информация, че е създаден успешно профил на управител на организация.

4.1.2 POST /api/user/register/:organization

Създава нов потребителски профил в указаната организация.
Очаква тялото на заявката да съдържа:

- user (Потребителско име): Символен низ с минимална дължина поне 2 символа и максимална дължина 23.
- password (Парола): Символен низ с минимална дължина 8 символа, максимална дължина 32, състоящ се само от големи латински букви, малки латински букви, цифри и - и _, трябва да съдържа поне 1 главна латинска буква, поне 1 малка латинска буква и поне 1 цифра.
- confirmPassword (Потвърди парола): Символен низ, чиято стойност трябва да съвпада със стойността на password.
- email (Адрес на електронна поща): Символен низ с максимална дължина 255 символа, трябва да представлява валиден адрес на електорнна поща.
- phone (Телефон): Символен низ с максимална дължина 19 символа, трябва да предоставя валиден телефоннен номер.

Прави опит да декодира JWT от параметъра organization. Ако успее прави опит да верифицира, че декодираното JWT съдържа поле id, което е идентификатор на съществуваща организация. Ако всичко е било успешно до тук, се проверява дали съществува потребител в организацията с име - стойността на полето user от тялото на заявката. Ако съществува се връща информация за това. Ако не съществува създава нов потребителски профил с организация - организацията с идентификатор декодираното id и роля а обикновен потребител. Ако успешно създаде потребителски документ в базата връща информация, че успешно е създаде потребителски профил. Ако на някоя стъпка е прихваната грешка, то се връща информация за това.

4.1.3 Хеширане на парола и запазване на контролна сума

При горните 2 заявки сървърната част на приложението генерира сол, след което хешира паролата с алгоритъма bcrypt. Следва изчисление на контролна сума (checksum) на данните от тялото на заявката, към които се добавят идентификатор на потребителя, идентификатор на организацията и символен низ, чиято стойност само сървъра знае. При

записването на данните в базата запазват данните от тялото на заявката без password и confirmPassword, към които биват добавени двата идентификатора, генерираната сол и хеша на паролата.

4.1.4 POST /api/user/login

Дава възможост за вход в системата.
Очаква тялото на заявката да съдържа:

- email (Адрес на електронна поща): Символен низ с максимална дължина 255 символа, трябва да представлява валиден адрес на електорнна поща.
- password (Парола): Символен низ с минимална дължина 8 символа, максимална дължина 32, състоящ се само от големи латински букви, малки латински букви, цифри и - и _, трябва да съдържа поне 1 главна латинска буква, поне 1 малка латинска буква и поне 1 цифра.

Прави се опит за извличане на потребител от базата данни с адрес на електорнна поща - стойността на полето email. Ако бъде намерен се проверява дали хеша на паролата, би съвпаднал с евентуално хеширане на стойността password от тялото на заявката, при използването на случайна сол с дължина - дължината на солта използвана за генериране на хеша на паролата, запазен в базата данни. Ако това е така се създава JWT, съдържаш идентификатора на потребителя, името на потребителя, адреса на електорнната поща и контролната сума. След което създадения JWT заедно с потребителското име, извлечно от базата данни, се връщат като отговор на заявката. Ако на някой етап бъде прихванатa грешка, то се връща информация за грешакта.

4.1.5 PUT /api/user/name

Променя потребителското име на потребителя.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да съдържа:

- userName (Потребителско име): Символен низ с минимална дължина поне 2 символа и максимална дължина 23.
- password (Парола): Валидна парола.

Прави опит да промени потребителското име на потребителя, съответно преизчислява контролната сума и я добавя към заявката за промяна към базата. При успех връща нов JWT за аутотекиране. При неуспех връща информация за настъпилата грешка.

4.1.6 PUT /api/user/email

Променя адреса на електорнна поща на потребителя.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да съдържа:

- email (Адрес на електронна поща): Символен низ с максимална дължина 255 символа, трябва да представлява валиден
- password (Парола): Валидна парола.

Прави опит да промени адреса на електорнна поща на потребителя, съответно преизчислява контролната сума и я добавя към заявката за промяна към базата. При успех връща нов JWT за аутотекиране. При неуспех връща информация за настъпилата грешка.

4.1.7 PUT /api/user/phone

Променя телефония номер на потребителя.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да съдържа:

- phone (Телефон): Символен низ с максимална дължина 19 символа, трябва да предоставя валиден телефоннен номер.
- password (Парола): Валидна парола.

Прави опит да промени телефонния номер на потребителя, съответно преизчислява контролната сума и я добавя към заявката за промяна към базата. При успех връща нов JWT за аутотекиране. При неуспех връща информация за настъпилата грешка.

4.1.8 PUT /api/user/password

Променя паролата на потребителя.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да съдържа:

- currentPassword (Текуща парола): Валидна парола.
- newPassword (Нова парола): Валидна парола.

Прави опит да промени паролата на потребителя, като генерира нова сол и хешира новата парола. Съответно преизчислява контролната сума и я добавя към заявката за промяна към базата. При успех връща нов JWT за аутотекиране. При неуспех връща информация за настъпилата грешка.

4.2 Организация

4.2.1 GET /api/organization/members

Връща списък от идентификаторите и имената на всички потребители, членове на организацията на потребителя.
Очаква потребителя да бъде аутотекиран.
Прави опит да извлече от базата данни идентификатора и потребителското име на членовете на организацията на потребителя. При успех връща JSON обкет с единствено поле members, което съдържа извлечената от базата данни информаци във вид на масив. При неуспех връща информация за настъпилата грешка.

4.2.2 PUT /api/organization/name

Променя името на организацията.
Очаква потребителя да бъде аутотекиран и да бъде създадетеля на организацията.
Очаква тялото на заявката да съдържа:

- newOrganizationName (Ново име на организацията): Символен низ с минимална дължина 3 символа и максимална дължина 99 символа.
- password (Парола): Валидна парола.

Прави опит да промени името на организацията. При успех връща информация, че името е променено. При неуспех връща информация за настъпилата грешка.

4.2.3 GET /api/organization/link

Връща линк, който може да бъде използван за създаване на потребителски профил в организацията.
Очаква потребителя да бъде аутотекиран и да бъде създадетеля на организацията.
Връща информация за линк за добавя на нов чле на организацията.

4.3 POST /api/order

Прави нова поръчка.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да е във формат multipart/form-data.
Очаква тялото да съдържа само 3 полета.

- password (Парола): Валидна парола.
- archive (Файлов архив): Файлов архив състоящ се от един файл с разширение измежду: .zip, .rar, .tar.gz и .tar.
- data (Данни): Символен низ, който представлява валиден JSON обект с информация за изработка на поръчката.

Запазва прикачения файл, в резултат на което се получава връзка за теглене на файла. Прави запис в бата данни за нова поръчка с данни от data, връзката за теглене, административна информация за файла, дата на почъване, текущата дата, идентификатор на потребителя и на организацията. При успех връща информация, че успешно е направена нова поръчка. При прихващането на грешка изпраща информация за грешката.

4.4 GET /api/active-orders

Връща списък от активните поръчки на организацията.
Очаква потребителя да бъде аутотекиран.
Прави опит да извлече активните поръчки от базата данни. При успех връща JSON с единсвено поле activeOrders, което съдържа списък с информация за активните поръчки. При неуспех се връща информация за настъпилата грешка.

4.5 GET /api/order/:id

Връща пълна информация за поръчка с идентификатор id.
Очаква потребителя да бъде аутотекиран.
Прави опит да извлече данните за поръчка с идентификатор id от базата данни. Проверява дали поръчката е направена в организацията на потребителя. Ако не е връща информация за това. Ако е връща извлечената информация.

4.6 GET /api/file/:id

Връща като отговор поискания файл.
Очаква потребителя да бъде аутотекиран.

4.7 POST /api/order/query/:page-size

Връща списък с елементи, всеки от които представлява не пълна информация за поръчките отговарящи на заявката и брой на поръчките отговарящи на заявката.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да съдържа:

- start (Начална дата): ISO Timestamp.
- end (Крайна дата): ISO Timestamp.
- status (Статус): масив от валидни състояния на поръчка.
- orderedBy (Поръчана от): масив от валидни идентификатори на потребители.
- fileExtention (Разширение на файла): масив от разширения.
- fileName (Име на файла): Символен низ представляваш маска за името на файла.

Прави една заявка за намиране броя на поръчките, направени в организацията отговарящи на заявката.
Прави втора заявка към базата данни за извличане на поръчки, направени в организацията отговарящи на заявката, като ги сортира по дата на поръчване и лимитира броя до числото, стойност на page-size параметъра.
При прихващане на грешка връща информаци за настъпилата грешка. Ако нито една грешка не е прихваната връща информация за намерения брой и извлечените заявки.

4.8 POST /api/order/query/:page-size/:page

Връща списък с елементи, всеки от които представлява не пълна информация за поръчките отговарящи на заявката, от страниця с номер - стойността на page, ако е положително цяло число.
Очаква потребителя да бъде аутотекиран.
Очаква тялото на заявката да съдържа:

- start (Начална дата): ISO Timestamp.
- end (Крайна дата): ISO Timestamp.
- status (Статус): масив от валидни състояния на поръчка.
- orderedBy (Поръчана от): масив от валидни идентификатори на потребители.
- fileExtention (Разширение на файла): масив от разширения.
- fileName (Име на файла): Символен низ представляваш маска за името на файла.

Прави заявка към базата данни за извличане на поръчки, направени в организацията отговарящи на заявката, като ги сортира по дата на поръчване, прескача page-size * (page - 1), и лимитира броя до числото, стойност на page-size параметъра.
При прихваще на грешка връща информаци за настъпилата грешка. Ако нито една грешка не е прихваната връща извлечените заявки.

4.9 Аутотекиране

Процеса по аутотекиране на потребител на приложеието е следния.
При правена на заявка към сървъра, която изисква потребителя да бъде аутотекиран трябва да бъде попълнено полето Authorization на header-ите на заявката със "Bearer "+ JWT жетона получен от сървъра при вход в системата.
При получаване на заявка сървъра прави лека валидация на стойността на Authorization header-a. Ако тя е успешна се пробва да декодира JWT жетона и да установи, че е бил издаден от сървъра (да го верифицира). Жетона трябва да съдържа информация за идентификатора на потребителя и контролната сума. Извършва се заявка за извличане на потребител с идентификатор, идентификатора от JWT жетона. Ако бъде намерен се проверява дали контролната сума на документа, извлечен от

базата данни и контролната сума от декодирания JWT жетон съвпадат. Ако не съвпадат или на някоя стъпка е била прихваната грешка се връща като отговор следния JSON { "error": { "accessDenied": true } }. Ако съвпадат към заявката се добавя извлечения документ и тя продължава напред.

Литература

[1] Node.js, About Node.js,

https://nodejs.org/en/about/

[2] Amazon AWS, What Is a Document Database?,

https://aws.amazon.com/nosql/document/

[3] MongoDB, What Is MongoDB?,

https://www.mongodb.com/what-is-mongodb

[4] MongoDB, MongoDB Node.js Driver,

http://mongodb.github.io/node-mongodb-native/

[5] Auth0, Introduction to JSON Web Tokens,

https://jwt.io/introduction/

[6] Auth0, Github project page, node-jsonwebtoken,

https://github.com/auth0/node-jsonwebtoken#readme

[7] Web Security Information, Base64 encoding vs Base64url encoding, BalaKishore Gaddam, June 07 2017,

http://websecurityinfo.blogspot.com/2017/06/base64-encoding-vs-base64url-encoding.html

[8] Github project page, brianloveswords/base64url,

https://github.com/brianloveswords/base64url#readme

[9] Coda Hale, How To Safely Store A Password, 31 Jan 2010

https://codahale.com/how-to-safely-store-a-password/

[10] Github project page, kelektiv/node.bcrypt.js,

https://github.com/kelektiv/node.bcrypt.js#readme

[11] Github project page, validatorjs/validator.js

https://github.com/validatorjs/validator.js#readme

[12] International Spectrum, IS.UUID.V4: UUID V4 Random Generation, Nathan Rector, May/Jun 2017 Magazine

https://www.intl-spectrum.com/article/r848/

[13] Github project page, kelektiv/node-uuid

https://github.com/kelektiv/node-uuid#readme

[14] Express, Express home page,

https://expressjs.com/

[15] Github project page, expressjs/body-parser

https://github.com/expressjs/body-parser#readme

[16] Github project page, expressjs/multer

https://github.com/expressjs/multer#readme

[17] Lform's blog, How JavaScript Became the Dominant Language of the Web, Charles Punchatz, August 7 2017,

https://www.lform.com/blog/post/how-javascript-became-the-dominant-language-c

[18] WebAssembly, https://webassembly.org/

[19] DEV.to, Will WebAssembly replace JavaScript? Or Will WASM Make JavaScript More Valuable in Future?, Vaibhav Shah, Nov 3 2018,

https://dev.to/vaibhavshah/will-webassembly-replace-javascript-or-will-wasm-m

[20] List of languages that compile to JavaScript and many other transpilers, https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS

[21] JAVASCRIPT THE REAL BAD PARTS, @johnkpaul johnkpaul.com/empirejs,

https://johnkpaul.github.io/presentations/empirejs/javascript-bad-parts

[22] TypeScript, https://www.typescriptlang.org/index.html

[23] Medium.com, Understanding Component-Based Architecture, dshaps, Jun 16, 2016,

https://medium.com/@dan.shapiro1210/understanding-component-based-architecture-3ff48ec0c238

[24] React, https://reactjs.org/

[25] Medium.com, Ranking of the best front-end frameworks in 2019, Radoslaw Fabisiak, Mar 10 2019,
https://medium.com/@fabisiakradoslaw/ranking-of-the-best-front-end-frameworks-in-2019-777ec90c8884

[26] MATERIAL-UI, https://material-ui.com/

[27] MATERIAL DESIGN, https://material.io/

[28] Redux, https://redux.js.org/

[29] Redux-First Router, https://github.com/faceyspacey/redux-first-router

[30] Redux-Saga, https://redux-saga.js.org/

[31] localForage, https://localforage.github.io/localForage/