МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра програмних систем і технологій

ЗВІТ

про проходження технологічної практики в умовах АЦСК АТ КБ «ПриватБанк»

Розробка системи аутентифікації та обміну даних в реальному часі

Виконав: ст. гр. ІПЗ-33 Ващенко С.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівники практики:

Керівник АЦСК Железняк П.Л.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

від кафедри Ткаченко М.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Київ – 2020**РЕФЕРАТ**

Пояснювальна записка до звіту про проходження технологічної практики в

умовах АЦСК АТ КБ «ПриватБанк»:22 сторінок, 1 рисунок, 9

використаних джерел.

**Мета індивідуального завдання**: Мета роботи полягає у розробці системи в якій буде впроваджено аутентифікація користувачів використовуючи соціальні мережі, та розробка автономної аутентифікації в систему. Використання обміну даними в реальному часі.

**Використані методи**: В процесі розробки системи було використано

* Мови JS, NodeJS
* Фреймворки Express для серверної частини
  + Bcrypts JS
  + Cloudinary
  + Google Api
  + JSONwebtoken
  + Mongoose
  + Passport
  + Socket.io
* Фреймворк React для клієнтської частини
  + Redux
  + Socket.io - client
  + Material ui
  + Eslint
  + Stylelint

**Отримані результати**: Було розроблено систему, що підтримує аутентифікація користувачів використовуючи сервіси Google, Facebook, Github та автономну аутентифікацію. Використовуючи Socket.io було розроблено віртуальне місце зустрічі в Інтернеті. Виконано

Зміст

[Вступ 5](#_Toc32708173)

[o Відомості про підприємство 5](#_Toc32708174)

[o Обґрунтування вибору теми індивідуального завдання 6](#_Toc32708175)

[o Опис вирішення завдання 6](#_Toc32708176)

[Основна частина 7](#_Toc32708177)

[Теоретична частина 7](#_Toc32708178)

[ Клієнтська частина 7](#_Toc32708179)

[o Основи JS 7](#_Toc32708180)

[o Основи React 8](#_Toc32708181)

[o Основи Material UI 11](#_Toc32708182)

[o Основи EsLint 11](#_Toc32708183)

[o Основи StyleLint 11](#_Toc32708184)

[ Серверна частина 12](#_Toc32708185)

[o Основи NodeJS 12](#_Toc32708186)

[o Основи Express 12](#_Toc32708187)

[o Основи Passport 13](#_Toc32708188)

[o Основи BcryptJS 13](#_Toc32708189)

[o Основи JSONWebToken 14](#_Toc32708190)

[o Основи Cloudinary 15](#_Toc32708191)

[o Основи Mongoose 15](#_Toc32708192)

[o Основи Socket.IO 16](#_Toc32708193)

[Практична частина 17](#_Toc32708194)

[1. Створюємо клієнтську частину 17](#_Toc32708195)

[2. Створюємо серверну частину 18](#_Toc32708196)

[3. Деплой системи на хостинг 19](#_Toc32708197)

[Висновки 21](#_Toc32708198)

[Список використаних джерел 22](#_Toc32708199)

# Вступ

## Відомості про підприємство

ПриватБанк — найбільший за розмірами активів [український банк](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%BD%D0%BA%D0%B8_%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B8) і лідер [роздрібного банківського ринку України](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%BD%D0%BA%D0%B8_%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B8#10_%D0%BD%D0%B0%D0%B9%D0%B1%D1%96%D0%BB%D1%8C%D1%88%D0%B8%D1%85_%D0%B1%D0%B0%D0%BD%D0%BA%D1%96%D0%B2_(%D0%B7%D0%B0_%D0%BA%D1%96%D0%BB%D1%8C%D0%BA%D1%96%D1%81%D1%82%D1%8E_%D0%BF%D0%BB%D0%B0%D1%82%D1%96%D0%B6%D0%BD%D0%B8%D1%85_%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D0%BA)), зареєстрований [19 березня](https://uk.wikipedia.org/wiki/19_%D0%B1%D0%B5%D1%80%D0%B5%D0%B7%D0%BD%D1%8F) [1992](https://uk.wikipedia.org/wiki/1992) року. Ініціатором створення банку, його першим головою правління був [Сергій Тігіпко](https://uk.wikipedia.org/wiki/%D0%A2%D1%96%D0%B3%D1%96%D0%BF%D0%BA%D0%BE_%D0%A1%D0%B5%D1%80%D0%B3%D1%96%D0%B9_%D0%9B%D0%B5%D0%BE%D0%BD%D1%96%D0%B4%D0%BE%D0%B2%D0%B8%D1%87).

Акредитований центр сертифікації ключів (АЦСК) — [акредитований](https://uk.wikipedia.org/wiki/%D0%90%D0%BA%D1%80%D0%B5%D0%B4%D0%B8%D1%82%D0%B0%D1%86%D1%96%D1%8F) державою у встановленому порядку орган, що надає послуги з надання [ЕЦП](https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D0%BF%D1%96%D0%B4%D0%BF%D0%B8%D1%81) з одночасним постачанням захищених носіїв ключової інформації або з використанням власних носіїв заявників типу [USB-флеш-накопичувач](https://uk.wikipedia.org/wiki/USB-%D1%84%D0%BB%D0%B5%D1%88-%D0%BD%D0%B0%D0%BA%D0%BE%D0%BF%D0%B8%D1%87%D1%83%D0%B2%D0%B0%D1%87).

Акредитацію сертифікаційних центрів в Україні здійснює Центральний засвідчувальний орган.

Акредитований центр сертифікації ключів має право:

* надавати послуги електронного цифрового підпису та обслуговувати виключно посилені сертифікати ключів;
* отримувати та перевіряти інформацію, необхідну для реєстрації підписувача і формування посиленого сертифіката ключа, безпосередньо у юридичної або фізичної особи чи її представника.

Акредитований центр сертифікації ключів має виконувати усі зобов'язання та вимоги, встановлені законодавством для центра сертифікації ключів, та додатково зобов'язаний використовувати для надання послуг електронного цифрового підпису надійні засоби електронного цифрового підпису.

Порядок акредитації та вимоги, яким повинен відповідати акредитований центр сертифікації ключів, встановлюються [Кабінетом Міністрів України](https://uk.wikipedia.org/wiki/%D0%9A%D0%B0%D0%B1%D1%96%D0%BD%D0%B5%D1%82_%D0%9C%D1%96%D0%BD%D1%96%D1%81%D1%82%D1%80%D1%96%D0%B2_%D0%A3%D0%BA%D1%80%D0%B0%D1%97%D0%BD%D0%B8).

## Обґрунтування вибору теми індивідуального завдання

Тема «Розробка системи аутентифікації та обміну даних в реальному часі» дана для надання навичок роботи з бібліотеками безпеки та вивчення базового розуміння структури клієнт-серверного додатка

## Опис вирішення завдання

Для було використано популярні мови та бібліотеки, а саме:

* [Клієнтська частина](#_Toc32706132) 
  + [JS](#_Toc32706133)
  + [React](#_Toc32706134)
  + [Material UI](#_Toc32706135)
  + [EsLint](#_Toc32706136)
  + [StyleLint](#_Toc32706137)
* [Серверна частина](#_Toc32706138) 
  + [NodeJS](#_Toc32706139)
  + [Express](#_Toc32706140)
  + [Passport](#_Toc32706141)
  + [BcryptJS](#_Toc32706142)
  + [JSONWebToken](#_Toc32706143)
  + [Cloudinary](#_Toc32706144)
  + [Mongoose](#_Toc32706145)
  + [Socket.IO](#_Toc32706146)

# Основна частина

## Теоретична частина

### Клієнтська частина

#### Основи JS

JavaScript це крос-платформний, об'єктно-орієнтована, скриптова мова. JavaScript - це маленька і легка мова; він спроектований не для створення закінчених, повноцінних додатків, а для легкого вбудовування в інші продукти і додатки, наприклад, в веб браузери. Усередині основного додатки, JavaScript може з'єднуватися з об'єктами цього додатка і здійснювати програмний контроль над ними.

Ядро JavaScript містить набір базових об'єктів, наприклад Array, Date і Math, і набір елементів мови: операторів, які керують структурою, і виразами. Ядро JavaScript може бути розширено для різних цілей за допомогою доповнень, наприклад:

1. Client-side JavaScript розширює ядро ​​мови надаючи об'єкти управління браузером (Navigator або інший веб-браузер) і його Document Object Model (DOM). Наприклад, клієнтські розширення дозволяють додатком розміщувати елементи в HTML-формі і реагувати на призначені для користувача події, такі як клацання миші, введення даних в форму і навігація по сторінках.
2. Server-side JavaScript розширює ядро ​​мови надаючи об'єкти для запуску JavaScript на сервері. Наприклад, server-side розширення дозволяють додатком з'єднуватися з реляційною базою даних, забезпечувати безперервність інформації між викликами додатки, або виконувати операції над файлами на сервері.

#### Основи React

React - це декларативна, ефективна і гнучка JavaScript бібліотека для створення користувацьких інтерфейсів. Вона дозволяє вам збирати складний UI з маленьких ізольованих шматочків коду, званих «компонентами».

Заснований на компонентах Створюйте інкапсульовані компоненти, які керують власним станом, а з них будуйте складні інтерфейси. Оскільки логіка компонентів написана на JavaScript, замість шаблонів, ви з легкістю можете передавати складні дані у вашому додатку і зберігати стан окремо від DOM.

1. Компонент зі станом

У доповнення до прийняття вхідних даних (доступ до яких здійснюється через this.props), компонент може зберігати дані внутрішнього стану (доступні через this.state). Коли дані стану компонента змінюються, відбувається повторне відрендерення розмітки з використанням функції render().

*import* React, { Component } from 'react';  
*import* { Route } *from* 'react-router-dom';  
*import* { connect } *from* 'react-redux';  
*import* AccessDenied *from* '../AccessDenied/AccessDenied';  
  
*class* PrivateRoute *extends* Component {  
 render() {  
 *const* { component: Component, ...rest } = *this*.props;  
 *const* { isAuthorization } = *this*.props.authorization;  
 *return* (  
 <Route  
 {...rest}  
 *render*={props => {  
 *return* isAuthorization === *true* ? <Component {...props} /> : <AccessDenied />;  
 }}  
 />  
 );  
 }  
}  
  
*function* mapStateToProps(state) {  
 *return* { authorization: state.authorization };  
}  
  
*export default* connect(mapStateToProps, *null*)(PrivateRoute);

1. Застосування та використання зовнішніх модулів

Використовуючи пропси і стан, ми можемо створити невеликий додаток для складання списку справ. Цей приклад використовує стан для відстеження поточного списку елементів, а також тексту, введеного користувачем. Хоча обробники подій здаються вбудованими, вони будуть зібрані та реалізовані за допомогою методу делегування подій. React дозволяє взаємодіяти з іншими бібліотеками та фреймворками.

*import* React, { Component } *from* 'react';  
*import* Box *from* '@material-ui/core/Box';  
*import* Button *from* '@material-ui/core/Button';  
*import* Grid *from* '@material-ui/core/Grid';  
*import* Typography *from* '@material-ui/core/Typography';  
  
*import* AuthorizationAPI *from* '../../../services/AuthorizationAPI';  
  
*class* ForgotPassword *extends* Component {  
 constructor(props) {  
 *super*(props);  
  
 *this*.state = {  
 formData: {  
 email: ''  
 },  
 send: *false*,  
 response: ''  
 };  
 }  
 *handleChange* = event => {  
 *const* { formData, send } = *this*.state;  
 formData[event.target.name] = event.target.value;  
 *this*.setState({ formData });  
 *if* (send === *true*) {  
 *this*.setState(prevState => ({  
 send: !prevState.send,  
 response: ''  
 }));  
 }  
 };  
 render() {  
 *const* { formData, send, response } = *this*.state;  
 *const* { toggleForgotPassword } = *this*.props;  
 *return* (  
 <ValidatorForm *ref*="form" *onSubmit*={*this*.*handleSubmit*} *onError*={errors => console.log(errors)} >  
 {response && (  
 <Grid *item xs*>  
 <Typography *align*={'center'} *variant*="body2">  
 {response}  
 </Typography>  
 </Grid>  
 )}  
 <TextValidator  
 *margin*="normal"  
 *label*="Email"  
 *onChange*={*this*.*handleChange*}  
 *name*="email"  
 *fullWidth  
 value*={formData.email}  
 *variant*="outlined"  
 *validators*={['required', 'isEmail']}  
 *errorMessages*={['This field is required', 'Email is not valid']}  
 />  
 <Box *mt*={2} *mb*={2}>  
 <Button *disabled*={send} *type*="submit" *fullWidth variant*="contained" *color*="primary">  
 Restore  
 </Button>  
 </Box>  
 <Grid *item xs*>  
 <Typography  
 *align*={'center'}  
 *onClick*={toggleForgotPassword}  
 *className*="other-account"  
 *variant*="body2"  
 >  
 {'Already have login and password?'}  
 </Typography>  
 </Grid>  
 </ValidatorForm>  
 );  
 }  
}  
  
*export default* ForgotPassword;

#### Основи Material UI

React JS material-ui - це частина Material Design. Material Design - це мова дизайну, вперше представлений Google в 2014 році. Це візуальна мова, який використовує макети на основі сітки, гнучку анімацію і переходи, доповнення та ефекти глибини, такі як освітлення і тіні. Мета Material Design зводиться до трьох речей: Створення, Уніфікація та Налаштування.

material-ui - це набір компонентів React, який реалізує Google Material Design (material-ui недавно випустили v1 бібліотеки). Ці компоненти працюють ізольовано, це означає, що вони є само-підтримують і вводять тільки ті стилі, які вони повинні відображати.

#### Основи EsLint

ESLint це інструмент, який дозволяє проводити аналіз якості вашого коду, написаного на будь-якому вибраному стандарті JavaScript. Він призводить код до більш-менш єдиного стилю, допомагає уникнути помилок, вміє автоматично виправляти багато зі знайдених проблем і відмінно інтегрується з багатьма інструментами розробки. До речі, він, як і інші Лінтера, не зобов'язує вас до одного якогось конкретного стилю. Навпаки - ви можете вибрати щось з кращих практик і доопрацювати на свій розсуд

#### Основи StyleLint

Stylelint для CSS - це фантастичний проект, який дозволяє легко застосовувати угоди і уникати помилок в стилях. У випуску версії 9.5.0 була додана підтримка першого класу для написання CSS в JS. Тепер виконувати лінтінг в стилі популярних бібліотек, таких як styleled-components і emotion, стало простіше, ніж коли-небудь.

### Серверна частина

#### Основи NodeJS

Node або Node.js - програмна платформа, заснована на движку V8 (здійснює трансляцію JavaScript в машинний код), що перетворює JavaScript з вузькоспеціалізованого мови в мову загального призначення. Node.js додає можливість JavaScript взаємодіяти з пристроями введення-виведення через свій API (написаний на C ++), підключати інші зовнішні бібліотеки, написані на різних мовах, забезпечуючи виклики до них з JavaScript-коду. Node.js застосовується переважно на сервері, виконуючи роль веб-сервера, але є можливість розробляти на Node.js і десктопні віконні додатки (за допомогою NW.js, AppJS або Electron для Linux, Windows і macOS) і навіть програмувати мікроконтролери (наприклад, tessel і espruino). В основі Node.js лежить подієво-орієнтоване і асинхронне (або реактивне) програмування з неблокуючим введенням / висновком.

#### Основи Express

Express - це мінімалістичний і гнучкий веб-фреймворк для додатків Node.js, що надає великий набір функцій для мобільних і веб-додатків. Express надає тонкий шар фундаментальних функцій веб-додатків, які не заважають вам працювати з давно знайомими і улюбленими вами функціями Node.js.

Приклад старту сервера

*export default* ForgotPassword;  
  
*let* express = require('express');  
*let* app = express();  
  
app.get('/', *function* (req, res) {  
 res.send('Hello World!');  
});  
  
app.listen(3000, *function* () {  
 console.log('Example app listening on port 3000!');  
});

#### Основи Passport

PassportJS - це middleware для авторизації під node.js. Passport підтримує авторизацію за допомогою величезної кількості сервісів, включаючи «Google» та інші твітери.

Приклад використання:

passport.use(  
 "jwt",  
 *new* JwtStrategy(optsJWT, *async* (jwt\_payload, done) => {  
 *try* {  
 *const* {\_id} = jwt\_payload.data;  
 *if* (!mongoose.Types.ObjectId.isValid(\_id)) {  
 *return* done(*null*, *false*);  
 }  
 *let* customer = *await* Customer.findById(\_id);  
 *if* (customer) {  
 *return* done(*null*, customer);  
 }  
 *return* done(*null*, *false*);  
 } *catch* (e) {  
 *return* done(e, *false*, e.message);  
 }  
  
 })  
);

*// @route GET /customer  
// @desc Return current customer  
// @access Private*router.get(  
 "/",  
 passport.authenticate("jwt", {session: *false*}),  
 getCustomer  
);

#### Основи BcryptJS

bcrypt - це функція збору паролів, розроблена Нільсом Провосом та Девідом Мазьєром, заснована на шифрі Blowfish, і представлена в USENIX у 1999 році. Окрім включення солі для захисту від атак на райдужну таблицю, bcrypt є пристосувальною функцією: з часом кількість ітерацій може бути збільшена, щоб зробити її повільніше, тому він залишається стійким до атак пошуку навіть із збільшенням обчислювальної потужності.

Функція bcrypt є алгоритмом хешування пароля за замовчуванням для OpenBSD та інших систем, включаючи деякі дистрибутиви Linux, такі як SUSE Linux.

Приклад використання

*const* salt = *await* bcrypt.genSalt(10);  
newCustomer.password = *await* bcrypt.hash(password, salt);

#### Основи JSONWebToken

JSON Web Token - це Інтернет-стандарт для створення token доступу на основі JSON, які стверджують деяку кількість претензій. Наприклад, сервер може генерувати маркер, який містить претензію "увійшов як адміністратор" та надав його клієнту. Потім клієнт може використовувати цей маркер, щоб довести, що він увійшов як адміністратор. Маркери підписуються приватним ключем однієї сторони (як правило, серверним), так що обидві сторони (інша, яка вже є, певними підходящими та надійними засобами, що мають відповідний відкритий ключ), зможуть перевірити, що маркер законний. Токени розроблені таким чином, щоб вони були компактними, захищеними URL-адресами та зручними для використання, особливо в контексті єдиного входу (SSO) веб-браузера. Заявки JWT можуть зазвичай використовуватися для передачі ідентифікації користувачів, що підтверджують автентичність, між постачальником послуг ідентифікатором та постачальником послуг або будь-який інший тип претензій, як того вимагають бізнес-процеси.

Приклад використання

jwt.sign({data: payload}, process.env.JWT\_SECRET, {expiresIn: 36000}, (err, token) => {  
 *return* res.json({  
 success: *true*,  
 token: "Bearer " + token  
 });  
});

#### Основи Cloudinary

Cloudinary дозволяє легко трансформувати зображення на ходу у будь-який необхідний формат, стиль та розмір, а також оптимізує зображення для мінімального розміру файлу, поряд із високою візуальною якістю для покращеного користувальницького досвіду та мінімальної пропускної здатності. Все це можна зробити, застосувавши динамічні перетворення зображення та URL-адреси доставки. Ви можете будь-коли змінити необхідні перетворення. Усі трансформовані зображення створюються на вимогу та доставляються вашим користувачам за допомогою швидкого CDN з оптимізованим кешуванням.

API завантаження складається з ряду методів для завантаження та управління медіа-активами в хмарі. Методи API REST можна викликати безпосередньо з вашого власного кодового коду або за допомогою одного з SDK Cloudinary, які обгортають API REST і значно спрощують його методи

Приклад використання

*const* userImg = *await* cloudinary.uploader.*upload*(customer.avatarUrl, {  
 folder: `${process.env.cloudinary\_path}/userAvatar`  
});

#### Основи Mongoose

Mongoose - це бібліотека JavaScript, яку часто використовують у додатках Node.js із базою даних MongoDB. У даній статті я збираюся познайомити вас з Mongoose та MongoDB і, що більш важливо, показати, де їх доречно використовувати у вашому додатку.

MongoDB - це база даних, що зберігає ваші дані як документи. Звичайно ці документи мають JSON (\* JavaScript Object Notation - текстовий формат обміну даними, заснований на JavaScript.

Mongoose - це ODM (\* Object Document Mapper - об'єктно-документний відображувач). Це означає, що Mongoose дозволяє вам визначати об'єкти зі строго-типізованою схемою, що відповідає документу MongoDB.

Приклад використання

*let* Customer = *await* CustomerModel.findOne({email: customer.email});

Customer = *new* CustomerModel({  
 email: customer.email,  
 firstName: customer.firstName,  
 lastName: customer.lastName,  
 avatarUrl: userImg.public\_id,  
 customerNo: (rand()).toString(),  
 socialmedia: [customer.typeSocial],  
 isAdmin: *false*,  
 enabled: *true*});

#### Основи Socket.IO

Socket.IO — [JavaScript](https://uk.wikipedia.org/wiki/JavaScript)-[бібліотека](https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%B1%D0%BB%D1%96%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC) для [веб-застосунків](https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA) і обміну даними в реальному часі. Складається з двох частин: клієнтської, яка запускається в [браузері](https://uk.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) і серверної для [node.js](https://uk.wikipedia.org/wiki/Node.js). Обидва компоненти мають схожий [прикладний програмний інтерфейс](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B8%D0%B9_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81). Подібно node.js, Socket.IO [подієво-орієнтована](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D1%96%D1%94%D0%B2%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0" \o "Подієва архітектура).

Socket.IO головним чином використовує протокол [WebSocket](https://uk.wikipedia.org/wiki/WebSocket" \o "WebSocket), але якщо потрібно, використовує інші методи, наприклад [Adobe Flash](https://uk.wikipedia.org/wiki/Adobe_Flash" \o "Adobe Flash) сокети, [JSONP](https://uk.wikipedia.org/wiki/JSONP) запити або [AJAX](https://uk.wikipedia.org/wiki/AJAX) запити, надаючи той же самий інтерфейс.

Приклад використання

socket.on('sendMessage', (message, callback) => {  
 *const* user = getUser(socket.id);  
  
 io.to(user.room).emit('message', { user: user.name, text: message });  
  
 callback();  
});

## Практична частина

### Створюємо клієнтську частину

При написанні клієнтської частини притримувався патерн *single-page application.*

Для створення базової структури системи використаємо шаблонізатор для ініціалізації проектів з використанням ReactJS

npx create-react-app my-app

cd my-app

npm start

Створемо загальну структуру routing

*import* React, { Component } *from* 'react';  
*import* { Route, Switch } *from* 'react-router-dom';  
*import* NotFound *from* '../NotFound/NotFound';  
*import* AuthorizationPage *from* '../Authorization/AuthorizationPage/AuthorizationPage';  
*import* HomePage *from* '../HomePage/HomePage';  
*import* PersonalProfile *from* '../PersonalData/PersonalData';  
*import* PrivateRoute *from* './PrivateRoute';  
*import* Chat *from* '../Chat/Chat';  
  
*class* Routes *extends* Component {  
 render() {  
 *return* (  
 <Switch>  
 <Route *exact path*="/authorization\*" *component*={AuthorizationPage} />  
 <Route *exact path*="/personaldata" *component*={PersonalProfile} />  
 <PrivateRoute *exact path*="/chat" *component*={Chat} />  
 <Route *exact path*="/" *component*={HomePage} />  
 <Route *component*={NotFound} />  
 </Switch>  
 );  
 }  
}  
  
*export default* Routes;

Створемо компонент, який відповідає за захист від переходу на рути на які користувачу не вистачає привілеїв

*import* React, { Component } from 'react';  
*import* { Route } *from* 'react-router-dom';  
*import* { connect } *from* 'react-redux';  
*import* AccessDenied *from* '../AccessDenied/AccessDenied';  
  
*class* PrivateRoute *extends* Component {  
 render() {  
 *const* { component: Component, ...rest } = *this*.props;  
 *const* { isAuthorization } = *this*.props.authorization;  
 *return* (  
 <Route  
 {...rest}  
 *render*={props => {  
 *return* isAuthorization === *true* ? <Component {...props} /> : <AccessDenied />;  
 }}  
 />  
 );  
 }  
}  
  
*function* mapStateToProps(state) {  
 *return* { authorization: state.authorization };  
}  
  
*export default* connect(mapStateToProps, *null*)(PrivateRoute);

### Створюємо серверну частину

Створюємо сервер з використанням потужного фреймворку Express JS з використанням додаткових пакетів для роботи.

Підключаємо базу даних до сервера

*const* mongoose = require('mongoose');  
*let* ConfigDataBAse = require('./ConfigDataBase');  
  
module.exports = mongoose.createConnection(ConfigDataBAse.mongoURI,{  
 useNewUrlParser: *true*,  
 useCreateIndex: *true*,  
 useFindAndModify: *false*,  
 useUnifiedTopology: *true*});

Проводимо ініціалізацію захисту доступу до рутів сервера

app.use(passport.initialize());  
require("./service/passport")(passport);

Підключаємо FTP до сервера

cloudinary.*config*({  
 cloud\_name: process.env.cloudinary\_cloud\_name,  
 api\_key: process.env.cloudinary\_apikey,  
 api\_secret: process.env.cloudinary\_apiSecret  
});

Запускаємо роботу сокета

io.on('connect', (socket) => {  
 socket.on('sendMessage', (message, callback) => {  
 *const* user = getUser(socket.id);  
  
 io.to(user.room).emit('message', { user: user.name, text: message });  
  
 callback();  
 });  
  
});

Запускаємо сервер на 5000 порту

server.listen(PORT, () => {  
 *if* (!process.env.NODE\_ENV) {  
 console.log(`Server start on ${PORT}`);  
 }  
});

### Деплой системи на хостинг

Завантажуємо нашу систему на репозиторій на полярний сервіс GitHub

git init

git add .

git commit -m “first deploy”

git remote add origin url

git push -u origin master

Переходимо на сервіс Heroku

В методах виду деплоя вибираємо метод, деплой використовуючи github. Вибираємо вітку на якій знаходиться система, та натискаємо «Deploy Branch» (див. рисунок 1.1)

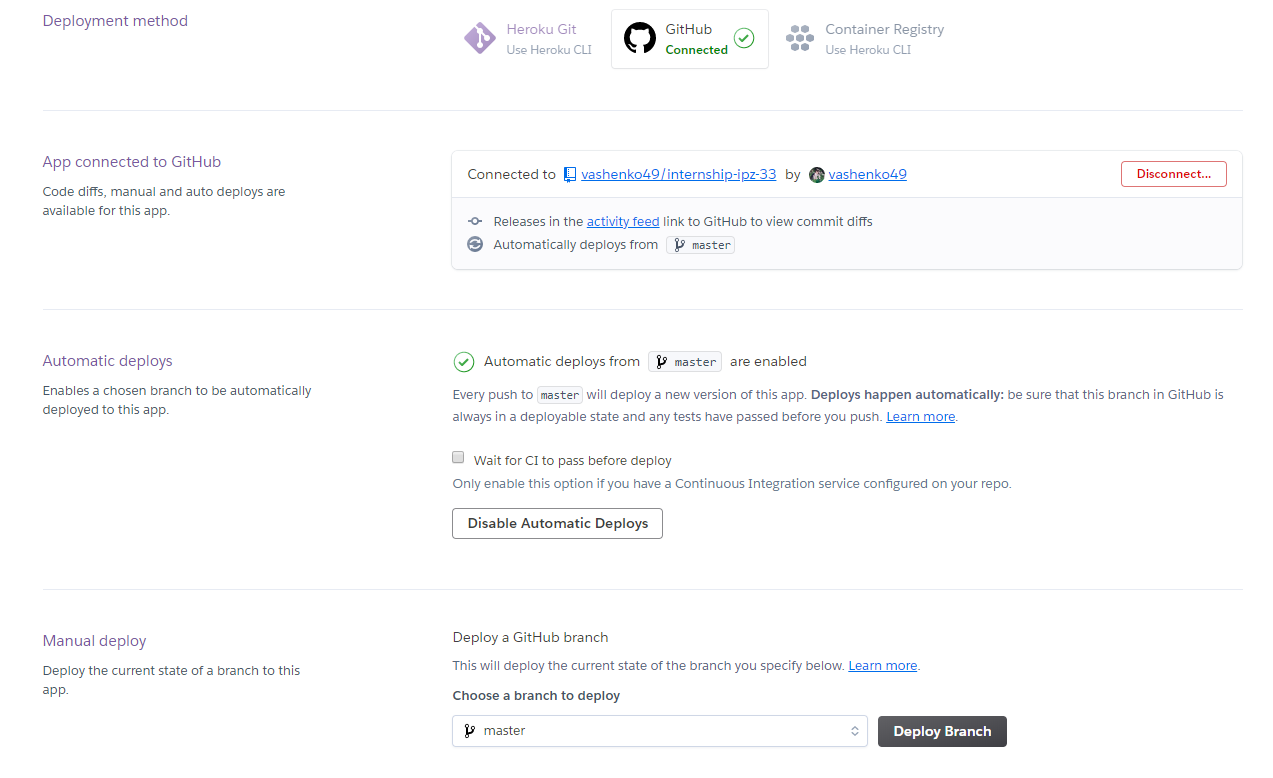


Рисунок 1.1

# Висновки

Було опрацьовано велика кількість тем, в результаті побудована система використовуючи патерни SPA та MVC. Система підтримує аутентифікація користувачів використовуючи сервіси Google, Facebook, Github та автономну аутентифікацію. Використовуючи Socket.io було розроблено віртуальне місце зустрічі в Інтернеті.

# Список використаних джерел

1. Сучасний підручник JavaScript - <https://learn.javascript.ru/>
2. MDN web docs -<https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/JavaScript_Overview>
3. Reactjs.org - <https://uk.reactjs.org/>
4. Webformyself - <https://webformyself.com/polnoe-rukovodstvo-po-react-js-material-ui/>
5. Habr - [https://habr.com/](https://habr.com/ru/post/322550/)
6. Wikipedia - <https://ua.wikipedia.org/>
7. Expressjs - <https://expressjs.com/ru/>
8. Wcoder - <https://wcoder.github.io/notes/nodejs-bcrypt>
9. Tutsplus - <https://code.tutsplus.com/uk/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527>