**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____Литвиненко О.Є.

«___» _____2021 р.

# ДИПЛОМНИЙ ПРОЄКТ

**(ПОЯСНЮВАЛЬНА ЗАПИСКА)**

**ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ**
**"БАКАЛАВР"**

**Тема:** «Розробка *web*-додатку «Перелік справ» з використанням *ReactJS*»

**Виконавець**: Степанов О.О.

**Керівник:** Сябрук І.М.

**Нормоконтролер:** Тупота Є.В.

**Київ 2021**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Освітнього ступеня бакалавр

Спеціальність 123 "Комп'ютерна інженерія"
(шифр, найменування)

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____Литвиненко О. Є.

«_____»_____2021 р.

**ЗАВДАННЯ**
**на виконання дипломного проєкту**

Степанова Олександра Олександровича
(прізвище, ім'я, по батькові)

**1. Тема роботи:** "Розробка *web*-додатку «Перелік справ» з використанням *ReactJS*"

затверджена наказом ректора від "04" лютого 2021 року № 135 /ст.

**2. Термін виконання роботи:** з 17.05.2021 до 20.06.2021

**3. Вихідні дані до роботи:** програмна документація, документація на операційні системи

**4. Зміст пояснювальної записки (перелік питань, що підлягають розробці):**

1) аналіз предметної області;
2) аналіз вимог до програмного забезпечення;
3) розробка *web*-додатку "Перелік справ"

**5. Перелік обов'язкового графічного матеріалу:**

1) сторінка входу в систему;
2) головна сторінка *web*-додатку;
3) діаграма варіантів використання системи

## 6. Календарний план

| № п/п | Етапи виконання дипломного проєкту | Термін виконання етапів | Примітка |
|---|---|---|---|
| 1 | Ознайомитись з постановкою задачі дипломного проєкту | 17.05.21-19.05.21 | |
| 2 | Вивчити спеціальну літературу і технічну документацію | 19.05.21-20.05.21 | |
| 3 | Проаналізувати принципи проєктування веб-систем | 21.05.21-24.05.21 | |
| 4 | Написати розділи дипломного проєкту щодо аналізу предметної області | 25.05.21-31.05.21 | |
| 5 | Проаналізувати алгоритми для програмної реалізації задачі планування власних справ | 01.06.21-04.06.21 | |
| 6 | Написати розділ дипломного проєкту щодо програмної реалізації | 05.06.21-15.06.21 | |
| 7 | Підготовити графічний демонстраційний матеріал, Оформити відгук і рецензію | 15.06.21-20.06.21 | |

**7. Дата видачі завдання** «_17_» травня 2021 р.

Керівник дипломного проєкту_____Сябрук І.М._____
(підпис)

Завдання прийняв до виконання_____Степанов О.О._____
(підпис студента)

# РЕФЕРАТ

Пояснювальна записка до дипломного проєкту “Розробка *web*-додатку «Перелік справ» з використанням *ReactJS* ”: основна частина – 50 с., 8 рис., 4 табл., 19 літературних джерел.

ПЕРЕЛІК СПРАВ, БРАУЗЕР, АРХІТЕКТУРА, ПРОЄКТУВАННЯ, *WEB*-ДОДАТОК, ВЕБ-СИСТЕМА.

**Об’єкт дослідження** – онлайн система для керування власними справами протягом дня.

**Предмет дослідження** – автоматизована система для планування власних справ.

**Мета дипломного проєкту** – розробка та створення програми “Перелік справ” з викристанням *ReactJS*.

**Метод проєктування** – розробка програмного забезпечення з застосуванням мов програмування низького рівня.

Прогнозні припущення щодо розвитку об’єкта дослідження – створення сторінки адміністратора для перегляду всіх справ його підлеглих.

Результати дипломного прокту рекомендується використовувати для конролю власних справ.

ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

*CSS* (*Cascading Style Sheets*) – формальна мова опису зовнішнього вигляду *web*-сторінок.

*HTML* (*HyperText Markup Language*) – стандартизована мова розмітки *web*-документів, що інтерпретується браузером.

*JS* – повноцінна динамічна мова програмування, відома як мова сценаріїв.

*IDE* – інтегроване середовище розробки.

*NPM* - менеджер пакетів *Node*.

*GUI* – Графічний інтерфейс користувача

*DOM – Document Object Model*

# ВСТУП

Ця дипломна робота зосереджена на вирішенні проблеми з приводу ефективного розподілення свого часу студентам. Були обрані саме студенти, тому що вони є потенційними цілями даної проблеми. Дивлячись на те, як студенти розподіляють свій час та енергію між навчанням, роботою та іншими різними аспектами повсякденного життя, виникла необхідність з'ясувати, чи можливо застосувати ефективне управління часом не тільки у зазначених сферах, а й у повсякденному житті для покращення якості їх розпорядження часом. Розглянувши більше повсякденне життя студентів, було виявлено такі його аспекти, а саме: приділення часу друзям, суспільна діяльність, духовне благополуччя, самозростання, та інші. Для багатьох першокурсників момент вступу до вищих навчальних закладів несе з собою нові проблеми, що можуть вплинути на процес навчання студента. Самостійність є однією з ключових проблем, адже щось виконувати самостійно часом може бути складним завданням, насамперед для осіб, що тільки починають вступати у доросле життя. В такому випадку ці особи більше не зможуть користуватися допомогою своїх батьків, як мінімум через велику дистанцію. Це початок дорослого життя. Студенти отримають цінний досвід, що потрібен для досягти успіху в реальному світі, і це може бути приголомшливим. Для того, щоб студенту було комфортно жити у нових умовах, не стикаючись з фінансовими та соціальними проблемами, необхідно мати достатній заробіток. Отже, для цього треба займатися різними видами робіт під час навчання. Було розглянуто, як студентам вдалося поєднати ці сфери свого життя. Управління рівновагою між робочим і особистим життям називають концепцією, яка спрямовує людину до належного способу розподілу часу та енергії між роботою, навчанням та іншими аспектами життя. Як зазначалось раніше, управління часом буде проблемою дослідження, і це буде

відносно пов’язано з роботою, навчанням та соціальним життям студентів у нових умовах. Аналіз базується на важливості ефективного управління часом для студентів, що додає цінностей, креативності та самоорганізації навчального життя молодих осіб. Також було розглянуто наслідки неадекватного управління часом та способи їх уникнення.

Метою даної дипломної роботи є розробка та створення програми, що буде допомагати студентам вирішувати проблему з приводу ефективного використання власного часу на ті, чи інші справи.

Це також буде інструментом для викладачів, щоб краще зрозуміти ефекти роботи та навчання студентів, які можуть вплинути на їх успішність у класі, а також присутність у класі.

**Мета дипломного проєкту** – розробка та створення програми “Перелік справ” з використанням *ReactJS*.

**Об’єкт дослідження** – онлайн система для керування власними справами протягом дня.

**Предмет дослідження** – автоматизована система для планування власних справ.

**Прогнозні припущення щодо розвитку об’єкта дослідження** – створення сторінки адміністратора для перегляду всіх справ його користувачів.

Результати дипломного проєкту рекомендується використовувати для контролю власних справ.

# РОЗДІЛ 1
# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1. Проблематика менедженту власних справ

Важко точно визначити, що таке час. Наприклад, з наукової точки зору це можна описати як перебіг подій з минулого в сучасність та майбутнє. Час це також четвертий вимір реальності, що описує події у тривимірному просторі. Однак перспектива, передбачена в цьому дослідженні, полягає в тому, що час являє собою період, протягом якого необхідно проводити діяльність. Таким чином, час - це ресурс, який знаходиться під власним контролем, і достатній для виконання певного завдання. У цьому відношенні час - це період, протягом якого відбуваються дії чи процеси. З цієї точки зору час стає дефіцитним ресурсом, яким потрібно правильно керувати, інакше нічого не вдається досягти. Пропозиція часу обмежена, тоді як її попит завжди безмежний. Однак, на відміну від інших організаційних ресурсів, яких не вистачає, час ніколи не можна вмикати чи вимикати, замінювати або складати, як сировину. Час - це унікальна величина, яку менеджер не може зберігати, здавати в оренду чи купувати. Тому час розглядається як важливий ресурс, який є обмеженим, дефіцитним, динамічним і не підлягає відновленню. Час тече із заздалегідь визначеною швидкістю, незалежно від обставин та подій, і всі однаково наділені однаковою кількістю відведеного часу незалежно від величини віку, полу, розміру заробітної плати и чи свого статусу у суспільстві. Таким чином, управління цим ресурсом стає необхідним для успішного виконання будь-якого завдання. Щодо дослідження того, як люди сприймають і думають про час, було запропоновано кілька перспектив.

З психофізичних досліджень час розглядається як ментальна конструкція, яка порівнює сприйняття часу з «годинниковим» часом. Соціологічні дослідження, навпаки, розглядають час як соціальну конструкцію, зручність, з якою погоджуються культури, тоді як поведінкові дослідження намагаються передбачити, що можуть робити люди, їх мотиви для певного мислення про час та поведінку, яка пов'язана з ними. Важливо, що в усіх поглядах на час, саме сприйняття часу (або тимчасове сприйняття) призводить до тимчасової поведінки. Як буде розглянуто нижче, на таке сприйняття часу значний вплив має культура.

Не існує загальновизнаного визначення управління часом, оскільки немає згоди щодо його визначення. Як результат, його було визначено та застосовано різними способами. Управління часом визначається як процес планування, організації та здійснення контролю за кількістю часу, витраченого на певні заходи з метою підвищення результативності, або продуктивності. Управління часом також визначається як систематичне застосування стратегій і методів здорового глузду, щоб допомогти людям стати ефективнішими як в особистому, так і в професійному житті. Отже, щоб управління часом було ефективним, йому повинні сприяти цілий ряд навичок, інструментів та прийомів, що можуть дати можливість досягти тих, чи інших цілей. Завдання, які потрібно було виконати, охоплюють широку сферу діяльності, яка включає планування, розподіл, встановлення цілей, делегування, аналіз витраченого часу, моніторинг, організацію, планування та встановлення пріоритетів. Згідно з *Hisrich and Peters*, управління часом передбачає інвестування часу, щоб визначити, що хто хоче від своєї діяльності. Це процес використання часу для ефективного виконання поставлених завдань за допомогою навичок та інструментів для досягнення організаційних завдань та цілей. Це також було описано як метод для керівників задля підвищення ефективності роботи власної та працівників. Тому ефективне управління часом є запорукою високого рівня продуктивності. Таким чином, управління часом стає процесом, за допомогою якого люди можуть виконувати організаційні завдання та цілі. Крім того, Уорд стверджує, що управління часом часто розглядається або подається як сукупність навичок управління часом. Теорія полягає в тому, що коли люди

опановують навички управління часом, вони стають більш організованими, ефективними та щасливими. Однак, згідно з Брегманом, найбільшим і найвизначнішим міфом в управлінні часом є розглядання усього зробленого, якщо людина лише слідує правильній системі, використовує потрібний час для виконання своїх завдань. Таким чином, Брегман стверджує, що це є помилкою, оскільки ми живемо в той час, коли потік інформації та спілкування вимагає час від часу переглядати наші графіки для. Гебре зауважив, що надзвичайно важливим фактором для досягнення успіху є розуміння важливості управління часом задля подальшого персонального розвитку. Водночас розуміння важливості управління часом стає важливим кроком. Наведені вище погляди демонструють, що час є суттєвим і що це цінний ресурс, призначений для використання в більш контрольованому та вигідному способі. Бевіус та Де Смет зазначає, що управління часом - це не лише проблема продуктивності, над якою компанії не мають контролю, воно все частіше стає організаційним питанням, корінні причини якого глибоко закладені в корпоративних культурах. Таким чином, компанії повинні забезпечити, щоб люди отримували інструменти та стимули для ефективного управління своїм часом.

Можна також додати, що управління часом стало інституційною практикою, оскільки топ-менеджмент організацій сприймає це серйозно. Керівництво відповідає за створення середовища, сприятливого для ефективності управління часом, встановлення пріоритетів та забезпечення того, щоб діяльність навколо цих пріоритетів виконувалась і скорочувала час, витрачений на непріоритетні задачі. Вплив управління часом також досліджували з різних аспектів. Ці аспекти включають ефективність роботи та академічні умови. Інші аспекти - це вплив на позитивні наслідки та результати, сприйняття контролю над часом, чи задоволеність роботою, що виконується. Тому на інституційному рівні керівництво повинно розробляти ефективніші практики управління часом, щоб покращити навички управління часом на робочому місці. Більшість студентів проводять час, роблячи речі, які не є актуальними під час їх щоденної діяльності. Насамперед, таким особам необхідно правильно розставити пріорітети з приводу

виконання необхідних завдань. Також, їм потрібно визначити, як вони проводять свій час щодня і що саме з цим роблять, і, відстежуючи свій час, вони можуть легко зрозуміти, як він був витрачений та на що буде витрачений згодом. Управління часом може бути складним, але в той же час може бути простим, якщо студент може відстежувати свій графік, зосереджуючись більше на записі виконаних завдань, а не на написанні завдань, які ще потрібно виконати. Управління часом насправді є формою управління життям, і контроль над своїм життям означає контроль над своїм часом, а контроль над власним часом означає контроль над подіями у власному житті. Це допоможе їм відстежувати свої досягнення і змусить їх процвітати, робити більше, пізнавати свої можливості, виходити за власні рамки. Цей процес допоможе студентам відокремити свої погані звички, що заважають їм регулярно виконувати свої завдання, тим самим звільняючи місце для правильних, корисних звичок. Управління часом можна охарактеризувати як акт організації вашого графіку таким чином, щоб ви ефективно і без зайвих зусиль досягали своїх цілей. Гарольд, з його науковою технікою управління, мав на меті збільшити продуктивність свого працівника. Для цього він проводив дослідження часу та руху, щоб з'ясувати, як він може максимізувати обсяг виконаної роботи за певний період часу. Управління часом - це інструмент підвищення цілі продуктивності, особливо для студентів, які одночасно працюють і навчаються.

Управління часом полягає не лише в тому, щоб зробити можливим підвищення продуктивності, а й в тому, щоб збалансувати своє життя таким чином, щоб отримувати задоволення від того, що робите.

У школі: це передбачає не просто навчання, а кількість сил та часу, витрачених на читання, вивчення, підготовку необхідних матеріалів до тих, чи іеших предметів.

На роботі: працюючи неповний робочий день, або повний робочий день, має пріоритетне значення для забезпечення економічних потреб. Хоча студенту краще працювати неповний робочий день, щоб утримати рівновагу.

Поїздки на роботу: залежно від місця вашого проживання та доступності транспорту, час, витрачений на очікування громадського транспорту, може зайняти багато часу протягом дня.

Сімейний час: спілкування або проведення часу з родиною, братами, сестрами, та іншими родичами, якщо необхідно - це ті стосунки, які підтримують вас як людину.

Спілкування: час, проведений з друзями, загалом є одним з найважливіших витрат цього русурсу. Вони служать безкорисливою можливістю насолодитися відпочінком із друзями та тими, з ким можливо почуватися комфортно.

Особисте життя: якщо у вас є партнер дівчина, або хлопець, ці стосунки потребують віддaності та часу, щоб процвітати.

Дозвілля: інтереси, захоплення та заходи персонального розвитку дуже важливі, і для їх підтримки потрібен час і відданість. Тож краще знати ті, на які варто витратити час.

Сон: недостатній сон і відпочинок, безумовно, змусять усе ваше життя страждати. Сном не треба нехтувати, це пріоритет, на який необхідно витрачати як мінімум 6-8 годин, залежно від віку організму та потреб. Це допоможе зберегти вашу енергію та змусить зосередитися на важливих речах, щоб уникнути проблем, пов’язаних зі стресом.

Будучи студентом, управління часом є важливою сферою, що допомагає реалізувати бажання досягти власних цілей. Для більшості з них, процес досягнення неохідних речей може викликати велику кількість проблем та складностей, у випадку, якщо студенти не знають, як саме розподілити власний час на виконання певних задач. Важливість управління часом оцінюється ефективністю, результативністю та саморозумінням, що можна отримати в результаті використання цього цінного ресурсу. Точне усвідомлення ціни використовуваного часу може допомогти постійно стежити за собою, а також відкидати непотрібні дії, які не принесуть якоїсь вигоди. Планування діяльності може допомогти студенту актуалізувати свої цілі та працювати у напрямку до них,

а також може змусити його уникати деяких нерегулярних звичок, які можуть відволікти від основних цілей.

Для студента управління часом може бути використано в різних сферах, таких як робота, навчання та власне дозвілля. Ефективне використання часу може збалансувати напругу між цими сферами, а також покращити результат отриманий з докладених зусиль. У організованого студента, цілі будуть досягнуті вчасно, а подальші завдання будуть добре відомі. Жоден студент не повинен залишатися без денного плану, так як таке відображення власних задач досить добре допомагає оцінити необхідний час для іх досягнення. Управління часом служить інструментом прийняття рішень, він нагадує про завдання, які ще мають бути виконані.

Час є критично важливим, і ним потрібно правильно керувати, щоб дати результати (рис. 1.1). Його заощадження можна розглядати як нерелевантне заняття на той момент, коли призводиться втрата цього дорогоцінного ресурсу. Добре, щоб учень звільнив місце на своєму столі перед тим, як приступити до роботи в школі, щоб вільний простір сприяв чіткому мисленню.

Після того, як проблема узгодження часу доступності буде усвідомлена, наступне, що слід зробити з нею - це надати перелік стратегій, щоб взяти на себе більший контроль з часом. Робити менше, не більше. З’ясуйте, що вам заважає. Складіть список дублів і регулярно перевіряйте його. Знайдіть баланс. Зробіть деякі необхідні скорочення, щоб уникнути перевантаження роботою.

Рис. 1.1. Проблема недостачі часу

## 1.2. Принципи розподілення часу при виконанні справ

Принцип вимушеної ефективності - ідея цього принципу полягає в тому, що ніколи не вистачає годин в добі на все, але завжди їх достатньо, щоб зробити найважливіші справи. Студент може відчувати, що йому не вистачає часу, щоб поєднати всі щоденні дії разом, але це те, з чим стикаються всі інші, і це ніколи не зміниться.

Принцип імпульсу - цей принцип стверджує, що потрібно вкласти досить велику кількість енергії, щоб перевершити початкову інерцію та опір для запуску проєкту, але після його запуску необхідно докласти набагато менше енергії, щоб продовжувати рухатися. Імпульс відноситься до кількості руху, яке має тіло чи об'єкт, і якщо тіло чи предмет рухаються, тоді воно має імпульс. Якщо завдання або проєкт запущено, імпульс генерується, і яким би малим не було завдання чи проєкт, він завжди накопичуватиме імпульс. Щоб застосувати цей принцип, найважливіші цілі повинні стати зрозумілими в довгостроковій та

короткостроковій перспективі. Після чого необхідне розбиття цілей на виконуючі завдання, після чого завдання можна буде починати.

Принцип навіювання - цей принцип свідчить про добрий намір керувати пріоритетами, виходячи з ваших цілей, і він може бути марним, якщо ви завжди не пам'ятаєте про цілі та пріоритети і не бачите їх. Цей принцип використовує візуальне розпізнавання для виконання завдань, і він дуже потужний, оскільки є найбільш домінуючим джерелом інформації. Для ефективного застосування цього принципу важливо визначити найважливіші справи, які потрібно зробити, та записати їх. Зображення або малюнок також можна використовувати для представлення того, що ви плануєте зробити. Наступним кроком є розміщення цього малюнку чи зображення саме там, де його можна бачити постійно, і де воно буде нагадувати про необхідність виконування завдань. І нарешті, ви можете повідомити близьких, друзів, або вчителів про свою бажану мету, щоб вони могли допомогти, а також нагадати вам, що вам потрібно зробити.

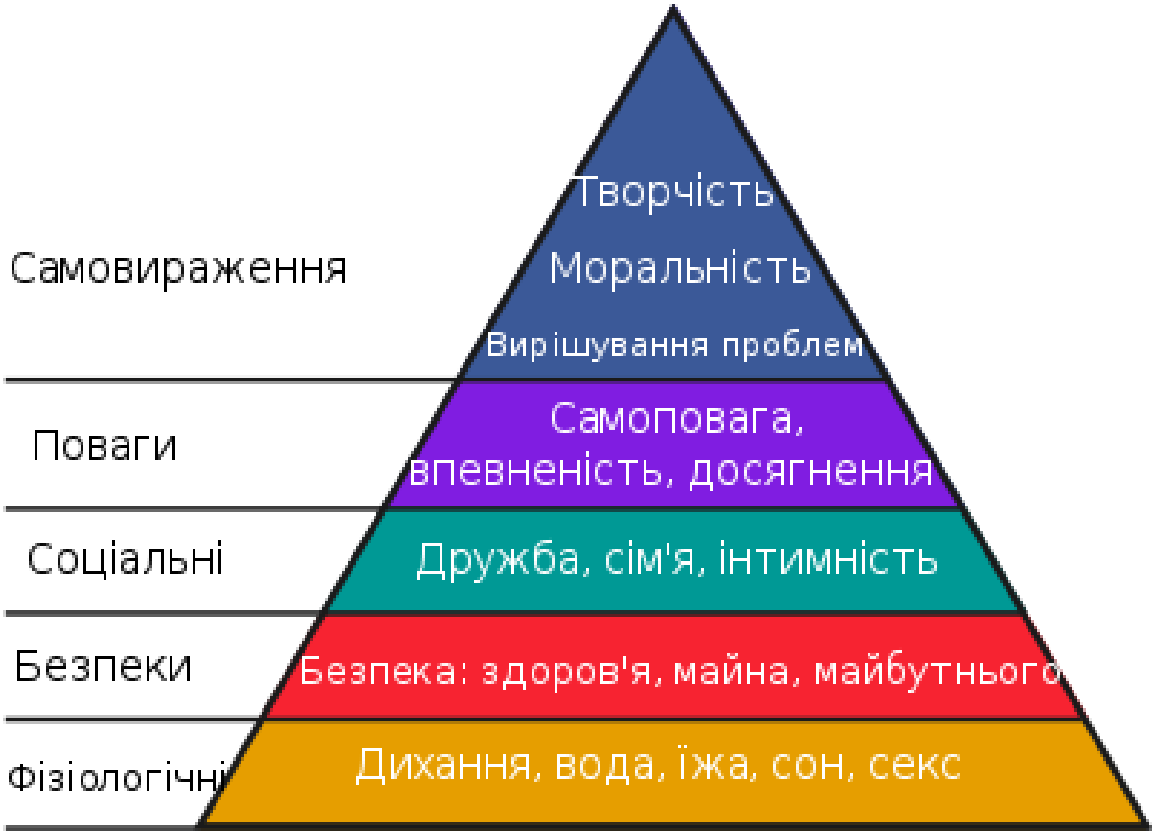


Рис. 1.2. Піраміда потреб Абрагама Маслоу

Ієрархія потреб Маслоу (рис. 1.2) складається з таких потреб:

Фізіологічні потреби - це основи основ всіх потреб, таких як їжа, вода, сон, притулок та дихання. У цих сферах потрібно рівне задоволення, щоб уникнути внутрішнього чи фізичного дисбалансу.

Потреби в безпеці: цей наступний рівень потреб складається з безпеки,

позбавлення від болю чи шкоди, емоційної безпеки, ресурсів, сім'ї, моралі, власності та справедливості.

Потреби у приналежності: викликані сильно соціальною природою людей, яких багато людей насолоджуються почуттями належності, дружби чи кохання. Все це можна задовольнити за допомогою соціальних взаємодій.. Цей тип потреб розвиває мотивацію бути частиною групи та можливість налагодити значущі стосунки та отримати підтримку з боку інших.

Потреби в повазі: це також можна назвати потребами, це те, що людина думає про себе. Повага для інших необхідна, тому що значна частина нашої спільноти регулює свої відношення з іншими людьми базуючись на сприйнятті та розумінні поваги до інших учасників суспільства.

Потреба самовираження: остання з можливих потреб судячи за теорією піраміди потреб Абрагама Маслоу. Вона є можливою тоді і тільки тоді, коли будуть усунуті усі попередні потреби. Самовираження є основою нашого персонального розвитку та представлення власних особистостей світові. Без такої потреби буде неможливе майбутнє зростання, так як вона включає в себе творчість, фантазію, прагнення, натхнення, погляди та цілі тієї, чи іншої особистості. Без цих потреб у кожній галузі студентського життя будуть проблеми.

Якщо часом не керувати корисно, це може викликати у будь-якого студента сильний стрес, що ускладнить зосередження. Буде розглянуто стрес та причини його виникнення, а також те, як правильно розпоряджатися своїм часом, щоб уникнути ситуації, що приводять до виникнення стресу. Стрес являє собою як психологічну, так і фізіологічну реакцію на реальну або сприйману загрозу, що вимагає певних дій задля вирішення отриманої проблеми. Стрес також можна розглядати як ситуацію, коли життя чинить на людей тиск, а також почуття, що з ним виникають. Це також назва, що розриває мости, або сила, що створює

навантаження на предмет і тіло. Для студента часто рекомендується поділитися своїми проблемами з родичами, близькими родичами, викладачами, чи навіть консультантами. Існує приказка, що спільна проблема - це проблема, зменшена вдвічі. Це бентежить, коли ви висловлюєте власні думки посеред теорій. Ви можете сперечатися та аналізувати теорії, але ніхто не висловлював своїх власних думок, як це роблять вчені. Поради завжди важливі для студента, вони також допомагають побудувати більш спокійну частину та комфортні умови для навчання. Стрес можна розглядати як механізм виживання, запрограмований давно, щоб підвищити внутрішнє усвідомлення небезпеки та перевести всі ресурси організму на підвищений стан готовності. Щоб студент уникнув стресу, він повинен усвідомити, що він / вона переживає стрес, і повинен чесно подумати про причини стресу.

Також, одним з основних факторів, що може впливати на правильність керування часом, є тиск. Він може виникати з багатьох джерел і досить драматично змінювати результативність методу розподілення часу. Джерела тиску можуть виникати:

З роботи - коли студент більше зосереджується на роботі, а не на навчанні, його навчальна робота буде накопичуватися, а студент в найгіршому випадку може бути відрахованим з ВУЗу. Робота може бути вимогливою і, як наслідок, забиратиме багато енергії, оскільки студенти зазвичай працюють у секторі прибирання та обслуговування, що вимагає фізичної праці. Робота може збігатися з навчанням, якщо студент знає максимальну кількість часу, який він або вона може витратити, щоб уникнути тиску.

Від однолітків - коли студент усвідомлює, що друг, який був гірше за нього, зараз отримує хороші оцінки та успішніше, ніж він. Така ситуація автоматично створюватиме тиск на студента.

Від батьків - кожен з батьків прагне успіху своїх дітей і зазвичай робить все, що від них залежить, щоб досягти успіху. Будучи студентом, коли ви усвідомлюєте, що ваші батьки багато зробили для подальшої вашої освіти, а ваші результати не дають хороших відгуків, ця ситуація спричинить свідомий тиск, що

впливає на його / її навчання. Усього цього тиску можна уникнути, якщо студент може зрозуміти і усвідомити ці тиски на початку.

Зміна. Кожен студент, який вирішив приїхати вчитися, повинен пройти через зміни в навколишньому середовищі та ситуації, оточеній ним. Коли студент приїжджає в регіон, де відбуваються зміни в середовищі, такі як погода, культура та мова, він / вона повинен розпізнати стимули для змін, а не дозволяти результатам падати куди завгодно. Зміни впливають на управління часом, оскільки існують нові способи робити щось, що може спричинити дезорганізацію, тиск та стрес для студента.

Метод дослідження. У цій дипломній роботі якісний метод дослідження буде використаний для досягнення бажаної мети. Якісне дослідження - це поглиблене вивчення того, що люди думають, відчувають, роблять, і найважливіше чому. На даний момент якісний підхід є актуальним, оскільки немає чітко визначених конкретних змінних для вивчення, і тема видається досить цікавою і має багато академічних цілей навчання.

Надійність та обґрунтованість. Надійність відноситься до узгодженості міри концепції, яка включає стабільність, внутрішню надійність та послідовність взаємодії між спостерігачами. Надійність в основному стосується питань узгодженості заходів і охоплює різні виміри. Ідея цього підходу полягає в тому, що, коли дослідник прагне розробити міру концепції, слід враховувати різні аспекти або компоненти цієї концепції. Різні способи реалізації валідності здійснюються за допомогою прогнозної валідності, конструктивної валідності, одночасної дійсності, валідності обличчя та конвергентної валідності. Запитання для співбесіди обговорюють основні наукові питання, а також теоретичний корпус. Він був створений таким чином, що кожен студент матиме одне і те ж структуроване запитання, на яке повинен відповісти.

Визначення управління часом. Метою цих питань було знайти, що студенти знають про управління часом, що вони думають про це, а також наскільки важливим чи корисним вони вважають це. Управління часом - це те, як людина управляє своїм часом, що включає те, як вона це робить, та інструменти, що

використовуються для її реалізації, щоб бути гарним менеджером часу. Управління часом залежить від людей, які мають напружений графік, і тих, хто має менш напружений графік, що поставить питання про те, хто потребує управління часом. Але також кожен потребує управління часом, хоча деякі люди лише можуть це мати на думці. Управління часом виглядає добре в теоретичній формі, але коли справа доходить до його практичного впровадження, лише декілька студентів можуть насправді розпорядитися своїм часом, як слід. І, керуючи часом, потрібно складати графіки, терміни, контрольно-пропускні пункти, і слід працювати над цими речами, як би мало їх не було. Не всі потребують управління часом, але для студента управління часом є дуже важливим. З іншого боку, управління часом - це здатність самоорганізовуватись, складати графіки та визначати пріоритети завдань, що є більш важливим зважаючи на інші речі. Приділення меншої уваги незначним речам допоможе організувати роботу та всі інші необхідні для виконання заходи. Перш ніж можна було впровадити управління часом, людина повинна реалізувати свої звички, щоб мати можливість застосовувати інструменти управління часом у своїй діяльності. Управління часом можна перенести у дійсність, тобто реалізувати принцип графічного відображення використовуючи календарі в Інтернеті, або будь-які інші ресурси, що дозволять створювати графіки та відстежувати статус виконання поставлених задач. Особа, яка керує власним доступним часом, повинна правильно розуміти як ефективно ним розпоряджатися, тобто активно розподіляти завдання таким чином, щоб одна частина переходила у ваше особисте життя, а інша - до інших частин власного життя.

Корисність або важливість. Що стосується важливості управління часом, її слід розглядати як інструмент, який дозволяє студенту бути повною мірою задіяним у всіх заходах однаково і нести відповідальність за забезпечення виконання завдань. Управління часом допомагає виконувати завдання на 100%, навіть коли виникають моменти відвертання уваги від завдання на яке виставляється фокус. Завжди допомагає відзначати важливі дати для конкретної події / діяльності. Коли в довгостроковій перспективі можна зробити так багато

різних речей, можна не пам’ятати їх усіх, тому управління часом допомагає в подібній ситуації тримати справи гладко. Управління часом також допомагає, наприклад, коли хтось запланував щось на конкретну дату, а потім хоче спланувати іншу діяльність на ту саму дату, це змушує їх усвідомити, що вони вже мають плани на той день. За допомогою цих способів можна визначити пріоритети завдань у тому порядку, який з них є більш важливим. Управління часом дуже важливо, оскільки воно змушує вас діяти тактично в складних графіках. Студенту іноді неможливо запам’ятати всі щотижневі графіки, але, записавши їх, він може легко уникнути пропуску будь-якого. Управління часом змушує вас діяти тактично, але це залежить від особистості та того, хто здатний це здійснити. Це дійсно добре для студентів, але насправді це може бути справді складним завданням, і єдиний спосіб досягти успіху в управлінні часом - це регулярна перевірка графіків і завжди їх дотримання. Управління часом є надзвичайно важливим, оскільки це те, як студент може добре організуватися у всіх сферах свого життя, це не лише допомагає лише навчальному часу, але й довгостроково допомагаю у всьому житті особи, яка користується методами регулювання використання часу. Управління часом підвищує якість виробленої роботи, і це потрібно мати на увазі, а також готує до майбутнього трудового життя. Управління часом можна розглядати як самодисциплiноване, і що без нього студенти не зможуть виконувати завдання. Студенти можуть робити багато роботи, але, не виконуючи її у потрібний час і в потрібний момент, якість продукції досить сильно знизиться. Є студенти, які хочуть стати підприємцями, і для того, щоб стати ними, потрібна самодисципліна, тому що бажання досягти чогось значимого насправді недостатньо, так виконання справ пов’язаних з підприємницькою діяльністю уз використанням управління часу приведе таких осіб до неминучого успіху у бажаній сфері діяльності. Тобто менеджмент часу допоможе визначити пріоритетні важливіші та менш важливі завдання.

## 1.3. Можливі рішення проблеми менеджменту власних справ

Досить багато рішень можна знайти для будь-якої проблеми, особливо в наш вік сильного розвитку інформаційних технологій. Майже кожна людина в нашому суспільстві має змогу отримати необхідну кількість інформації з приводу будь-якого питання, оплачуючи лише підключення до Інтернету. У всесвітній мережі Інтернет люди діляться своїми ідеями кожну секунду, і це вражає наскільки сильно якась мережа може прискорити розвиток людства. Не зважаючи на це, проблеми все одно виникають і їх, як висновок, необхідно вирішувати. От і з приводу менеджменту часу є декілька наявних рішень. Присутні дві основні групи вирішення цієї проблеми: фізичні та електронні рішення.

Фізичне рішення являють собою паперові, або вироби з інших матеріалів, на які можна наносити інформацію. У більшості випадків найкращим варіантом є блокнот. Він компактний і зручний, в нього можна записати досить великий графік та можна виконувати будь-які зміни з ним. З іншої сторони, фізичні носії інформації є скінченними, тобто в якийсь прекрасний момент вони можуть закінчитися і доки не буде придбано новий екземпляр – не можна буде регулювати власний розпорядок дня.

Щодо електронних носіїв, сьогодні вони присутні майже у кожній нашій життєвій ситуації. Вони розвинулись настільки, що на них можна робити майже все, що завгодно. Конкретніше, можливо використовувати різні додатки, *web*-додатки, або ж сайти для задоволення тих, чи інших потреб. Тобто можливо створити якийсь застосунок, який зможе задовольняти потреби студентів з приводу менеджменту власних справ будь-де та будь-коли. Однак, процес створення звичайного додатку досить складний та затратний. Сайт – не найкращий варіант, через те що на ньому досить не просто реалізувати інтерактивну логіку та мінімізувати час затримки між обробкою та оновленням сторінки, через те що при кожному запиті сторінка оновлюється. *Web*-додаток дозволяє керувати своїм функціоналом без постійного оновлення інтерфейсу і також *web*-додаток досить простий у своєму створенні, так як потребує знання лише базових мов

призначених для фронт-енду. Саме тому *web*-додаток э найкращім варіантом вирішення проблеми ефективного регулювання часу студентами з приводу досягнення їх цілей, як персональних, так і соціальних.

## 1.4. Висновки до розділу

Метою даного розділу було дослідження спрямоване на розгляд проблем, які можуть виникнути у студентів під час їх повсякденного життя, без використання ефективного управління часом та власними справами для виконання персональних задач. Управління часом дуже важливе для усіх студентів, оскільки воно може дійсно дати можливість не відставати від навчального процесу. Багато студентів не використовують управління часом, але, скоріше за все, мають це на увазі для подальшого застосування, щодо діяльності, яку вони матимуть у майбутньому. Але тип системи безпечного збереження інформації у голові, що використовують більшість студентів у наш час часто може виявитись невдалим вибором, оскільки завжди можна щось загубити у власних думках, чи не завжди пам’ятати.

Було виявлено, що управління часом служить інструментом самоорганізації для студента, оскільки воно згладжує процес навчання, змушуючи особу мати достатньо часу для різних видів діяльності спрямованої на персональний розвиток. Даний інструмент також служить підготовкою до кар'єри студента, оскільки він направляє вас до того, як бути більш ефективним та орієнтованим на завдання, щоб отримати якісну продукцію.

Проаналізовано потреби, що наявні у кожної особи цього світу. Кожна наступна потреба не може бути прийнята до уваги доти, доки не буде задовільнена попередня потреба. Фізична потреба, будує каркас для усіх наступних потреб, бо неможливо почати думати про щось нове, не закривши потреби зв’язані з повітрям чи їжею.

Також проаналізовано різні ситуації, які впливають на ефективне управління часом студентів. На кожну з них можлива протидія, про що вказано в цій

дипломній роботі. Було з’ясовано різні способи мотивації студентів до можливості продовжувати навчання. Мотивація є головним ключем до успіху, вона підтримує вас у періоди труднощів та незгод. Це освіжає розум, дає мораль і надію на те, що все рухається у правильному напрямку.

# РОЗДІЛ 2
# АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1. Функціональні вимоги

Функціональні вимоги – судячи з назви, це вимоги які відносяться до програмного забезпечення/продукту, а саме: до фреймворку, що був обраний та використаний, до логіки роботи програмного продукту, методів обробки даних, передбачуваної поведінки, та іншого. Головна відмінність від нефункціональних вимог полягає у тому, що функціональні вимоги, насамперед, визначають що саме буде робити розроблюваний продукт, а нефункціональні – який вигляд буде мати система. Такі вимоги визначаються під час етапу аналізу вимог для створюваного програмного продукту.

Найбільш, мабуть, класичний приклад функціональних вимог на сьогоднішній день, це термін "Технічне завдання". Чого тільки під цим не мається на увазі на різних проєктах. Починаючи від найпростішого високорівневого опису функціональності системи і закінчуючи потужним документом, що описує не тільки всі можливі вимоги до системи, але також і апаратно-програмне середовище реалізації, докладний опис архітектури, опрацьовану схему БД, повністю розписаний *UI* (інтерфейс користувача), і ще багато чого. Безсумнівно, що будь-який з цих варіантів (так само, як і всі проміжні) мають право на існування. Слід підкреслити, що, розмовляючи про розробку документа вимог (або технічного завдання) необхідно, щоб всі сторони, які беруть участь в розмові, розуміли предмет розмови однаково. Це дозволить в подальшому уникнути розчарувань при отриманні результату.

Життєвий цикл є ключовою частиною створення функціональних вимог для програмного продукту. Цей цикл включає в себе фази створення програмного забезпечення, кожна з яких може виконуватися ітераційно. Крім того, сам життєвий цикл не є строго лінійним, оскільки існує досить велика кількість зворотних зв'язків, що впливають на весь хід проєкту.

Крім основних процесів Життєвий цикл , існують так звані процеси управління, що підтримують і супроводжують весь цикл розробки програмного забезпечення: управління вимогами, управління проєктом в цілому, конфігураційне управління, управління ризиками і т.д.

Основою для початку визначення вимог є цілі проєкту. Однозначне трактування цілей дозволяє сформувати однозначне бачення Замовником і Розробником того ефекту, який буде отриманий бізнесом замовника в результаті створення і впровадження проєктованого програмного забезпечення. Цілі проєкту служать свого роду точкою зору зацікавлених сторін на досягнуті в ході реалізації проєкту результати.

Цілі не повинні сприйматися як банальна відписка, що служить введенням в документ. Цілі створення продукту є основою для визначення бізнес-вимог до системи. Це найперший, самий верхній рівень документа вимог. Бізнес-вимоги описують, яким чином розробляється система пов'язана з досягненням бізнес-цілей організації і що вона повинна для цього робити. Можна сказати, що бізнес-вимоги є свого роду завданнями, які повинна вирішувати автоматизована система для досягнення цілей свого створення.

Бізнес-вимоги визначають функціональні і нефункціональні вимоги. На практиці може вийти так, що функціональні і нефункціональні вимоги можуть "заходити" за кордону бізнес-вимог, однак вони ніколи не повинні їм суперечити.

До функціональних вимог відноситься все те, що описує функціональність системи. Тобто, що, як і коли робить система. А також, хто приймає в цьому участь.

В результаті необхідно сформувати мінімальні функціональні вимоги, які будуть установлювати вимоги щодо очікуваних функціональні можливостей

програмного забезпечення, що розробляється для даного дипломної роботи. Було створено таблицю, з зібраними функціональними вимогами до майбутньої системи менеджменту переліку справ. До таких вимог увійшли вимоги щодо реєстрації у *web*-додатку, авторизації користувача у *web*-додатку, можливостей щодо внесення налаштувань, та виходу з *web*-додатку, можливість додати, видалити, редагувати подію користувачем у створюваному *web*-додатку (див. табл. 2.1).

Таблиця 2.1

Функціональні вимоги

| Предмет вимог | Детальний опис |
|---|---|
| Авторизація користувача системи | Гість на кожній сторінці повинен мати змогу зайти в систему. Якщо пошта користувача та пароль перевірені системою, користувач заходить в систему і система повинна повідомити йому що він був залогований, тоді він працює з системою як авторизований користувач, і всі подальші вкладки стануть для нього доступними. |
| Авторизація користувача системи | При некоректному введенні логіну та / або пароля відображається повідомлення про подію та користувачі пропонується повторно заповнити поля електронної пошти та паролю. |
| Авторизація користувача системи | Користувачеві надається 7 спроб для входу в систему, в іншому випадку система блокує його на 1 годину. |

Продовження таблиці 2.1

| Предмет вимог | Детальний опис |
|---|---|
| Реєстрація користувача | Гість повинен мати можливість зареєструватися на сайті. Для цього він може з будь-якої сторінки сайту за посиланням «зареєструватися» перейти на сторінку з формою реєстрації. Після цього, система повинна перенаправити користувача на сторінку входу. |
| Головна сторінка | На головній сторінці, користувач повинен мати змогу вийти із системи. |
| Сторінка “Мої справи” | На сторінці “Мої справи”, користувач повинен бачити календар з обраним сьогоднішнім днем. |
| Сторінка “Мої справи” | На сторінці “Мої справи”, користувач повинен мати змогу обрати будь який день з календаря. |
| Сторінка “Мої справи” | Після обраного дня користувач повинен бачити весь перелік справ на день. |
| Сторінка “Мої справи” | Якщо перелік справ пустий, користувач повинен мати змогу заповнити його. |
| Сторінка “Мої справи” | Користувач повинен мати змогу відмітити затрачений час після виконаної активності. |
| Сторінка “Мої справи” | Кожна додана справа до списку повинна мати свій актуальний статус. |

Закінчення таблиці 2.1

| Предмет вимог | Детальний опис |
|---|---|
| Сторінка “Мої справи” | Повинні бути статуси:<br>- Аналіз;<br>- Готова до виконання;<br>- В роботі;<br>- Завершення роботи; |

## 2.2. Нефункціональні вимоги

Нефункціональні вимоги - це характеристики системи, які не мають прямого відношення до автоматизованих процесів, але, тим не менш, це вимоги без яких працювати з системою було б, м'яко кажучи, проблематично. Як приклад, сюди можна віднести вимоги по зручності у використанні, продуктивності, масштабованості, вимоги до документування, до організаційного та методичного забезпечення, до інтеграції із зовнішніми системами.

Нефункціональні вимоги - описують характеристики системи і її оточення, а не поведінку системи. Тут також може бути приведений перелік обмежень, що накладаються на дії і функції, виконувані системою.

Нефункціональні вимоги не пов'язані безпосередньо з функціями, виконуваними системою. Вони пов'язані з такими інтеграційними властивостями системи, як надійність, час відповіді або розмір системи. Крім того, нефункціональні вимоги можуть визначати обмеження на систему, наприклад на пропускну здатність пристроїв введення-виведення, або формати даних, які використовуються в системному інтерфейсі.

Нефункціональні вимоги ґрунтуються на бюджетних обмеженнях, враховують організаційні можливості компанії-розробника та можливість взаємодії з іншими розробляємими програмними і обчислювальними системами,

а також такі зовнішні фактори, як правила техніки безпеки, законодавство про захист інтелектуальної власності і т.п.

Система “Перелік справ” потребує дотримання певних нефункціональних вимог (див. табл. 2.2).

Таблиця 2.2

Нефункціональні вимоги

| Предмет аналізу | Вимога |
|---|---|
| *Front-end* | *HTML, JS, ReactJS* |
| Архітектурний стиль | *Redux* |
| Середовище розробки | *WebStorm* |
| Швидкість запитів до БД | 2 секунди |
| Кросплатформеність | Система повинна бути інтегрована з будь-яким браузером. |
| Захист системи | Авторизація та реєстрація |
| Швидкість запитів до БД | 2 секунди |
| Надійність | Система не повинна бути підвласна хакерським атакам. |

Щоб розпочати визначення апаратних вимог, необхідно розуміти, що на самому ділі апаратні вимоги означають. Апаратні вимоги – це вимоги, що стосуються характеристик машини, на якій буде відбуватися запуск того, чи іншого програмного продукту. Такими машинами можуть виступати як локальні, так і мережеві варіанти. Кожен з цих варіантів має на увазі різницю між обчислювальними можливостями, місцем розташування та типом з’єднання з всесвітньою мережею Інтернет.

Для створюваного проєкту найбільш доцільним варіантом є використання локальної машини, так як на даному етапі проєктування, не розглядається можливість виходу на міжнародний ринок. Також, немає необхідності в використанні досить сильної апаратної частини, так як проєкт не є настільки вимогливий, щоб виникала потреба орендувати, або купувати сервер. Більш того,

велика машина досить вибаглива в обслуговуванні, та потребує великої кількості фінансових інвестицій задля її ж підтримки. Локальний варіант цілком і повністю покриває потреби програмного продукту що буде розроблено в ході виконання дипломної роботи.

Мінімальні вимоги до машини, на якій буде можливо запустити *web*-додаток “Перелік справ” на *ReactJS* дивитися у таблиці 2.3.

Таблиця 2.3

Апаратні вимоги

| Апаратний компонент | Вимога |
|---|---|
| Процессор | *Intel Core 2 Duo E8400 3GHZ* |
| Оперативна пам’ять | 4 *ГБ* |
| Операційна система | *Windows XP, Linux* |
| Пам’ять | 128 *ГБ* |
| Монітор | *ASUS VS197DE* |
| Клавіатура | *Logitech K280e* |

## 2.3. Огляд обраного програмного забезпечення

Різноманітні бібліотеки та фреймворки *JS* випускаються з так часто і вдало, що це все революціонізує спосіб побудови нового інтерактивного інтерфейсу. *ReactJS* - це одна з таких бібліотек, необхідних для створення веб-інтерфейсів користувача. *ReactJS* - це фреймворк призначений для створення інтерфейсів, який дійсно є відкритим програмним забезпеченням. Даний фреймворк в основному використовується для великих, складних веб-інтерфейсів та для односторінкових додатків, яким являється продукт, що розробляється в даній дипломній роботі. Був розроблений інженером програмного забезпечення *Facebook* Джорданом Уолком.

У *ReactJS* використовується *JSX* замість регулярного використання *JS* для створення шаблонів. *JSX* - це базовий *JS*, який полегшує цитування в *HTML* і використовує синтаксис тегів *HTML* для створення підкомпонентів.

Через *ReactJS* компонент передає колекцію незмінних значень як властивість у своїх тегах *HTML*. Компоненти не можуть змінювати будь-які властивості безпосередньо, але можуть передати функцію назад, за допомогою якої ми можемо внести зміни.

*ReactJS* завжди передається із власними бібліотеками, що дозволяють розробникам створювати різноманітні програми з необхідною інтерфейсною функціональністю. Він має колекцію модульних елементів інтерфейсу, як для пристроїв *Android*, так і для *iOS*. Використовуючи чистий *ReactJS*, ви можете використовувати стандартний зовнішній вигляд та стилі для розробки будь-якого мобільного додатку.

Унікальна об'єктна модель віртуального документу *ReactJS* дозволяє створювати структуру даних для різних функціональних операцій. Він добре реагує на зміни в розрахунках, а також на нові оновлення програмного забезпечення. Це спрощує розробку програм із специфічними логічними операціями, що часто пошкоджуються через оновлення програмного забезпечення. Бібліотека *ReactJS* дозволяє здійснювати зміну при кожному оновленні вкладки.

Переваги *ReactJS* перед іншими фреймворками, а саме *Angular* та *VueJS*:

*ReactJS* легше зрозуміти новачка, так як майже увесь фреймворк спрощений. Підхід на основі компонентів, чітко визначений життєвий цикл та використання простого *JS* роблять *ReactJS* дуже простим для вивчення, побудови та підтримки професійних веб та мобільних додатків. *ReactJS* використовує спеціальний синтаксис *JSX*, який дозволяє поєднувати *HTML* з *JS*. Це не є вимогою, але *JSX* набагато простіший у використанні, ніж його більш сучасна версія. Розробники все ще можуть писати на простому *JS*.

Колись технології були набагато складнішими, але *ReactJS* надає нам можливість робити ті самі дії з минулого у більш простій формі. Кожен проєкт *ReactJS* будується з використанням так званих багаторазових елементів. Це означає, що за допомогою виклику з інших компонентів кожен елемент

інтерфейсу, який ви вже створили, можна використовувати де завгодно у вашому проєкті.

Основним недоліком багатьох програм написаних на *JS* є те, що вони недоступні для пошукової системи. Однак у цьому напрямку деякі зробили значні вдосконалення, але не дуже багато, щоб доповнити практику використання систем глобального пошуку. Дивно, але *ReactJS* досконало працює з системами глобального пошуку. Ви можете запустити *ReactJS* на сервері, і віртуальний *DOM* буде відображатися як звичайна веб-сторінка для браузера, без необхідності в інших налаштуваннях.

Ви можете використовувати модулі *Browserify, RequireJS, EcmaScript* 6, які можна використовувати для автоматичної вставки залежностей через *Babel, ReactJS-di*.

Крім того, *ReactJS* завоював авторитет завдяки наявності зручного набору інструментів, що дозволяє розробникам працювати ще простіше. Інструменти розробника *ReactJS* розроблені як вдосконалення технологій *Chrome*. Ви можете перевірити ієрархію елементів *ReactJS*, а також поточний стан та реквізити системи.

Конкретне проєктування динамічного *web*-додатку з незграбними *HTML*-рядками було майже неможливою справою, оскільки потрібно було дуже специфічне та чітке кодування. Однак *ReactJS* вирішив цю проблему. Тут використовується *JSX*, спеціальний синтаксис, який дозволяє використовувати цитати *HTML* та синтаксис тегів *HTML* для створення різних підкомпонентів.

*ReactJS* є одним з найпоширеніших фреймворків, серед спільноти розробників і є основним, який багато розробників вирішили використовувати по всьому світу. Таким чином, вам буде простіше знайти рішення для проблеми пов’язаної з *ReactJS,* ніж з будь-яким іншим фреймфорком. Як варіант, вивчити *ReactJS*, маючи лише базові знання з *JS* та інтерфейсної розробки, простіше, ніж інші існуючі бібліотеки.

Мета використання *ReactJS* полягає у створенні користувацьких інтерфейсів *web*-додатків з надзвичайною простотою та елегантністю. На відміну від інших, це

найсильніша система. Вона дозволяє користувачеві виконувати процеси, використовуючи *JSX* замість простого *JS*, але у будь-якому випадку, ви можете використовувати його також.

*WebStorm*. *WebStorm* - це сучасна екосистема *JS*, зображена на рисунку 2.1. Вона включає інтелектуальне заповнення коду, виявлення помилок на льоту, потужну навігацію та рефакторинг для *JS*, *TypeScript*, мов таблиць стилів та всіх популярних фреймворків. Основні переваги середовища програмування *WebStorm* наведені у таблиці 3.1.


Рис. 2.1. *WebStorm*

Таблиця 3.1

Переваги середовища програмування *Web-Storm*

| | |
|---|---|
| Сучасні фреймворки | *WebStorm* надає розширену допомогу в кодуванні *Angular*, *React*, *Vue.js* та *Meteor*. |
| Розумний редактор | *IDE* аналізує проєкт, щоб забезпечити найкращі результати завершення коду для всіх підтримуваних мов. Сотні вбудованих перевірок повідомляють про можливі проблеми прямо під час набору тексту та пропонують варіанти швидкого виправлення. |
| Навігація та пошук | *WebStorm* допомагає ефективніше обійти код та заощадити час при роботі з великими проєктами. |
| Налагодження та тестування | *WebStorm* надає потужні вбудовані інструменти для налагодження, тестування та відстеження програм на стороні клієнта. |

## 2.4. Висновки до розділу

Насамперед, найголовнішим моментом під час створення будь-якого програмного продукту є необхідність в формуванні функціональних та нефункціональних вимог, так як реалізація цього процесу дасть можливість до початку розробки самого програмного продукту вже розуміти які задачі він буде виконувати, які функції будуть виконуватись, яким чином застосунок буде функціонувати та на що потрібно буде зосередитися програмним інженерам найбільше. Авжеж, представлені вимоги не є обов’язковими, тобто кожен розробник, чи то одна особа, чи ціла компанія з великою локалізацією працівників,

мають право вирішувати та розробляти свої програмні продукти за тим методом, який їм завгодно, з використанням будь-яких технік створення/реалізації проєктів. Тим не менше, створення функціонального та нефункціонального документування є важливим кроком, коли мова йде про потреби клієнта. Коли клієнт є учасником проєкту, він має конкретні потреби та бажання, що мають бути задокументовані, саме тому для чіткого розуміння поставлених задач створюються як функціональні так і нефункціональні вимоги.

З приводу фреймворку, було обрано саме *ReactJS* через те, що дана бібліотека досить проста в своєму освоєнні та використанні. За досить невеликий проміжок часу, можна отримати можливість створювати *web*-додатки, що по своїй функціональності та стильовій насиченості зовсім не уступають сучасним масштабним перспективним проєктам. Також представлений фреймворк досить популярний у світі, саме тому він має високий рівень підтримки зі сторони оновлень та зі сторони вирішення будь-яких можливих проблем.

# РОЗДІЛ 3

# РОЗРОБКА *WEB*-ДОДАТКУ “ПЕРЕЛІК СПРАВ”

## 3.1. Налаштування та розробка *web*-додатку “Перелік справ”

На самому початку створено новий проєкт за допомогою команди *npx*. *Npx* - інструмент, призначений для того, щоб допомогти стандартизувати досвід використання *npm*-пакетів. Так само, як і *npm* спрощує установку і управління залежностями, розміщеними в реєстрі, *npx* спрощує використання *CLI*-утиліт і інших виконуваних файлів. Це значно спрощує ряд речей, які ми раніше робили за допомогою звичайного *npm*.

*C:\Stepanov>npx create-react-app todoList*

Створений *web*-додаток *ReactJS* запущено та протестовано за допомогою веб-браузера.

*C:\Stepanov>cd todoList*
*C:\Stepanov>npm install --save react@15.0.0 react-dom@15.0.0*
*C:\Stepanov>npm start*

За замовчуванням *npm install <package-name>* встановлює останню версію пакета зі знаком *version*. Встановлення *npm* в контексті проєкту *npm* завантажує пакети в папку *node_modules* проєкту відповідно до специфікацій *package.json*, оновлюючи версію пакета.

Можна вказати ключ для запуску проєкту *-g*; якщо потрібно встановити пакет у глобальному контексті, що дасть змогу використовувати пакет де завгодно на комп'ютері (це загальноприйнято для пакетів інструментального рядка, таких як *live-server*).

Використавши цей ключ до команди встановлення *npm*, ми встановлюватимемо лише пакети із залежностей, тим самим різко зменшуючи розмір модулів *node_modules* до того, що є абсолютно необхідним для роботи системи.

В результаті, після запуску зібраного проєкту отримаємо таке вікно, що зображене на рисунку 3.1.

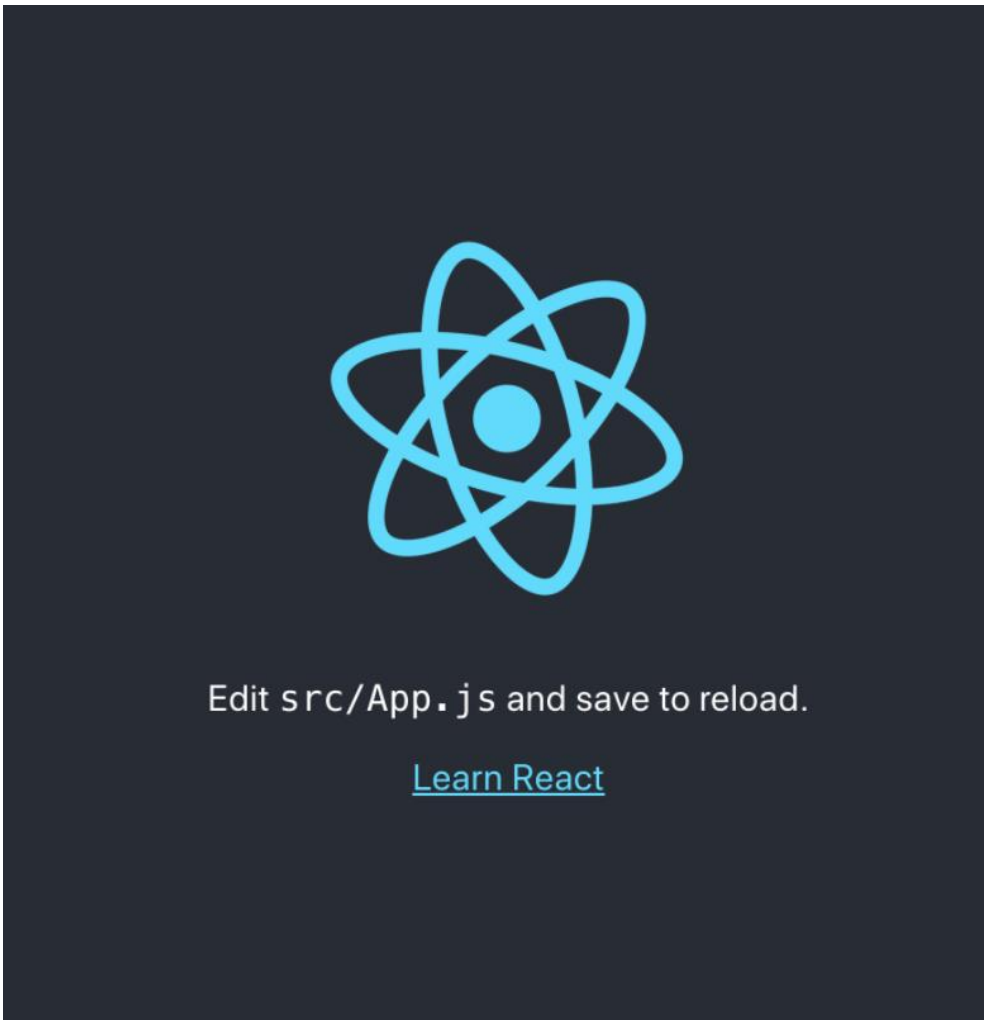


Рис. 3.1. Сторінка запущеного пустого веб-проєкту

Налаштовано *CSS*-файли для візуального відображення *html*-сторінок та компонентів *ReactJS*.

```
body{
background: rgb(74, 23, 167);
}

.list-container{
background: rgb(41, 33, 33);
width: 40vw;
margin: 10em auto;
border-radius: 15px;
padding: 20px 10px;
5color: white;
border: 3px solid rgb(36, 110, 194);
}

.activity{
border: 1px solid white;
border-radius: 5px;
padding: 0.5em;
margin: 0.5em;
}

.activity button{
background: rgb(251, 12, 88);
border-radius: 5px;
margin: 0px 5px;
padding: 3px 5px;
border: none;
cursor: pointer;
color: green;
```

```
float: right;
}

.header{
margin: 0.5em;
font-size: 2em;
text-align: center;
}

.add-activity input[type=text]{
margin: 2.5em 2em;
width: 80%;
outline: none;
border: none;
padding: 0.7em;
}
```

Попередній лістинг показує лише основні класи *web*-додатку, котрі конфігурують колір та стилі *ReactJS* компонентів.

Створено основі компоненти *web*-додатку, котрі будуть відображатися на головній сторінці.

```
function Activity({ activity, index, successActivity, deleteActivity }) {
 return (
  <div
    className="activity"
    style={{ textDecoration: activity.completed ? "line-through" : "" }}
  >
    {activity.title}
```

```
            <button style={{ background: "green" }} onClick={() =>
    deleteActivity(index)}>x</button>
            <button onClick={() => successActivity(index)}>Success</button>
          </div>
        );
    }
```

Під час розробки цього фрагмента було використано *useState* з бібліотеки *ReactJS*, оскільки він потрібен для управління станом у функціональних компонентах. Компонент *Task* повертає трохи *JSX*, щоб визначити, як виглядатиме кожен елемент завдання.

У компоненті *Activity* функція *useState* повертає масив із двома елементами. Перший елемент - це значення поточного стану для завдань, а другий - це функція, яка може бути використана для оновлення завдань:

```
    function Activity() {
      const [activitiesRemaining, setActivitiesRemaining] = useState(0);
      const [activities, setActivities] = useState([
        {
          title: "Do homework",
          completed: false
        },
        {
          title: "Do sport activity",
          completed: false
        },
        {
          title: "Help the grandmother",
          completed: true
        }
      ]);
```

Створено три функції для додавання нової справи до списку, відмітки про виконання та видалення справи зі списку.

```
const addActivity = title => {
  const newTasks = [...activities, { title, completed: false }];
  setActivities(newTasks);
};

const successActivity = index => {
  const newActivities = [...activities];
  newActivities[index].completed = true;
  setActivities(newActivities);
};

const deleteActivity = index => {
  const newActivities = [...activities];
  newActivities.splice(index, 1);
  setActivities(newActivities);
};
```

### 3.2. Варіанти використання системи

Діаграма випадків використання - це діаграма поведінки, яка візуалізує спостережувані взаємодії між акторами та системою, що розробляється. Діаграма складається з системи, пов'язаних випадків використання та суб'єктів і пов'язує їх між собою. Діаграма випадків використання (див. рис. 3.2) повинна описувати бажану функціональність системи та співвідносити її із випадками використання та акторами. Таким чином актор може представляти існуючі точки зору системи

та те, як вони трактуються по-різному - лише завдяки цьому можна повністю зрозуміти вимоги.

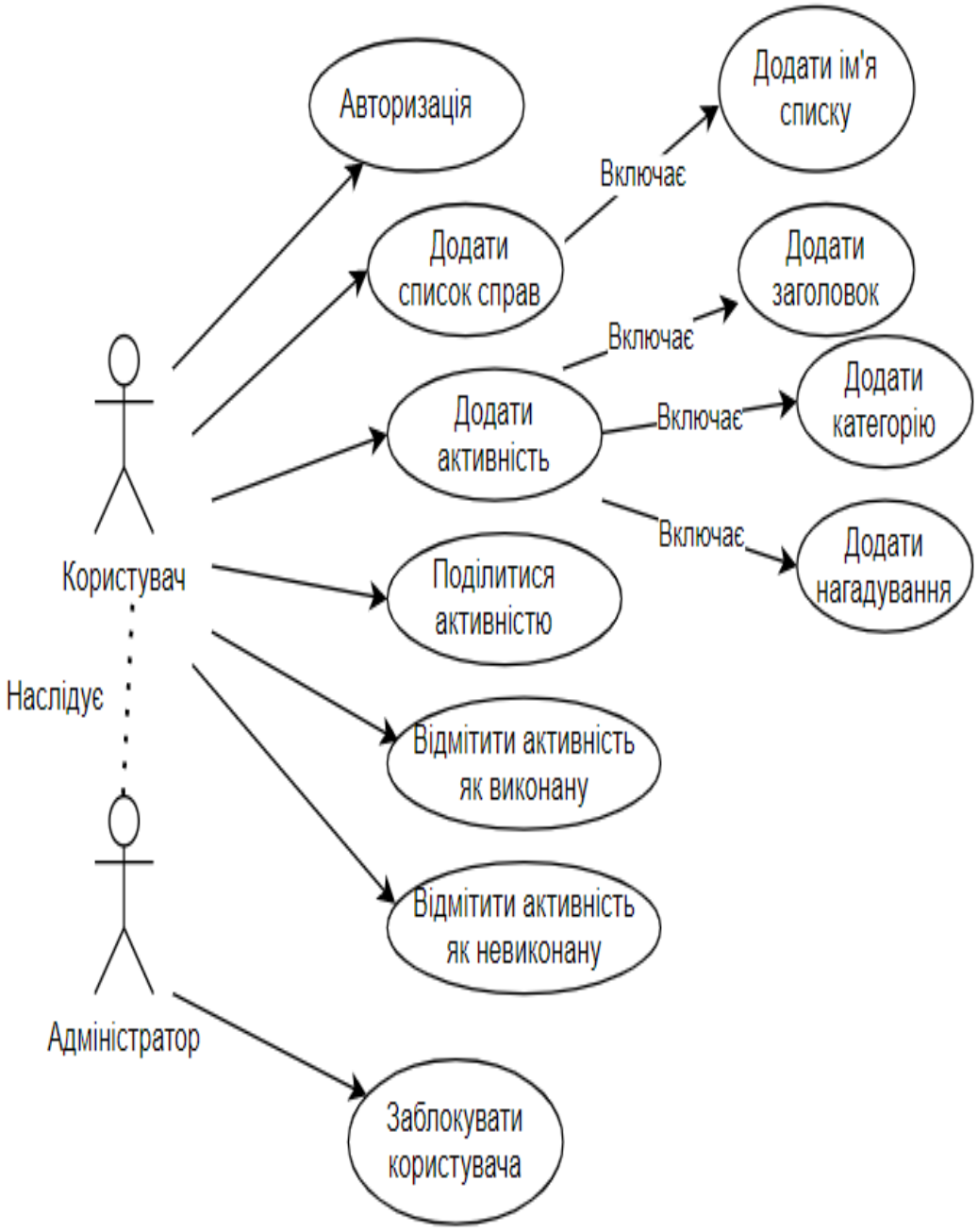


Рис. 3.2. Діаграма варіантів використання системи

1. Користувач авторизується в системі.
2. Користувач додає список справ.
3. Користувач іменує список справ.

4. Користувач додає нову активність до списку.
5. Користувач додає заголовок до активності.
6. Користувач додає категорію і після цього може поставити нагадування перед виконанням справи.
7. Користувач може поділитися активністю з іншими користувачами.
8. Після виконання, користувач відмічає активність завершеною.
9. Користувач може не виконати активність та робить відмітку про це.
10. Адміністратор володіє всіма правами, що і користувач.
11. Адміністратор може заблокувати або розблокувати користувача в разі необхадності.

## 3.3. Тестування програми

Запуск та тестування програми “Перелік справ”, розроблений з використанням фреймворку *ReactJS*.

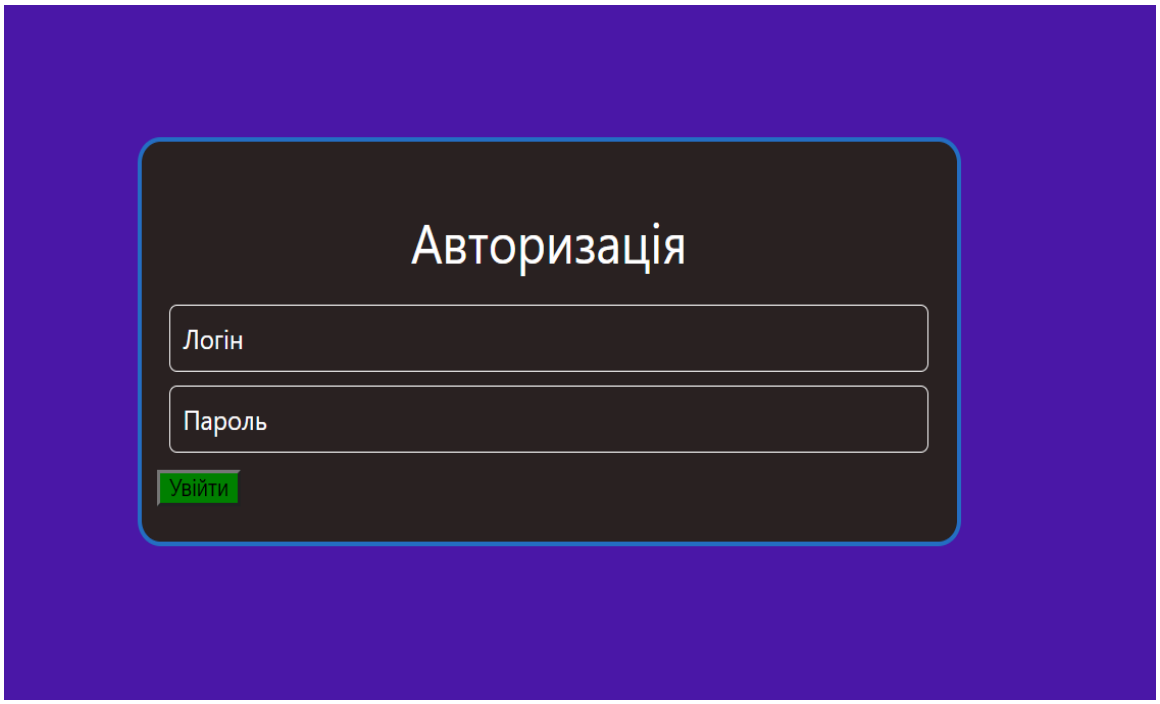


Рис. 3.3. Сторінка входу в систему

На рисунку 3.3 зображена сторінка входу в систему. Зареєстрований користувач після правильно введених даних успішно потрапляє до головної сторінки.

На рисунку 3.4 відображається головна сторінка програми переліку справ.

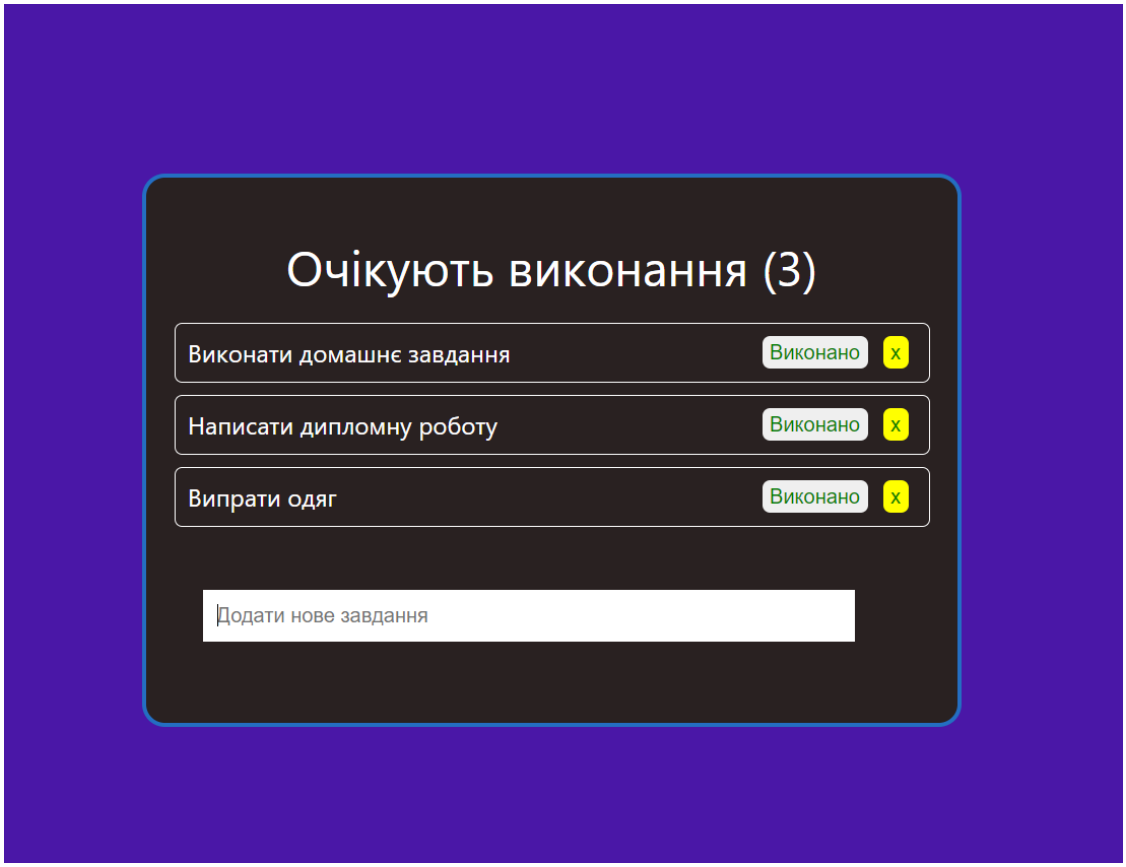


Рис. 3.4. Головна сторінка *web*-додатку

Успішно ввівши логін та пароль відбувається переадресація на головну сторінку. На ній присутні поле для додавання нової справи до списку, список справ на сьогодні, та кнопки зміни статусу.

Важливим елементом системи є кнопка видалення завдання. Без цього функціоналу не було б будь-якої відмінності від блокноту, в якому, як відомо, видаляти написаний текст неможливо

Також, звичайно, по факту виконання того чи іншого завдання, користувач може змінити статус поточної справи (див. рис. 3.5).

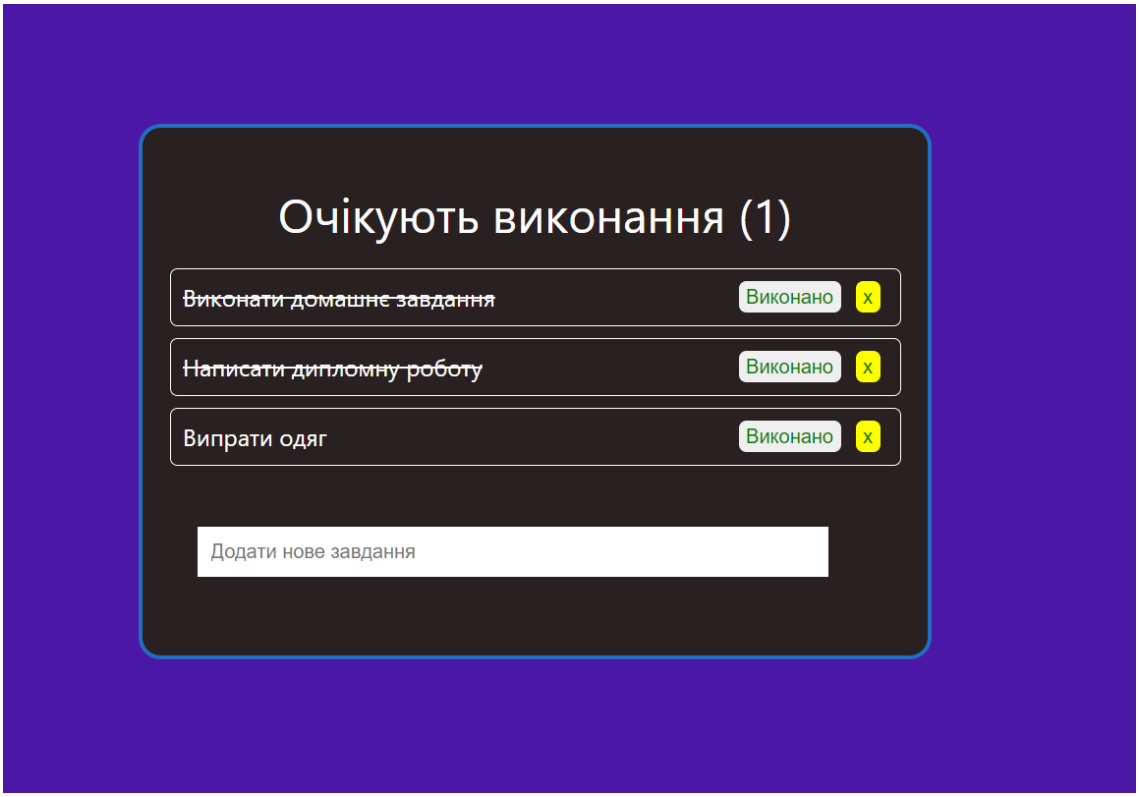


Рис. 3.5. Зміна статусу обраної справи

## 3.4. Висновки до розділу

Використовуючи створений додаток локально, тобто без завантаження на *web*-сервер, отримуємо від нього 2 корисні властивості:

1. Організація розпорядку дня, тобто огляд денного списку власних справ та запис усіх можливих завдань.

2. Наявність робочого варіанту системи, в яку можна мати можливість постійно вносити зміни. Тим самим аналізуючи нові методи та засоби для реалізації певних модулів, або розробляючи нові варіанти чи технології для відтворення програмних засобів.

Тобто, маючи таку програму в власному використанні, постійно є можливість підтримувати її актуальною у використанні *web*-технологій розробки, завдяки реалізації нового функціоналу або масштабування самої програми.

Також, такий екземпляр розробленого програмного продукту можна розповсюдити серед друзів, одногрупників, однокласників, чи інших, задля

допомоги вирішення їх проблеми з приводу менеджменту власного часу та тестування *web*-додатку у реальних ситуаціях.

# ВИСНОВКИ

В результаті виконання дипломної роботи було розроблено *web*-додаток «Перелік справ» з використанням *ReactJS*. Даний *web*-додаток розроблений з усім необхідним функціоналом, згідно з поставленою задачею. В результаті розробки були:

1. Отримані навички аналізу вимог до програмного забезпечення;
2. Виявлені переваги створення односторінкових *web*-додатків;
3. Отримані навички з розробки за сучасними технологіями;
4. Досліджено архітектуру типового клієнтського додатку;
5. Проаналізовано інші архітектурні рішення які використовують при створенні *web*-додатків;
6. Описано основні технології для розробки та їх популярні підходи, а також їх доречність для використання в даному випадку. Розглянуто популярні та доцільні методи для створення клієнтської частини додатку.

Подальше використання розробленого продукту можливе у спрямуванні його можливостей до різних галузей, в яких є необхідність контролю та планування власних справ поточного дня.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Кривов, А. З. Методы разбора задачи анализа данных / А. З. Кривов, А. А. Малинова, А. Б. Курбанец, Д. О. Аналитова // Молодой студент. — 2009. — № 16 (323). — С. 21-29.
2. Кумачев, А. Ф. Алгоритм построения веб-приложений с использованиям сторонних библиотек / А. Ф. Кумачев. // Молодой студент. — 2009. — № 23 (231). — С. 15-19.
3. Амелов В.В. Обработка информации перед проєктированиям програмного обеспечения / В.В. Амелов, А.Ф. Карасев, Ф.С. Пейн // Международный научно-исследовательский журнал. — 2010. — № 1 (43) Часть 4. — С. 11-15.
4. *Highly corrupted image inpainting through hypoelliptic diffusion / U. Boscain, R. A. Chertovskih, J. P. Gauthier et al. // Journal of Analysis Imaging and Vision. – 2013. – Vol. 50, no. 3. – P. 123-128.*
5. *JS*-библиотека для создания пользовательских интерфейсов– [Електронний ресурс]. – Режим доступу: *https://ru.reactjs.org/*
6. Введение в разработку на основе *ReactJS* – [Електронний ресурс]. – Режим доступу: *https://learn.JS.ru/screencast/react/.*
7. *Building Custom Deep Learning Based OCR models*. – [Електронний ресурс]. – Режим доступу: *https://nanonets.com/blog/attention-ocr-for-text-recogntion/.*
8. *Detect text in images*. – [Електронний ресурс]. – Режим доступу: *https://cloud.google.com/vision/docs/ocr* .
9. *ABBYY FineReader Engine*. – [Електронний ресурс]. – Режим доступу: *https://www.abbyy.com/ocr-sdk/* .
10. Ковтун В.П. Сравнение алгоритмов выделения контуров на цифровом изображении и выбор наилучшего алгоритма для реализации страниц// Вопросы науки и образования. — 2015. — № 12 (32). — С. 10-12.

11. *OpenCV Tutorial – Erosion and Dilation of Image*. – [Електронний ресурс]. – Режим доступу: *https://machinelearningknowledge.ai/opencv-tutorial-erosion-and-dilation-of-image/#Dilation* .
12. *Harman D., Salton G. Information Retrieval // Encyclopedia of Computer Science.* 2003. *P*. 858–863.
13. *Desar A., Malish G., Govak M. The support vector machine under test // Computing. — 2001. — Vol. 21 (*1-2*). — P. 124-129.*
14. *Seber G., Lee A. Linear Regression Analysis. : John Wiley & Sons*, 2012.
15. *Baoxun X. An Improved Random Forest Classifier for Text Categorization // JCP*. — 2012 — *Vol*. 7 *(*12*). — P*. 2913-2920.
16. *Create a Web Page Using React*. – [Електронний ресурс]. – Режим доступу:*https://medium.com/swlh/create-a-web-page-using-react-d5ad9d03fb1f.*
17. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проєкти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63.
18. ДСТУ 3008-95 “Документація. Звіти у сфері науки і техніки. Структура і правила оформлення”.
19. ГОСТ 2.106-96 ЕСКД “Текстовые документы”.

ДОДАТОК А

Лістинг реалізованого *web*-додатку

*App.js*

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';
class App extends Component {
render() {
return (
<div className="App">
<header className="App-header">
<img src={logo} className="App-logo" alt="logo" />
<p>
Edit <code>src/App.js</code> and save to reload.
</p>
<a
className="App-link"
href="https://reactjs.org"
target="_blank"
rel="noopener noreferrer"
>
Learn React
</a>
</header>
</div>
);
}
}
```

```
export default App;

App.css
.App {
text-align: center;
}
.App-logo {
animation: App-logo-spin infinite 20s linear;
height: 40vmin;
}
.App-header {
background-color: #282c34;
min-height: 100vh;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
font-size: calc(10px + 2vmin);
color: white;
}
.App-link {
color: #61dafb;
}
@keyframes App-logo-spin {
from {
transform: rotate(0deg);
}
to {
transform: rotate(360deg);
}
```

```
}
Index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import Activity from './components/Activity';
import * as serviceWorker from './serviceWorker';
ReactDOM.render(<Activity />, document.getElementById('root'));
serviceWorker.unregister();

Activity.js
import React, { useState, useEffect } from 'react';
import './Activity.css';
function Activity1({ activity, index, successActivity, deleteActivity }) {
return (
<div
className="activity"
style={{ textDecoration: activity.completed ? "line-through" : ""}}
>
{activity.title}
<button style={{ background: "yellow" }} onClick={() => deleteActivity(index)}>x</b
utton>
<button onClick={() => successActivity(index)}>Виконано</button>
</div>
);
}
function CreateActivity ({ addActivity }) {
const [value, setValue] = useState("");
const handleSubmit = e => {
e.preventDefault();
```

```
if (!value) return;
addActivity(value);
setValue("");
}
return (
<form onSubmit={handleSubmit}>
<input
type="text"
className="input"
value={value}
placeholder="Додати нове завдання"
onChange={e => setValue(e.target.value)}
/>
</form>
);
}
function Activity() {
const [activitiesRemaining, setActivitiesRemaining] = useState(0);
const [activities, setActivities] = useState([
{
title: "Виконати домашнє завдання",
completed: false
}
]);
useEffect(() => { setActivitiesRemaining(activities.filter(task => !task.completed).leng
th) };
const addActivity = title => {
const newTasks = [...activities, { title, completed: false }];
setActivities(newTasks);
};
```

```
const successActivity = index => {
const newActivities = [...activities];
newActivities[index].completed = true;
setActivities(newActivities);
};
const deleteActivity = index => {
const newActivities = [...activities];
newActivities.splice(index, 1);
setActivities(newActivities);
};
return (
<div className="list-container">
<div className="header">Очікують виконання ({activitiesRemaining})</div>
<div className="tasks">
{activities.map((task, index) => (
<Activity1
activity={task}
index={index}
successActivity={successActivity}
deleteActivity={deleteActivity}
key={index}
/>
))}
</div>
<div className="create-task" >
<CreateActivity addActivity={addActivity} />
</div>
</div>
);
}
```

```
export default Activity;

Activity.css
body{
background: rgb(74, 23, 167);
}
.list-container{
background: rgb(41, 33, 33);
width: 40vw;
margin: 10em auto;
border-radius: 15px;
padding: 20px 10px;
color: white;
border: 3px solid rgb(36, 110, 194);
}
activity{
border: 1px solid white;
border-radius: 5px;
padding: 0.5em;
margin: 0.5em;
}
.activity button{
background: rgb(251, 12, 88);
border-radius: 5px;
margin: 0px 5px;
padding: 3px 5px;
border: none;
cursor: pointer;
color: green;
float: right;
```

```
}
.header{
margin: 0.5em;
font-size: 2em;
text-align: center;
}
.add-activity input[type=text]{
margin: 2.5em 2em;
width: 80%;
outline: none;
border: none;
padding: 0.7em;
}

Login.js
import React, { useState } from 'react';
import './Activity.css';
function LoginForm({field}) {
return (
<div
className="activity"
style={{ textDecoration: field.completed ? "line-through" : "" }}
>
{field.title}
{}
</div>
);
}
function Login1 () {
return (
```

```
<button style={{ background: "green" }}>Увійти</button>
);
}
function Login() {
const [fields] = useState([
{
title: "Логін",
completed: false
},
{
title: "Пароль",
completed: false
}
]);
return (
<div className="list-container">
<div className="header">Авторизація</div>
<div className="tasks">
{fields.map((field, index) => (
<LoginForm
field={field}
index={index}
key={index}
/>
))}
</div>
<div className="create-task" >
<Login1 />
</div>
</div>
```

```
);
}
export default Login;
```