Курс "Використання сучасних Web-фреймворків"

Лабораторна робота "Технології JS. Основи React"

Завдання. За допомогою бібліотеки React розробити сайт "Дошка оголошень". Текстові оголошення представити компонентом Note. Передбачити створення, редагування, видалення оголошень. Стан оголошень зберігати в батьківському компоненті Board, управляти станом через властивості, зміни стану передавати через події.

Потрібні знання:

- HTML, CSS
- JavaScript
- Синтаксис ES6 (ECMAScript 6)
- Технології Node.js (пакетний менеджер npm)
- Основи бібліотеки React

Хід роботи

1. Інсталюйте Node.js (https://nodejs.org/)
2. Інсталюйте (в браузер Chrome) React Developer Tools (https://chrome.google.com/webstore).
3. В браузері Chrome перейдіть до списку розширень (chrome://extensions) і для React Developer Tools ввімкніть опцію Allow access to file URLs
4. Інсталюйте редактор Visual Studio Code (https://code.visualstudio.com/) або Sublime Text.
5. Створіть каталог проектів React (наприклад, D:/ReactProjects).
6. Запустіть термінал (Node.js command prompt) і перейдіть в каталог проектів React.
7. Інсталюйте інструментарій Create React App:
   npm install -g create-react-app

8. Згенеруйте проект: create-react-app bulletin-board (це займе декілька хвилин)
9. Зробіть браузер Google Chrome браузером за замовчуванням.
10. Перейдіть в каталог bulletin-board (cd bulletin-board). Запустіть з консолі сервер розробника: npm start. Ви побачите в браузері ваш сайт.
11. Відкрийте в редакторі VSCode або Sublime Text каталог проекту bulletin-board і вивчіть його вміст (файл package.json, каталог src).
12. Відкрийте файл App.js і змініть Learn React на, наприклад, Запрошуємо до React. Перейдіть в браузер щоб побачити результат.
13. Виконайте "чистку" каталогу src: видаліть файли App.css, App.js, App.test.js, logo.svg

14. Скопіюйте в файл index.css вміст однойменного файлу з каталогу нашої лабораторної роботи (на сервері кафедри ПЗАС). Цей файл містить стилі, необхідні для нашого проекту.

```
@import url(http://fonts.googleapis.com/css?family=Shadows+Into+Light);
body,html,div.board,div#react-container {    height: 100%;    overflow: hidden;}
body {    margin: 0;    padding: 0;}
div.board {
background-color: brown;    width: 100%;    height: 1000px;
background: #eab92d;
background: -moz-radial-gradient(center, ellipse cover, #eab92d 57%, #c79810 99%);
background: -webkit-gradient(radial, center center, 0px, center center, 100%, color-stop(57%, #eab92d),
        color-stop(99%, #c79810));
background: -webkit-radial-gradient(center, ellipse cover, #eab92d 57%, #c79810 99%);
background: -o-radial-gradient(center, ellipse cover, #eab92d 57%, #c79810 99%);
background: -ms-radial-gradient(center, ellipse cover, #eab92d 57%, #c79810 99%);
background: radial-gradient(ellipse at center, #eab92d 57%, #c79810 99%);
filter: progid:DXImageTransform.Microsoft.gradient(startColorstr='#eab92d', endColorstr='#c79810',
        GradientType=1);
}
div.note {    height: 150px;    width: 150px;    background-color: yellow;    margin: 2px 0;    position:
        absolute;    cursor: -webkit-grab;    -webkit-box-shadow: 5px 5px 15px 0 rgba(0, 0, 0, .2);    box-
        shadow: 5px 5px 15px 0 rgba(0, 0, 0, .2);}
div.note:active {    cursor: -webkit-grabbing;}
div.note p {    font-size: 22px;    padding: 5px;    font-family: "Shadows Into Light", Arial;}
div.note:hover > span {    opacity: 1;}
div.note > span {    position: absolute;    bottom: 2px;    right: 2px;    opacity: 0;    transition: opacity .25s
        linear;}
div.note button {    margin: 2px;}
div.note > textarea {    height: 75%;    background: rgba(255, 255, 255, .5);}
div.board button {  color: #fff;  font-size: 14px;  box-shadow: 0px 0px 5px 0px rgba(0, 0, 0, 0.55);}
#add {  background-color: #5cb85c;  height: 3em;}
#edit {  background-color: #337ab7;}
#save {  background-color: #5cb85c;}
#remove {  background-color: #d9534f;}
```

15. Створіть в каталозі src новий файл Note.js для нашого головного компонента:

```
import React, { Component } from 'react‘
class Note extends Component {
    render() {
                return (
                <div className="note">
                            <p>Learn React</p>
                            <span>
                                        <button>Edit</button>
                                        <button>Remove</button>
                            </span>
                </div>
                )
    }
}
export default Note
```

16. Відкрийте файл index.js і відредагуйте його:
    - В 9-ому рядку рендерити треба компонент Note
      ReactDOM.render(. . . <Note /> . . . document.getElementById('root'))
    - В 4-ому рядку імпортувати треба компонент Note
      import Note from './Note‘
    - Збережіть зміни і подивіться в браузері результат

17. Поліпшимо вигляд кнопок Edit та Remove. Зробимо це за допомогою значків (icons) з пакету React icons. Відкрийте ще одне консольне вікно , перейдіть в каталог d:/ReactProjects/bulletin-board і подайте команду:
    npm install --save react-icons
18. Відкрийте в редакторі файл Note.js і додайте в секцію імпорту два рядки:

```
import { VscEdit } from 'react-icons/vsc’
import { VscTrash } from 'react-icons/vsc'
```

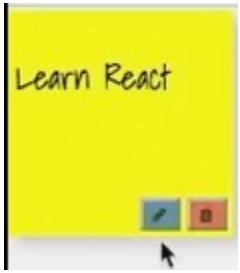
19. Продовжимо редагувати файл Note.js. Інстальовані значки самі є React-компонентами і їх можна використовувати всередині нашої ієрархії. Тож змініть 11-й і 12-й рядки на такі:

```
<button><VscEdit /></button>
<button><VscTrash /></button>
```

Подивіться в браузері як змінився вигляд кнопок.

20. Додамо до кнопок id (стилі з файлу index.css):

```
<button id="edit"><VscEdit /></button>
<button id="remove"><VscTrash /></button>
```



В браузері ми побачимо, що до кнопок додався колір.

21. Пов'яжемо кнопки з подіями onClick :

```
<button onClick={this.edit} id="edit"><VscEdit /></button>
<button onClick={this.remove} id="remove"><VscTrash /></button>
```

Тут в фігурних дужках присутні ESX expressions, що посилаються на відповідні функції (методи класу компонента). Так що в наступному пункті ми додамо ці методи до класу.

22. В файлі Note.js додайте перед методом render ще два методи (поки що як заглушки):

```
edit() {
    alert('editing note')
}
remove() {
     alert('removing note')
}
```

Тепер перед цими методами додайте конструктор, в якому виконайте прив'язку :

```
constructor(props) {
    super(props)
    this.edit = this.edit.bind(this)
    this.remove = this.remove.bind(this)
 }
```

Перейдіть в браузер і впевніться, що введені події працюють.

23. Тепер додамо стан редагування (editing state) до нашого компонента Note.

В конструкторі після рядка super(props) вставте:

```
this.state = {
    editing: false
}
```

В методі edit встановимо стан викликом setState:

```
edit() {
    this.setState({
        editing: true
    })
}
```

Тепер треба змінити наш метод render. Якщо компонент знаходиться в стані редагування, то повинна відображатися форма з textarea та кнопкою збереження. В протилежному випадку виконується звичайне відображення. Є сенс ввести два методи: renderForm і renderDisplay, а метод render буде викликати ці методи в залежності від стану editing. В наступному пункті реалізуємо це.

24. Додайте до компонента Note метод renderForm :

```
renderForm() {
    return (
        <div className="note">
            <form >
              <textarea />
              <button id="save"><VscSave /></button>
            </form>
        </div>
    )
}
```

- Додайте в секцію імпорту рядок:

```
import { VscSave | from 'react-icons/vsc’
```

- Перейменуйте старий метод render на renderDisplay.
- Виконайте прив’язку нових методів в конструкторі:

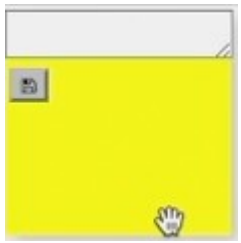
```
this.renderForm = this.renderForm.bind(this)
this.renderDisplay = this.renderDisplay.bind(this)
```

- Додайте метод render:

```
render() {
    return this.state.editing ? this.renderForm() :
            this.renderDisplay()
}
```

Перевірте в браузері як це працює.

Тепер при натисканні значка редагування буде з'являтись форма редагування.



25. Додамо тепер можливість збереження введених даних.

Додайте після методу remove метод save:

```
save() {
    alert('saved!')
}
```

Додайте також прив'язку в конструкторі:

```
this.save = this.save.bind(this)
```

Пов'яжіть кнопку Save з подією onClick:

```
<button onClick={this.save} id="save"><VscSave /></button>
```

Перевірте в браузері чи все працює.

26. Тепер подумаємо про те, як зберегти введений в textarea текст. Тут нам на допомогу можуть прийти посилання (ref). Ref є посиланням на довільний контент. Підкорегуйте текстове поле наступним чином:

```
<textarea ref={input => this._newText = input} />
```

Тут використана callback-функція, аргументом якої є текст, що вводить користувач, і яка зберігає текст у відповідній змінній.

Роздрукуйте значення цієї змінної:

```
save() {
    alert(this._newText.value)
}
```

Перевірте в браузері як це працює.

Далі ми скористаємось цим щоб оновити текст оголошення Note.

27. Тепер давайте створимо компонент Board батьківський до Note, який буде рендерити компонент Note. Тож всередині каталогу src створіть файл Board.js з таким вмістом:

```
import React, { Component } from 'react‘
import Note from './Note‘
class Board extends Component {
    render() {
            return (
             <div className="board">
                    <Note></Note>
             </div>
             )
    }
}
export default Board
```

Внесіть також зміни в 4-й та 9-й рядки файлу index.js :

```
import Board from './Board‘
ReactDOM.render(. . . <Board /> . . . document.getElementById('root'))
```

Перегляньте результат в браузері.

28. Тепер внесемо невеликі корегування в компонент Board. Додамо стан у вигляді масиву об'єктів. Замість рендерингу одного оголошення Note задамо відображення масиву компонентів Note на основі даних стану (змінна notes). Тож додайте до компонента Board конструктор:

```
constructor(props) {
    super(props)
            this.state = {
              notes: [
                {
                      id: 0,
                      note: "Call Lisa“
                 },
                 {
                      id: 1,
                      note: "Email John“
                  }
               ]
          }
}
```

29. Після конструктора додайте метод eachNote :

```
eachNote(note, i) {
    return (
        <Note key={i}  index={i}>
          {note.note}
        </Note>
    )
}
```

- Останнім рядком в конструкторі додайте прив'язку:

```
this.eachNote = this.eachNote.bind(this)
```

- Відкрийте файл Note.js і в методі renderDisplay замініть рядок Learn React рядком

```
<p>{this.props.children}</p>
```

Це означає, що ми будемо відображати властивості всіх дочірніх елементів Note.

30. Поверніться до компонента Board. Замініть в методі render рядок <Note></Note> рядком {this.state.notes.map(this.eachNote)}
Це означає, що треба оглянути всі елементи масиву notes, які є в стані, і для кожного елементу викликати метод eachNote.

- Поверніться в браузер. Ви там побачите тільки останнє оголошення (Email John) через накладку. Тому треба дещо змінити в нашому CSS.
- Відкрийте файл index.css. В стилі div.note ми бачимо position: absolute. Змініть на position: relative; (Пізніше ми сюди повернемось щоб змінити знов на absolute після того, як ми ще попрацюємо над позиціонуванням).
- Поверніться в браузер. Тепер ми побачимо обидва оголошення. Натисніть F12.
- Вивчіть в браузері структуру React компонентів, скориставшись вкладкою Components.

31. Наступним кроком буде надавання можливості редагувати оголошення. Поверніться до компонента Board. В ньому зберігається стан усіх оголошень Note. Після конструктора додайте метод update :

```
update(newText, i) {
    console.log('updating item at index', i, newText)
    this.setState(prevState => ({
      notes: prevState.notes.map(
        note => (note.id !== i) ? note : {...note, note: newText}
      )
    }))
 }
```

- В конструкторі додайте прив’язку:
  `this.update = this.update.bind(this)`

В методі setState використана callback-функція, яка приймає попередній стан, шукає оголошення з заданим id і змінює в ньому текст на newText.

Залишилось питання: як передати відредагований текст з note в newText? Шляхом додавання до кожного Note події onChange.

32. Перейдіть в метод eachNote. В тегі <Note> введіть подію onChange, в обробнику якої викликається метод update:

```
<Note key={i}  index={i}  onChange={this.update} >
    {note.note}
</Note>
```

Тепер перейдіть в компонент Note. При виникненні onChange ми повинні визначити необхідну поведінку. Ми це зробимо в методі save. Відредагуйте його так:

```
save(e) {
    e.preventDefault()
    this.props.onChange(this._newText.value, this.props.index)
    this.setState({
            editing: false
    })
}
```

Тут викликається метод preventDefault, який зупиняє стандартну подію браузера. Потім викликається onChange (що посиляється на метод update), якому передається змінений текст і індекс оголошення. Далі відміняється стан редагування.

- Далі в методі renderForm видаліть у кнопки подію onClick (де викликався метод save):

  ```
  <button id="save"><VscSave /></button>
  ```

  Натомість введіть подію onSubmit для форми:

  ```
  <form onSubmit={this.save} >
  ```

  Зауваження. Розробка цього додатку ілюструє загальну технологію роботи React: ми зберігаємо дані стану в батьківському компоненті, ми передаємо стан через властивості (props), і далі ми передаємо зміни через події.

- Далі поверніться в браузер і перевірте редагування оголошень. Все повинно працювати.

33. Тепер розглянемо як видаляти оголошення. Відкрийте файл Board.js і додайте метод remove :

```
remove(id) {
    console.log('removing item at', id)
    this.setState(prevState => ({
            notes: prevState.notes.filter(note => note.id !== id)
    }))
}
```

Тут при зміні стану використана callback-функція, яка повертає масив notes, за винятком елементу з заданим id.

Наступним кроком додайте onRemove компоненту Note в методі eachNote:

```
<Note key={i}
    index={i}
    onChange={this.update}
    onRemove={this.remove} >
    {note.note}
</Note>
```

- Далі перейдіть в файл Note.js і налаштуйте метод remove :

```
remove() {
    this.props.onRemove(this.props.index)
}
```

Фінальний крок. Поверніться в Board.js і в кінці конструктора виконайте прив'язку метода remove :

```
this.remove = this.remove.bind(this)
```

Поверніться в браузер і впевніться, що видалення працює.

34. Тепер розглянемо як додавати нові оголошення. Отже поверніться до компонента Board і створіть метод add :

```
add(text) {
    this.setState(prevState => ({
            notes: [
                ...prevState.notes,
                {
                    id: 3,
                    note: text
                }
            ]
    })) }
```

Пояснення: метод add отримує деякий текст, викликає setState, що приймає попередній стан і повертає об'єкт – масив notes: всі notes з попереднього стану плюс новий note з id=3 (поки-що так) та заданим текстом.

- Додайте в конструкторі прив'язку метода add :

```
this.add = this.add.bind(this)
```

- Відмітимо, що присвоєння id=3 є досить незграбним. Тому, щоб це виправити, створіть метод, який буде генерувати унікальний id :

```
nextId() {
        this.uniqueId = this.uniqueId || 0
        return this.uniqueId++
}
```

Тепер в методі add замініть рядок id: 3, рядком

```
id: this.nextId(),
```

- Додайте в конструкторі прив'язку метода nextId:

```
this.nextId = this.nextId.bind(this)
```

- Тепер додамо на дошку оголошень кнопку додавання нового оголошення. Отже, відкрийте файл Board.js і додайте в метод render всередину тега div нашу кнопку:

```
<button onClick={this.add.bind(null, "New Note")}      id="add">
        <VscAdd />
</button>
```

Додайте також в секцію імпорту рядок:

```
import { VscAdd } from 'react-icons/vsc’
```

Далі видаліть всі елементи масиву notes (в конструкторі):

```
notes: [ ]
```

Тепер перейдіть в браузер і натисніть на кнопку 

Буде створене нове оголошення.

Натисніть значок редагування і задайте новий текст.

Натисніть значок збереження.

35. Тепер зосередимося на оновленні (модернізації) наших компонентів. Ми тут використаємо два методи з життєвого циклу компонентів React.

Перший метод, componentWillMount, зазвичай викликається перед відображенням компонента, тому його часто використовують для завантаження даних.

Другий метод, componentDidMount, викликається після монтування компонента.

Тож відкрийте файл Board.js. Ми тут внесемо деякі корегування стану. Нашим оголошенням відповідає масив notes, який зараз пустий. Давайте завантажимо в цей масив оголошення використовуючи fetch API запит. fetch дозволяє робити запити, схожі на XMLHttpRequest.

Зразу після конструктора додайте до класу компонента метод componentWillMount наступного вмісту:

```
componentWillMount() {
        var self = this
        if(this.props.count) {
        fetch(`https://baconipsum.com/api/?type=all-meat&sentences=${this.props.count}`)
                        .then(response => response.json())
                        .then(json => json[0]
                                        .split('. ')
                                        .forEach(sentence => self.add(sentence.substring(0, 25))))
        }
}
```

А де ми можемо передати властивості (props) в компонент Board? Відповідь: в файлі index.js, де монтується компонент Board. Тож змініть в цьому файлі 7-й рядок на наступний:

```
ReactDOM.render(…<Board count={50} />…
  document.getElementById('root'))
```

Тут саме час перевірити наш браузер, де ми повинні побачити 50 оголошень, які розміщені поки-що не дуже гарно. Їх бажано розмістити по всьому екрану. Попрацюємо над цим в наступному пункті.

36. Давайте зробимо деякі удосконалення стилю, використовуючи особливості життєвого циклу компонента. Почнемо з додавання до нашого компонента Note методу randomBetween. Його призначення – розмістити наші оголошення випадковим чином по всьому екрану. Тож відкрийте файл Note.js і додайте цей метод.

```
randomBetween(x, y, s) {
    return x + Math.ceil(Math.random() * (y-x)) + s
}
```

Додайте в конструкторі прив'язку цього метода :

```
this.randomBetween = this.randomBetween.bind(this)
```

Тепер додайте метод componentWillMount :

```
componentWillMount() {
    this.style = {
      right: this.randomBetween(0, window.innerWidth - 150, 'px'),
      top: this.randomBetween(0, window.innerHeight - 150, 'px'),
      transform: `rotate(${this.randomBetween(-25, 25, deg)})'
    }
}
```

Прив'язку цього метода виконувати не треба.

- Тепер в методі renderForm приєднайте атрибут стилю до нашого note :

  <div className="note" **style={this.style}**>

  Це стосується також і методу renderDisplay :

  <div className="note" **style={this.style}**>

  Тепер в файлі index.css для div.note змініть позиціонування на absolute :

  position: absolute

  Збережіть зміни, оновіть браузер і ви побачите результат.

37. Тепер покращимо редагування. Ви мабуть помітили, що при натисканні значка редагування попередній текст зникає. Нам треба ще додати удосконалення стилю.
Тож відкрийте файл Note.js і додайте до класу компонента метод componentDidUpdate :

```
componentDidUpdate() {
    var textArea
    if(this.state.editing) {
            textArea = this._newText
            textArea.focus()
            textArea.select()
    }
}
```

Далі додамо в методі renderForm тегу textarea значення за замовчуванням:

```
<textarea ref={input => this._newText = input}
                        defaultValue={this.props.children} />
```

Перейдіть в браузер. Тепер при натисканні значка редагування в textarea буде з'являтись старий текст.

38. Тепер внесемо деякі удосконалення в компонент Board. Тож знайдіть метод eachNote і підправте рядок :

```
<Note key={note.id} index={note.id} . . . .
```

Причина в тому, що коли ми розміщуємо випадково на екрані наші notes, ми повинні відслідковувати який note відредагований або видалений.

39. Внесемо ще одне удосконалення в компонент Note. Додайте до класу цього компонента метод життєвого циклу shouldComponentUpdate :

```
shouldComponentUpdate(nextProps, nextState) {
    return (
    this.props.children !== nextProps.children || this.state !== nextState
   )
}
```

Тут відслідковується ситуація, коли компонент редагується, а текст залишається той самий. Або компонент видаляється. В цьому випадку re-render не потрібний.

Перевірте виконання цих операцій в браузері.

40. Фактично ми завершили розробку нашого застосунку. Залишилось ознайомитися з тим, як робити реліз (продакшн). Тут нам знову допоможе Create React App. Тож, якщо ваш додаток зараз працює, то зупиніть його (з консолі).

Далі подайте в консолі команду npm run build

В результаті буде побудований оптимізований варіант додатку, який можна запустити на сервері.

Тепер перегляньте каталог вашого проекту в редакторі. Ви помітите, що з'явився каталог build, всередині якого буде каталог static з новими файлами (можете їх переглянути). Буде помітна мініфікація.

В коментарях написано, що каталог build готовий до розгортання (deploy). Ми можемо його розгорнути на статичному сервері. Тож інсталюємо сервер командою npm install -g serve , а далі подаємо команду run serve -s build

Тепер наберіть в браузері http://localhost:5000

Реліз нашого додатку буде запущено.