# Лабораторна робота №3

## Розробка інтерфейсів користувача Web-додатків

## засобами фреймворку VueJS

*Метою* лабораторної роботи є оволодіння практичними навичками створення Web-орієнтованого інтерфейсу користувача засобами фреймворку VueJS та мови Javascript.

*Завдання* на лабораторну роботу:

1. Ознайомитись із офіційною документацією (https://vuejs.org/v2/guide/) та безкоштовним посібником з VueJS (https://flaviocopes.com/page/vue-handbook/).
2. Реалізувати функціональні вимоги, визначені у варіанті, відповідно до наступних вимог:

   2.1. Реалізувати функціональність та інтерфейс користувача лабораторної роботи №2 за допомогою фреймворка VueJS.

   2.2. Реалізувати збереження даних у пам’яті подібно до моделей із лабораторної роботи №2.

## Методичні рекомендації

### Запуск додатку “Список справ VueJS”

Вихідний код переробленого під VueJS додатку “Список справ” доступний за посиланням: https://github.com/Aroxed/vuejs-todo.

Для програмування мовою Javascript рекомендовано використовувати середовище розробки Visual Studio Code (VS Code), яке включає інтеграцію із системами контролю версій (зокрема, Git), засоби відлагодження коду (debugger), підтримку автозавершення (autocomplete) та форматування коду, а також інші функції.

Після встановлення VS Code необхідно завантажити додаток, виконавши у командному рядку команду:

```
git clone https://github.com/Aroxed/vuejs-todo
```

Далі у VS Code необхідно вибрати та відкрити каталог `vuejs-todo` і виконати у терміналі (cmd.exe - Windows) наступні команди, що записані у README.md у репозиторії:

```
npm install
npm run serve
```

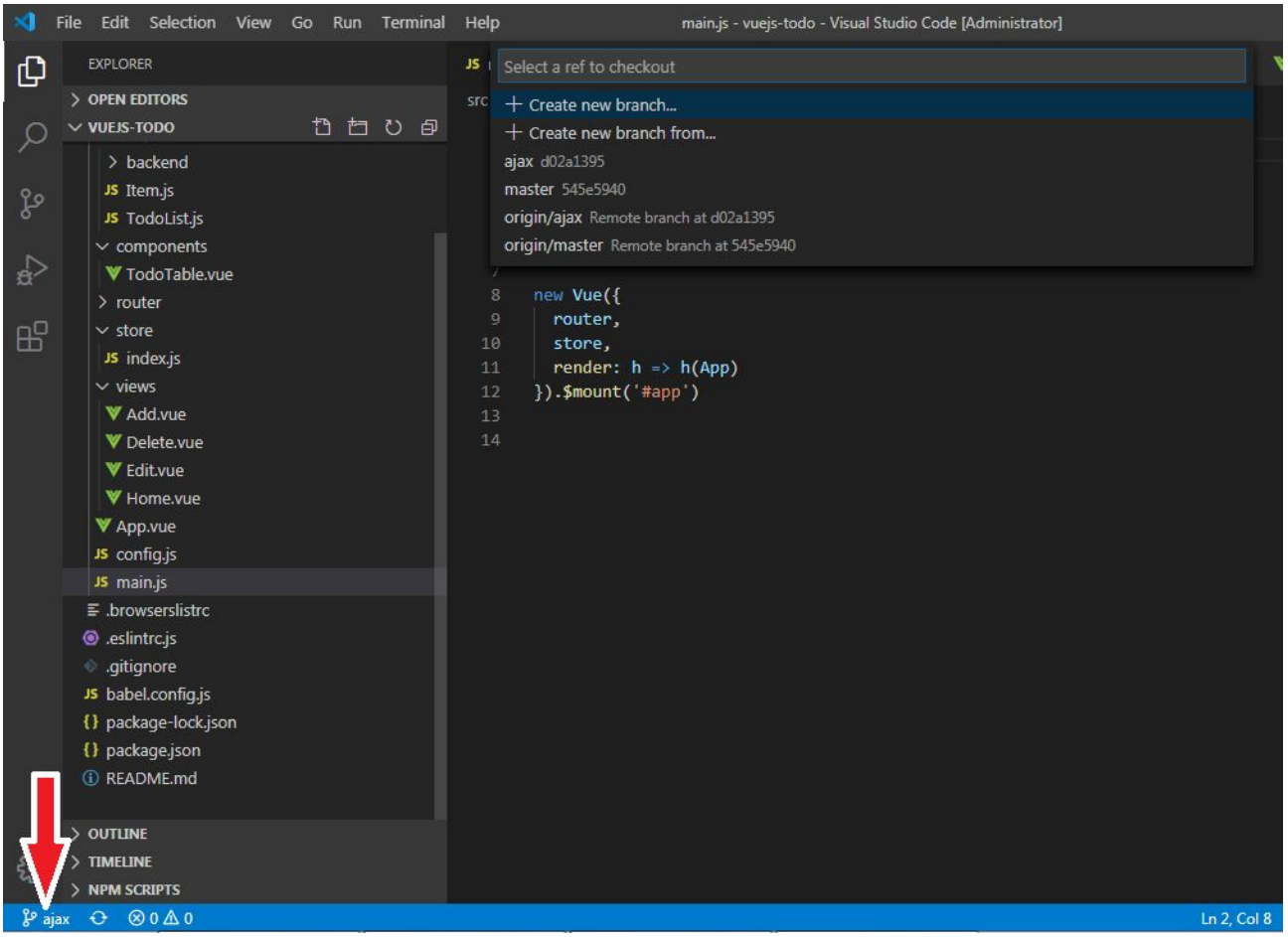
Перша команда інсталює необхідні пакети, а друга запустить додаток на виконання.

**Дві версії додатка**

Додаток “Список справ” реалізовано у двох версіях з однаковим інтерфейсом користувача, але з різним способом збереження даних:

- у пам’яті за допомогою списків та об’єктів javascript (гілка репозиторію **`master`**);
- на сервері у базі даних SQLite (гілка репозиторію **ajax**). Серверна реалізація - репозиторій з кодом на Python/Django.

Для переходу з гілки на гілку репозиторія у середовищі VS Code необхідно використовувати перемикач у лівому нижньому куті екрану як на рисунку нижче:




**Можливості VueJS**

VueJS є Web-фреймворком, призначеним для створення так званих односторінкових додатків або ж для створення клієнтського Web-інтерфейсу користувача складних додатків. Особливостями VueJS є:

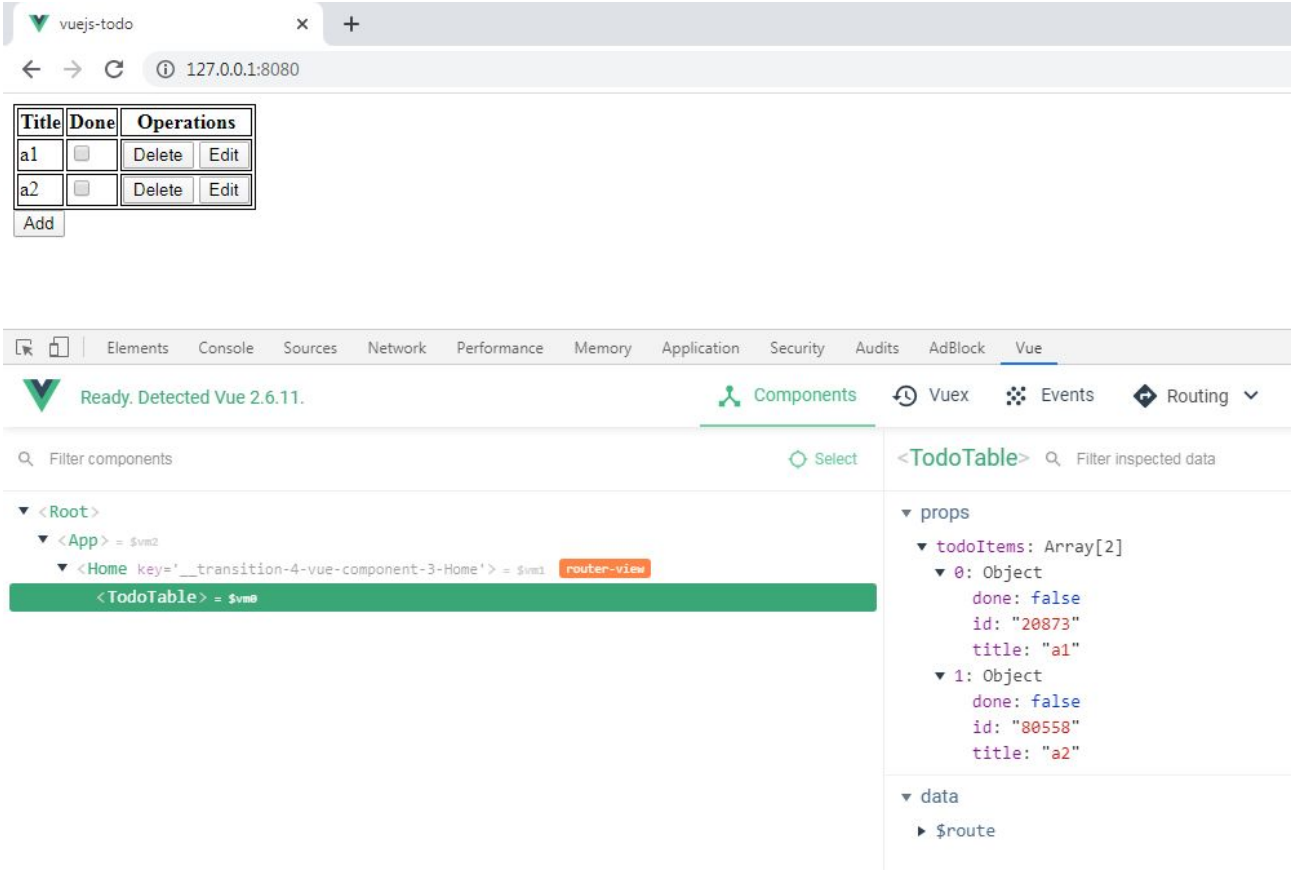
1. Використання шаблону MVC у вигляді MVVM з орієнтацією лише на презентаційний (View) аспект додатка.
2. Абстрагування від безпосереднього використання DOM Web-сторінки, завдяки реактивним об'єктам Javascript, тобто можливістю управління виведенням/введенням даних на сторінці, уникаючи безпосереднє звертання до елементів DOM-моделі.
3. Компонентна модель додатку, що дозволяє інкапсулювати та повторно використовувати розроблений код.

4. Відкрита архітектура фреймворку, що дозволяє інтегрувати сторонні бібліотеки (зокрема, для роботи з Ajax-запитами).
5. Відносна простота оволодіння та інші властивості (див. офіційну документацію).

**“Екосистема“ VueJS**

Так звана екосистема VueJS, подібно до інших популярних Web-фреймворків (React або Angular), базується на використанні серверного середовища розробки NodeJS, а також включає наступні інструменти:

➔ Інструмент розробника **vue-devtools** (https://github.com/vuejs/vue-devtools), що дозволяє відлагоджувати додатки у браузерах Google Chrome та Mozilla Firefox:



### ➔ Стандартний інструментарій для розробки на Vue.js (Vue-CLI)

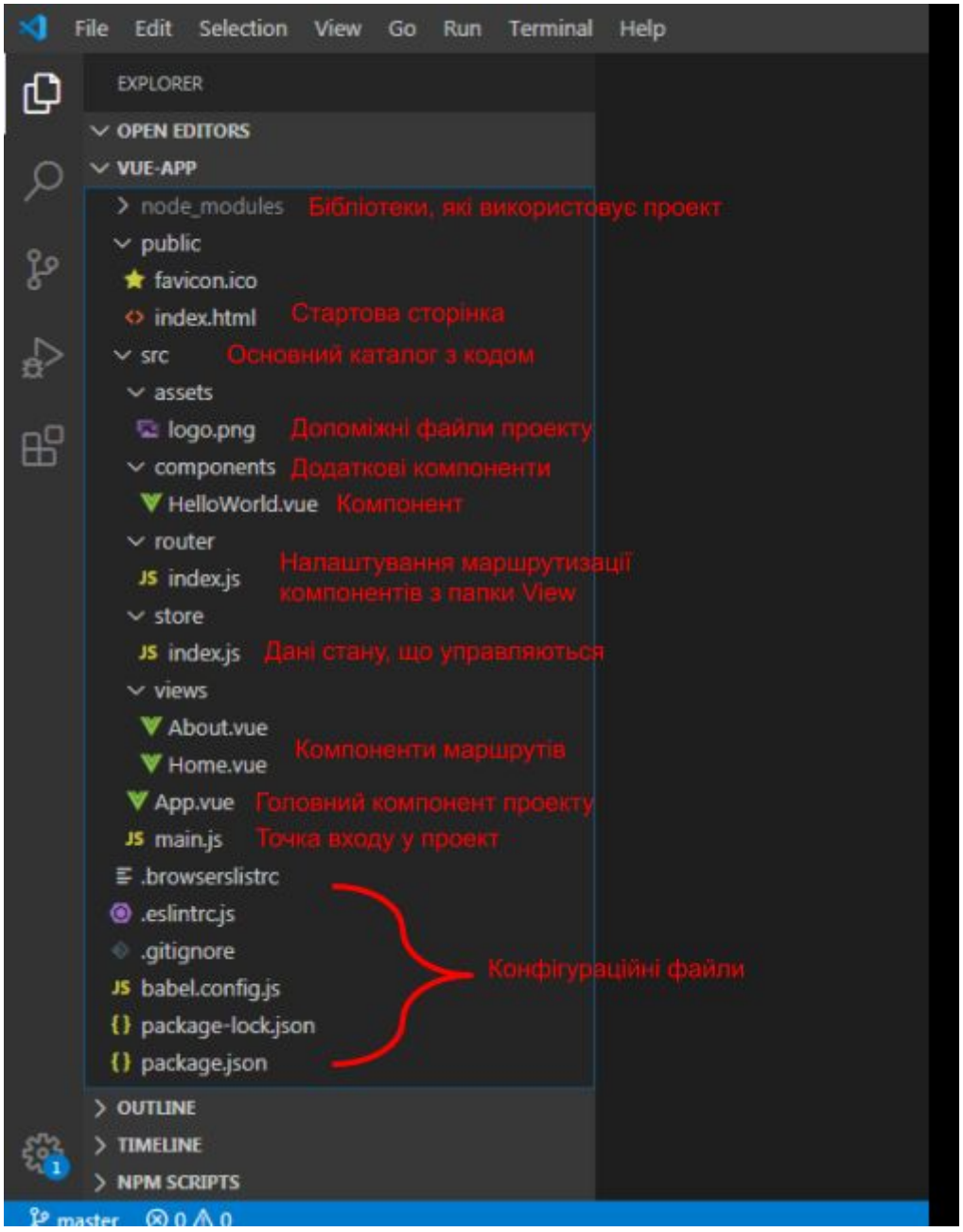
Vue-CLI - набір утиліт, що дозволяє створювати додатки VueJS та конфігурувати різноманітні бібліотеки, що використовує додаток, зокрема: пакування усіх файлів проекта в єдиний файл, мінімізація файлів, підтримка стилевих мов (SCSS, SASS), засоби контролю синтаксису (“лінтинг”), засоби тестування коду, а також підтримки старих версій Javascript та інші можливості.

**Створення та запуск додатку VueJS**

Для створення додатків VueJS необхідно виконати наступні кроки:

1. Встановити платформу NodeJS з менеджером пакетів NPM.
2. Встановити VueJS та Vue-Cli командою `npm install -g @vue/cli`.
3. Створити новий проект командою vue create `my-vue-app`, де `my-vue-app` - назва проекту.
4. Обрати опціональні бібліотеки для проекта: **Babel** (підтримка старих браузерів, **Router** (маршрути сторінок проекта), **Vuex** (централізоване управління станом - даними проекта) та **Linter/Formatter** (контроль синтаксису/форматування коду). Решту налаштувань залишити за замовчуванням.
5. Запуск додатку: `cd vue-app` та `npm run serve`.

**Файлова структура проекта VueJS**




**Алгоритм виконання додатка VueJS**

1. Користувач у браузері вводить рядок URL, наприклад http://127.0.0.1:8080/
2. Завантажується сторінка `index.html`, в якій є посилання на `app.js` (тимчасовий файл, у якому зібрано увесь код проекту - JS, CSS, HTML).

2.1. Якщо розібрати `app.js` на складові, то спочатку виконується файл `main.js` проекту, де створюється екземпляр Vue, якому передається головний компонент App.Vue:

```
JS main.js
src > JS main.js > ...
1   import Vue from 'vue'
2   import App from './App.vue'
3   import router from './router'
4   import store from './store'
5
6   Vue.config.productionTip = false
7
8   new Vue({
9     router,
10    store,
11    render: h => h(App)
12  }).$mount('#app')
13
```

2.2. Файл App.vue має дві секції: `<template>` - шаблон HTML-сторінки, куди підставляються дані у процесі виконання коду, а також секція `<style>` - стилі CSS.

```
App.vue
src > App.vue > template
1   <template>
2     <div id="app">
3       <div id="nav">
4         <router-link to="/">Home</router-link> |
5         <router-link to="/about">About</router-link>
6       </div>
7       <router-view/>
8     </div>
9   </template>
10
11  <style>
12  #app {
13    font-family: Avenir, Helvetica, Arial, sans-serif;
14    -webkit-font-smoothing: antialiased;
15    -moz-osx-font-smoothing: grayscale;
16    text-align: center;
17    color: #2c3e50;
```

Важливою є директива <router-view>, яка в залежності від поточного URL та налаштувань файла `router/index.js` визначає який компонент відобразити в даний момент.

Файл `router/index.js` виглядає так:

index.js

src > router > index.js > ...

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/Home.vue'

Vue.use(VueRouter)

  const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/about',
    name: 'About',
    // route level code-splitting
    // this generates a separate chunk (about.[hash].js) for this route
    // which is lazy-loaded when the route is visited.
    component: () => import(/* webpackChunkName: "about" */ '../views/About.vue')
  }
]
```

де вказано, що на посилання **/** (корінь сайта) треба відобразити компонент `Home` з файлу `/views/Home.vue`, код якого має вигляд:

Home.vue

src > views > Home.vue > {} "Home.vue" > template

```vue
<template>
  <div class="home">
    <img alt="Vue logo" src="../assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>

<script>
// @ is an alias to /src
import HelloWorld from '@/components/HelloWorld.vue'

export default {
  name: 'Home',
  components: {
    HelloWorld
  }
}
</script>
```

Код файла `Home.vue` містить дві секції: шаблон `<template>` та `<script>` - JS-код компонента. В секції код імпортується додатковий компонент `HelloWorld.vue`, якому передається параметр **msg** з константним значенням “Welcome to your Vue.js app”. У свою чергу файл `HelloWorld.vue` приймає властивість **msg** та відображає її серед інших елементів HTML (див. HelloWorld.vue):

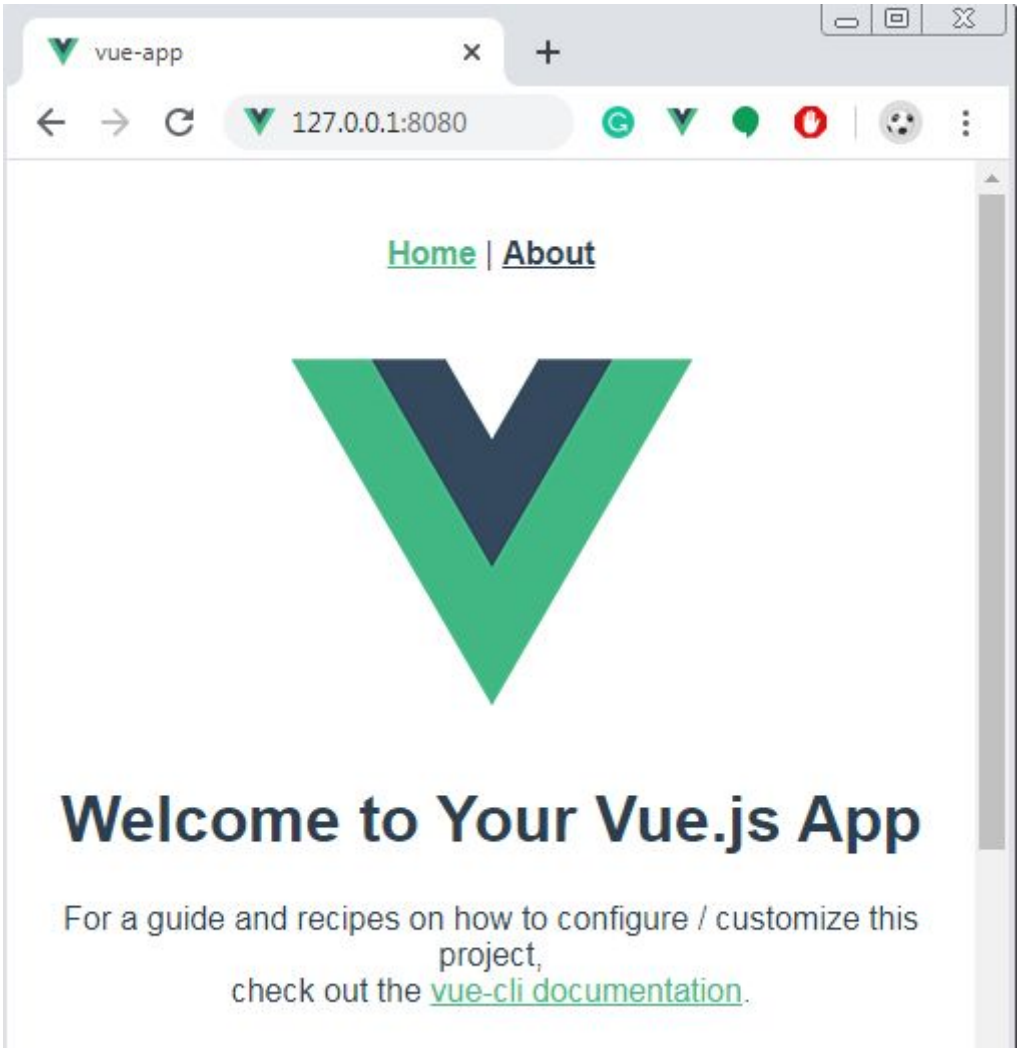
HelloWorld.vue

src > components > HelloWorld.vue > {} "HelloWorld.vue" > template

```vue
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
    <p>
      For a guide and recipes on how to configure / customize this project,<br>
      check out the
      <a href="https://cli.vuejs.org" target="_blank" rel="noopener">vue-cli documentation</a>.
    </p>
    <h3>Installed CLI Plugins</h3>
```

```
HelloWorld.vue
src > components > HelloWorld.vue > {} "HelloWorld.vue" > script > props
      <li><a href="https://github.com/vuejs/awesome-vue" target="_bla
    </ul>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>
```

Кінцеве виведення виглядає наступним чином:



**Створення власного додатка**

Для створення власного додатка можна використовувати щойно розглянутий із наступними модифікаціями:

1. Вилучити зайві компоненти.
2. Змінити файл /router/index.js, додаючи до нього необхідні посилання на нові компоненти (наприклад, як у файлі маршрутизації додатку “Список справ”).

## Особливості реалізації додатка “Список справ на VueJS”

Як було зазначено вище, додаток “Список справ” реалізовано у двох версіях: збереження даних у пам’яті та на сервері. Розглянемо перший варіант, який не вимагає встановлення серверної частини додатка.

Розгляд додатка VueJS варто починати з файлу маршрутизації `router/index.js`:

src > router > JS index.js > ...

```
import Vue from 'vue'
import VueRouter from 'vue-router'
import Home from '../views/Home.vue'
import Add from '../views/Add.vue'
import Edit from '../views/Edit.vue'
import Delete from '../views/Delete.vue'

Vue.use(VueRouter)

  const routes = [
  {
    path: '/',
    name: 'Home',
    component: Home
  },
  {
    path: '/add',
    name: 'Add',
    component: Add
  },
  {
    path: '/edit',
    name: 'Edit',
    component: Edit,
    props: true // important to allow receiving props in Edit component
  },
  {
    path: '/delete',
    name: 'Delete',
    component: Delete,
    props: true // important to allow receiving props in Delete component
  }
]
```

Налаштування складається із чотирьох традиційних операцій: Add, Edit, Delete та Home. Операція перемикання прапорця виконання справи не має власної сторінки (виконується головному вікні), тому у роутері не записується. Кожна секція складається зі шляху (**path**), який з'являється у рядку браузера, назви компонента (**name**), об'єкта компонента (**component**) та **props** - у найпростішому випадку прапорець true/false (чи передавати дані у компонент, чи - ні). Таким чином, при переході між посиланнями, виконується відображення нового компонента.

Компонент `Home.vue` відіграє роль головної сторінки та передає управління при виникненні певних подій (додавання, вилучення тощо).

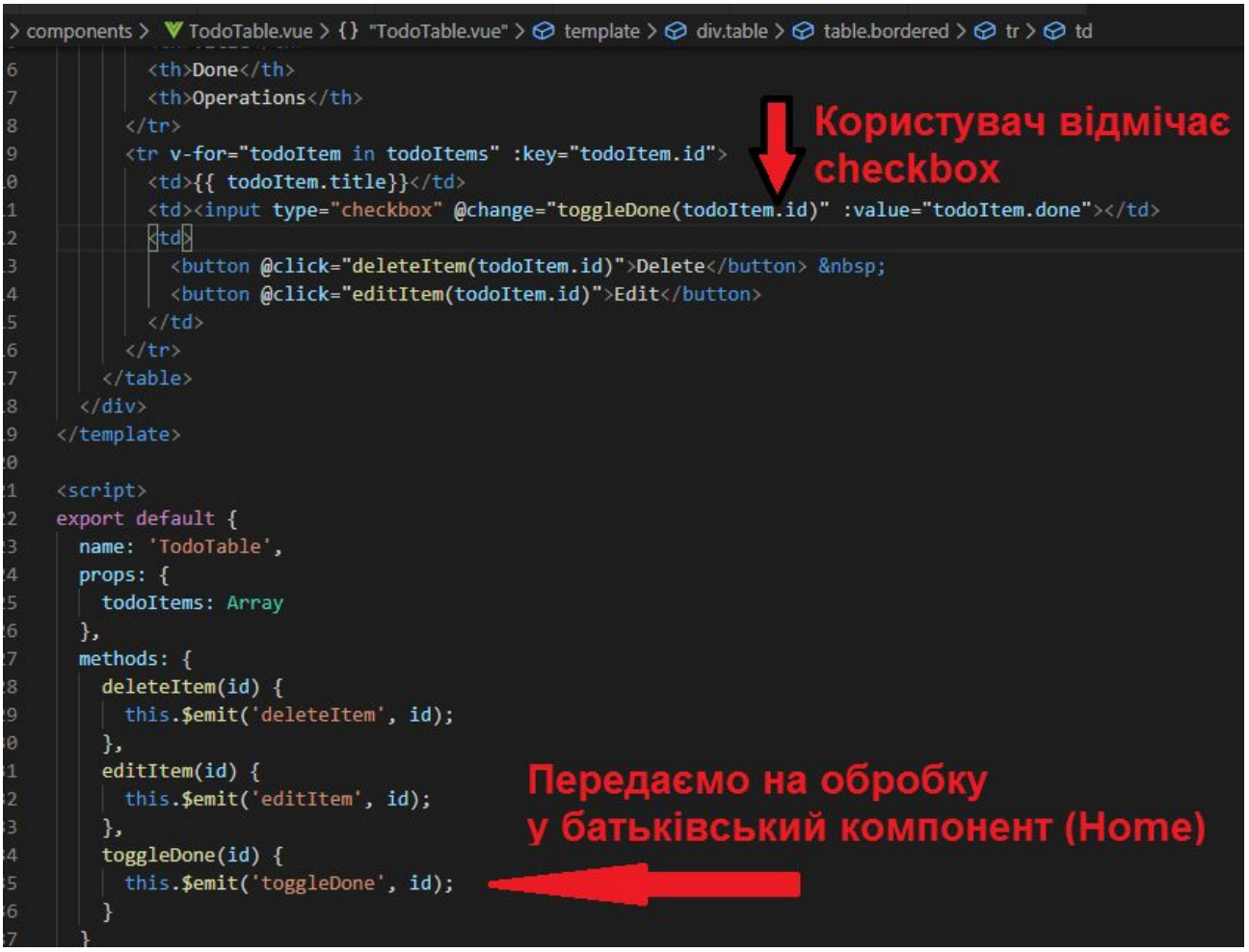
```
<TodoTable :todoItems="$store.state.todoItems.items"

  @toggleDone="toggleDone" @deleteItem="deleteItem" @editItem="editItem"/>

<button @click="add">Add</button>
```

Даний код шаблону підключає компонент Todo для виведення таблиці зі списком справ. Цікавим є спосіб передачі оброблювача подій: спеціальний символ @ означає, що далі йде назва події, а у лапках - код Javascript, який буде викликано за умови виникнення події. Події можуть бути як стандартними, наприклад: @click, або тими, що генеруються користувачем. Наприклад, коли виникає подія @toggleDone, тоді буде викликано відповідну функцію toggleDone. Генерує цю подію користувач за допомогою компонента TodoTable (`src/components/TodoTable.vue`):

Спеціальна команда `this.$emit('toggleDone', id);` передає на обробку подію `'toggleDone'` зі значенням **id** - номером справи для зміни стану.

Такий підхід є стандартним для VueJS:

| |
|---|
| Дані від контейнерного (батьківського) компонента передаються у дочірній через props, а від дочірнього до батьківського - у вигляді події. Безпосередньо у дочірньому компоненті props змінювати заборонено. |




**Управління станом додатка - Vuex**

Стан додатка - це об’єкти, масиви та інші змінні, які мають бути доступними усіма компонентами. Для того, щоб не виникало конфліктів доступу на ***зміну значень*** до них, виділяють спеціальний файл `/store/index.js`, де зберігають дані, а також реалізують функції зміни даних, так звані “мутатори” (mutators), так дії (actions), що дозволяють викликати мутатори як синхронно, так і асинхронно. Асинхронний спосіб важливий при доступі до даних, що знаходяться на сервері. Доступ ***на читання*** дозволено безпосередньо у вигляді **$this.store.state.myObject,** хоча рекомендованим способом є використання так званих getters. У даному додатку файл стану (`/store/index.js`) виглядає наступним чином:

```
store > JS index.js > ...
import Vue from 'vue'
import Vuex from 'vuex'
import TodoList from '@/cls/model/TodoList.js'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    todoItems: new TodoList()
  },
  mutations: { // cannot be asynchronous! (always sync)
    ADD_TODO: (state, todoItem) => {
      state.todoItems.addItem(todoItem);
    },
    TOGGLE_DONE_TODO: (state, todoItemId) => {
      state.todoItems.toggleDone([todoItemId]);
    },
    DELETE_ITEM_TODO: (state, todoItemId) => {
      state.todoItems.deleteItem([todoItemId]);
    },
    EDIT_ITEM_TODO: (state, item) => {
      state.todoItems.editItem(item.itemId, item.itemTitle);
    }
  },
  actions: { // can be asynchronous!
    ADD_TODO: (context, todoItem) => {
      context.commit('ADD_TODO', todoItem);
    },
    TOGGLE_DONE_TODO: (context, todoItemId) => {
      context.commit('TOGGLE_DONE_TODO', todoItemId);
    },
```

Станом тут виступає об’єкт класу TodoList(), який відіграє роль моделі (див. ЛР№2) і приховує логіку роботи зі списком справ. Наприклад, щоб додати елемент до списку, виконується наступна послідовність дій:

1. Компонент Add в одному зі своїх методів (оброблювача події натискання кнопки “Add”) викликає дію (action) ADD_TODO:

```
this.$store.dispatch('ADD_TODO', this.itemTitle);
```

2. Дія ADD_TODO у файлі `/store/index.js` в свою чергу викликає однойменний мутатор:

```
ADD_TODO: (state, todoItem) => {

  state.todoItems.addItem(todoItem);

},
```

3. Нарешті, мутатор викликає відповідний метод **addItem** об'єкта **state.todoItems**.

   Додатковий крок з діями виглядає зайвим, але без нього неможливо обійтись у випадку асинхронних дій. Тестовий приклад асинхронної дії наведено для операції EDIT_ITEM_TODO, коли її виконання відбувається за 3 секунди. Інші приклади реалізовано у клієнт/серверній версії додатка (див. https://github.com/Aroxed/vuejs-todo/tree/ajax/src).

**Класи моделей даних**

Класи для обробки даних - моделей реалізовано аналогічним чином, як для приклада лабораторної роботи №2. У ідеальному випадку перехід з ЛР№2 до ЛР№3 має відбуватись з мінімальними змінами класів моделі, оскільки фокус фреймворка VueJS орієнтовано на подання, реалізація моделі залишається на програмісті. У даному проекті ці файли розташовано за шляхами: `src\cls\model\TodoList.js` та `src\cls\model\Item.js`.

## Варіанти завдання

Роботу виконувати самостійно. Номер варіанта завдання співпадає з відповідним номером у лабораторній роботі №2.

## Вимоги до оформлення лабораторної роботи у електронному вигляді

Звіт про лабораторну роботу в репозиторії GitHub включає: назву лабораторної роботи, варіант студента, текст завдання, 2-3 копії екранних форм (screenshots), а **також відповіді на контрольні запитання**.

## Контрольні запитання

1. Пояснити переваги використання клієнтських Web-фрейморків.
2. Пояснити складові додатку VueJS (components, data, computed та інші).
3. Пояснити спосіб використання бібліотеки Vuex у складі Vue.