# Лабораторная работа 6. МОДУЛЬ ПРИЛОЖЕНИЯ «ПОКУПКА И ПРОДАЖА АКЦИЙ»

**Цель и задачи**

Целью работы является изучение возможностей применения фреймвор- ка Vue (https://v3.ru.vuejs.org/ru/) для разработки интерфейсов пользователя web-приложений и организации E2E тестирования клиентской части прило- жения. Для достижения поставленной цели требуется решить следующие за- дачи:

* + разработка интерфейса web-приложения;
  + создание web-сервера на основе NestJS. Подготовка web-сокетов для обновления информации о стоимости у всех клиентов;
  + создание каркаса web-приложения с использованием Vue;
  + разработка перечня компонентов;
  + создание статической версии интерфейса;
  + программирование потока изменения состояний web-приложения;
  + разработка скрипта автоматического тестирования web-приложения.

# Основные теоретические сведения

Vue – фреймворк для создания пользовательских интерфейсов. В отли- чие от фреймворков-монолитов, Vue создавался пригодным для постепенно- го внедрения. Его ядро решает задачи уровня представления, упрощая инте- грацию с другими библиотеками и существующими проектами. С другой стороны, Vue подходит и для разработки сложных одностраничных прило- жений.

Selenium – фреймворк тестирования web-приложений, позволяет про- граммировать автоматизированные тесты клиентской части web-приложений, а также записывать и воспроизводить действия пользователей.

Headless-браузер – браузер без интерфейса пользователя, предназначен- ный для тестирования web-приложений.

# Общая формулировка задачи

Необходимо создать web-приложение, обеспечивающее работу брокера, у него есть запас денежных средств, он имеет возможность купить или про- дать акции (любое доступное количество), а также контролировать измене- ние котировок акций. В приложении должен отображаться баланс (запас де- нежных средств плюс стоимость акций), а также прибыль или убыток, кото- рые он получил по каждой акции. Основные требования следующие:

1. Приложение получает исходные данные из модуля администрирова- ния приложения «Биржа акций» в виде настроек в формате JSON-файла и в виде данных от web-сокета по изменению стоимости акций во времени.
2. В качестве сервера используется NestJS.
3. Участники торгов подключаются к приложению «Покупка и продажа акций».
4. Предусмотрена HTML-страница администратора, на которой отобра- жается перечень участников. Для каждого участника отображается его ба- ланс, количество акций каждого типа у каждого участника и его прибыль или убыток по каждой акции в текущий момент времени.
5. Предусмотрена HTML-страница входа в приложение, где каждый участник указывает (или выбирает из допустимых) свое имя.
6. Предусмотрена HTML-страница, на которой участнику отображаются:
   * текущая имитируемая дата;
   * текущая стоимость каждой из акций, выставленных на торги;
   * общее количество доступных средств;
   * количество, стоимость и прибыль/убыток по каждой купленной акции. На этой же странице у брокера есть возможность:
   * открыть диалоговое окно просмотра графика изменения цены каждой акции (с момента начала торгов до текущего момента) с учётом сообщений об изменении стоимости акций;
   * купить/продать интересующее его количество акций. Комментарии:
   * брокер не может купить акции, если денег не хватает;
   * купля/продажа происходит «мгновенно».
7. Разработаны автоматизированные тесты для проверки корректности работы клиентской части web-приложения с использованием headless- браузера или фреймворка Selenium. Как минимум необходимо проверить, что при покупке/продаже N акций в определённую дату соответствующим обра- зом изменяется баланс средств брокера и через некоторое время получается правильная прибыль/убыток по данной акции.

Преимуществом будет использовать Material Design Framework (https://vuetifyjs.com/en).

# Описание ключевых методов при выполнении работы

***Создание каркаса web-приложения с использованием Vue****.* Для созда- ния нового Vue приложения можно установить глобально модуль «@vue/cli» и воспользоваться следующей командой:

vue create vue-app

Здесь «vue-app» – название создаваемого Vue-приложения.

Для запуска приложения необходимо выполнить следующую команду:

npm run start

Приложение запустится по адресу http://localhost:8080/.

Основной файл приложения «src/main.js», основной компонент –

«src/App.vue». В простейшем случае основной файл может выглядеть следу- ющим образом.

**import** { ***createApp*** } **from 'vue' import** App **from './App.vue' *createApp***(App).mount(**'#app'**)

Здесь создаваемый компонент будет отображён на основной странице index.html в элементе с идентификатором «app».

***Компоненты Vue****.* Компонент Vue – это «фрагмент кода», который представляет собой часть веб-страницы. Создание простейшего компонента («src/App.vue»):

<**template**>

<**div**>

<**h3**>{{msg}}</**h3**>

</**div**>

</**template**>

<**script**>

**export default** {

**name**: **'App'**,

data() { **return** { **msg**: **"Привет Vue"** } }

</**script**>

<**style scoped**>

**h3** { **color**: **green**; }

</**style**>

В данном случае на web-странице отобразится простейшее приветствие

«Привет Vue». Следует обратить внимание, что компонент Vue состоит из трёх частей – шаблона (template), JavaScript-кода (script) и CSS-настроек (style). Параметры компонента из JavaScipt могут отображаться в шаблоне с использованием привязки. Для привязки в теле элемента используются двой- ные фигурные скобки, например, {{msg}}. В теле элемента в двойных фи- гурных скобках может использовать одно JavaScript выражение, возвращаю- щее значение для отображения. В шаблоне должен находиться один HTML- элемент, который может объединять в себе множество других HTML- элементов.

Vue-компоненты могут содержать свои свойства (data), получать свой- ства из родительского компонента (props), вызывать функции (methods), со- держать вычислимые поля (computed), хуки жизненного цикла компонента и т.п.

Предположим, что родительский компонент называется «App.vue».

<**template**>

<**HelloWorld msg="Первое Vue3 приложение"**/>

</**template**>

<**script**>

**import** HelloWorld **from './components/HelloWorld.vue' export default** {

**name**: **'App'**,

**components**: { HelloWorld }

}

</**script**>

В данном примере «App.vue» создаётся дочерний компонент «compo- nents/HelloWorld.vue», в него передаётся параметр msg. Рассмотрим пример дочернего компонента.

<**template**>

<**div**>

<**h3 v-bind:title="**text**"**>{{msg}}</**h3**>

<**button v-on:click="**reverse**"**>Перевернуть</**button**>

</**div**>

</**template**>

<**script**>

**export default** {

**name**: **'HelloWorld'**, **props**: { **msg**: **String** },

data() { **return** { **text**: **"Тестовый текст"** } },

**methods**: { // Методы компонента reverse() {

**this**.**text** = **this**.**text**.split(**''**).reverse().join(**''**)

}

}

}

</**script**>

В данном примере «HelloWorld.vue» получает строковый параметр

«msg» из родительского компонента, отображая его в теле элемента <h3>, в качестве подсказки (title) у этого же элемента используется свойство компо- нента «text». Привязка к свойству осуществляться с помощью директивы «v- bind:» (альтернативное обозначение «:»). Компонент также отображает кноп- ку (button), к которому с помощью директивы «v-on:» (альтернативное обо- значение «@») на событие «click» подключен метод компонента «reverse». Указанный метод переворачивает текст в свойстве «text». Обратите внима- ние, что обращение к свойству осуществляется с использованием указателя

«this».

Привязка с использованием директивы «v-bind:» является односторон- ней, для двусторонней привязки используется директива «v-model».

<**input v-model="**question**"**/>

В приведённом примере «question» должен быть объявлен как свойство компонента, тогда его изменение будет отображаться в <input> и наоборот.

Как и в большинстве фреймворков Vue поддерживает передачу свойств компонентам-потомкам и в обратную сторону – передачу событий. Рассмот- рим пример компонента-родителя «App.vue».

<**template**>

<**div**>

<**Button btn-label="**Сч. A**" idx="**0**" @inc-value="**increment**"**>

</**Button**>

<**Button btn-label="**Сч. B**" idx="**1**" @inc-value="**increment**"**>

</**Button**>

<**div**>Счётчики: A={{**counter**[0]}}, B={{**counter**[1]}}</**div**>

</**div**>

</**template**>

<**script**>

**import** Button **from "@/components/Button"**; **export default** {

**name**: **'HelloWorld'**, **components**: {Button},

data() { **return** { **counter**: [0, 0] } },

**methods**: { increment(index) { **this**.**counter**[index]++ } }

}

</**script**>

В данном примере созданы два компонента «Button», каждому из кото- рых передано имя «btn-label», в дочернем компоненте оно трансформируется по правилу CamelCase в «btnLabel», передан индекс «idx» в массиве

«counter», указана функция, которая будет обрабатывать события, получае- мые от дочернего компонента «inc-value», в дочернем компоненте трансфор- мируется – в «incValue». При поступлении событий от дочернего компонента будет вызываться функция «increment», ожидается, что в качестве параметра у этой функции будет указан индекс в массиве «counter». Значения «counter» отображаются в соответствующем <div> в компоненте.

Рассмотрим пример «Button.vue».

<**template**>

<**button @click="$emit**(**'**incValue**'**, idx)**"**>

{{ btnLabel }}

</**button**>

</**template**>

<**script**>

**export default** {

**name**: **"Button"**,

**props**: [**"**btnLabel**"**, **"**idx**"**]

}

</**script**>

Компонент «Button» получает от родительского компонента два свой- ства «btnLabel» и «idx». Значение «btnLabel» отображается в качестве текста кнопки <button>, значение «idx» передаётся в качестве параметра генерируе- мого события EventEmitter с помощью «$emit», где «incValue» – имя создава- емого события, после имени через запятую перечисляются параметры, пере- даваемые с событием.

***Маршрутизация в Vue****.* Для маршрутизации потребуется дополнитель- ный модуль «vue-router». При создании приложения в «main.js» необходимо подключить маршруты.

**import** { **createApp** } **from 'vue' import** App **from './App.vue' import router from './router'**

**createApp**(App).use(**router**).mount(**'#app'**)

В данном примере маршруты загружаются из файла «router.js». **import** { *createRouter*, *createWebHashHistory* } **from 'vue-router' import** Home **from '../views/Home.vue'**

**const** routes = [

{ **path**: **'/'**, **name**: **'Home'**, **component**: Home },

{ **path**: **'/about'**, **name**: **'About'**,

*// Создаст компонент, lazy-подгружаемый при первом обращении*

component: () => **import**(**'../views/About.vue'**)

}

]

**const *router*** = *createRouter*({ **history**: *createWebHashHistory*(), routes

})

**export default** router

В приведённом примере предусмотрено два пути «/», обрабатываемый компонентом «Home» и «/about», обрабатываемый компонентом «About». Последний загружается в память браузера при первом обращении к компо- ненту.

Для корректного отображения маршрутов и ссылок переходов между компонентами необходимы соответствующие изменения в «App.vue».

<**template**>

<**div id="nav"**>

<**router-link to="/"**>Home</**router-link**> |

<**router-link to="/about"**>About</**router-link**>

</**div**>

<**router-view**/>

</**template**>

В данном случае с использованием «router-link» создаются две ссылки на странице с переходами на «/» и «/about». Результаты нажатия на ссылки будут отображаться в «router-view».

Для обработки параметров маршрута, например, в «About.vue» можно воспользоваться следующим примером.

<**template**>

<**div**>

<**div**>params={{ $route.params }}</**div**>

<**div**>query={{ $route.query }}</**div**>

<**div**>path={{ $route.path }}</**div**>

<**div**>id={{ $route.params.id }}</**div**>

</**div**>

</**template**>

Здесь «$route.params» перечислит имена и значения всех полученных параметров в URL, «$route.query» перечислит имена и значения всех полу- ченных параметров запроса, «$route.path» вернёт полный путь запроса, а

«$route.params.id» вернёт значение единственного параметра URL по имени

«id». Следует отметить, что для корректной работы с параметрами URL соот- ветствующие изменения должны быть внесены в описания маршрута в «rout- er.js», в частности строка «about» для обработки «id» должна измениться сле- дующим образом.

{ **path**: **'**/about/**:id'**, **name**: 'About', **component**: About }

***Управление состоянием приложения****.* Vuex (https://next.vuex.vuejs.org/) представляет собой контейнер для управления состоянием web-приложения. Для работы с единым состоянием приложения его необходимо подключить в

«main.js» (можно последовательно с маршрутизатором).

**import** { **createApp** } **from 'vue' import** App **from './App.vue' import** router **from './router' import store from './store'**

**createApp**(App).use(**store**).use(router).mount(**'#app'**)

В данном примере единое состояние приложения описано в «store.js».

**import** { createStore } **from 'vuex' export default** createStore({

state () { **return** { **count**: 0 } },

**mutations**: {

increment (state) { state.**count**++ }

},

**getters**: { // Вычисление значения pow2(state, getters) {

**return Math**.pow(state.**count**, 2)

}, // С параметром

pow: (state) => (p) => {

**return Math**.pow(state.**count**, p)

}

}

})

Состояние приложения «state» содержит счётчик «count», имеет два

идентичных метода получения значения «getters»: «pow» и «pow2», отлича- ющиеся тем, что в первом значение параметра возведения в степень фикси- ровано числом «2», а во втором – передаётся в переменной «p». Функция увеличения «increment» позволяет изменить состояние приложения.

Компонент, использующий общее состояние приложения может выгля- деть следующим образом.

<**template**>

<**div**>

<**button @click="**increment**"**>Inc</**button**>

<**div**>{{**count**}}^2 = {{**power2**}}</**div**>

<**div**>{{**count**}}^3 = {{**power3**}}</**div**>

</**div**>

</**template**>

<**script**>

**import** {***mapState***} **from "vuex"**; **export default** {

**methods**: {

increment() { *// Увеличение значения*

**this**.**$store**.commit(**'increment'**)

}

},

**computed**: {

power2() { *// Вычисляемой свойство*

**return this**.**$store**.**getters**.pow2

},

power3() {

**return this**.**$store**.**getters**.pow(3)

},*// Подключение count*

...***mapState***([**"count"**])

}

}

</**script**>

По нажатию на кнопку «Inc» вызывается функция «increment», с помо- щью «this.$store.commit()» вызывается функция общего хранилища, обраще- ние на чтение к функциям хранилища осуществляется с использованием

«this.$store.getters», текущее значение «count» получается с помощью функ- ции «mapState» из Vuex. Важно, что функции получения значения из храни- лища находятся в разделе «computed» компонента, т.е. они вычисляются и при изменении состояния Vuex автоматически обновляются.

Следует обратить внимание, что методы «mutations» предназначены для синхронных изменений. Для асинхронных изменений (например, с обраще- нием к web-серверу) необходимо использовать «actions» (в «store.js»). **mutations**: { // Синхронное обновление состояния

increment(state, obj) { state.**count** += obj.**amount**

}

},

**actions**: { // Асинхронное обновление состояния inc(context, obj) {

context.**commit**(**'increment'**, obj)

}

}

В компоненте следует использовать «dispatch()» для обращения к «ac-

tions».

**methods**: {

increment() { // Асинхронное уеличение значения

**this**.**$store**.dispatch(**'inc'**, {**amount**: 3})

}

}

## Автоматизация тестирования с помощью Selenium. Selenium

(https://[www.selenium.dev/)](http://www.selenium.dev/)) представляет собой фреймворк для тестирования web-приложений. Для корректной работы необходимо установить интерфейс между Selenium и соответствующим браузером: https://[www.npmjs.com/package/selenium-webdriver.](http://www.npmjs.com/package/selenium-webdriver) Важно, что браузер дол- жен быть установлен глобально, а соответствующий файл драйвера помещен в папку, указанную в переменной PATH.

Пример автоматизированного теста для тестирования с использованием браузера «Google Chrome».

**const** {Builder, **By**, **Key**, until} = require(**'selenium-webdriver'**); (**async function** *example*() {

**let** driver = **await new** Builder().forBrowser(**'chrome'**).build();

**try** {

*// Перейти по URL*

**await** driver.get(**'https://www.yandex.ru'**);

*// Ввести текст "cheese" и нажать на клавиатуре "Enter"*

**await** driver.findElement(**By**.*name*(**'text'**)).sendKeys(**'cheese'**,

**Key**.**ENTER**);

**let** firstResult = **await** driv- er.wait(until.elementLocated(**By**.*css*(**'.main content'**)), 10000);

***console***.log(**await** firstResult.getAttribute(**'textContent'**));

}

**finally**{

**await** driver.quit();

}

})();

В данном примере загружается браузер «chrome», осуществляется пере- ход по адресу «https://[www.yandex.ru](http://www.yandex.ru/)», в поле с именем «text» вводится слово

«cheese», ищется компонент с классом «main content» и в консоль выводит- ся значение «textContent».

При использовании Selenium браузер будет запущен и можно наблюдать все выполняемые операции.

***Автоматизация тестирования с использованием headless-браузеров****.* Существует множество headless-браузеров, рассмотрим применение Google Puppeteer (https://developers.google.com/web/tools/puppeteer/) для тестирования клиентской части web-приложения с использованием Jest.

**const** puppeteer = require(**"puppeteer"**)

**test**(**"Headless test"**, **async** () => {

**const** browser = **await** puppeteer.launch({

**headless**: **true**

}) // если поставить false, то запустится браузер

**const** page = **await** browser.newPage() **await** page.goto(**"http://localhost:3000"**) **await** page.waitForSelector(**".app-title"**)

**const** html = **await** page.$eval(**".app-title"**, e => e.**innerHTML**)

**expect**(html).toBe(**"Hello world"**)

}, 16000)

В данном примере загружается модуль «puppeteer», на основе которого создаётся браузер «browser», открывается новая страница «newPage()», осу- ществляется переход по адресу «http://localhost:3000», ожидается окончание формирования на странице компонента с классом «app-title», из которого в переменную html помещается свойство «innerHTML», затем выполняется сравнение с «Hello world».

# Вопросы для контроля

1. Из чего состоит компонент Vue?
2. Как осуществляется привязка параметров и свойств компонента к элементам, атрибутам и событиям в HTML?
3. Как передаётся информация между родительским и дочерним компо- нентами Vue?
4. Как обеспечить маршрутизацию и обработку параметров маршрута?
5. Какие основные понятия есть в Vuex? Как реализуется единое состоя- ние приложения?
6. Как внести изменения в состояние приложения Vuex (синхронно, асинхронно)?
7. Как создать автоматизированный тест клиентской части web- приложения?

# Дополнительные источники в сети Интернет

Обычно используются следующие источники:

* + Прогрессивный JavaScript-фреймворк // URL: https://v3.ru.vuejs.org/ru/;
  + Vue CLI // URL: https://cli.vuejs.org/ru/;
  + Vue Router // URL: https://next.router.vuejs.org/;
  + What is Vuex? // URL: https://next.vuex.vuejs.org/;
  + Material Design Framework // URL: https://vuetifyjs.com/en/;
  + Selenium automates browsers. That's it! // URL: https://[www.selenium.dev/;](http://www.selenium.dev/%3B)
  + selenium-webdriver // URL: https://[www.npmjs.com/package/selenium-](http://www.npmjs.com/package/selenium-) webdriver;
  + Puppeteer // URL: https://developers.google.com/web/tools/puppeteer/.

В результате выполнения всех лабораторных работ у студента должен сформироваться навык по созданию web-приложений с использованием со-

временных web-технологий. При этом студент на практике отработает те знания, которые он получил в лекционном материале.

Следует обратить внимание на оформление кода: должны в отдельных папках храниться ресурсы, доступные по прямой ссылке (рисунки, скрипты, файлы стилей и т.п.), защищаемые паролем страницы и не защищаемые.

После успешного выполнения всех лабораторных работ преподавателю необходимо предоставить:

1. архив проекта со всеми исходными кодами (без папки

«node\_modules»),

1. макеты страниц, созданные в Figma, если это требовалось заданием,
2. «скриншоты» основных страниц web-приложения,
3. отчеты по всем лабораторным работам.

# Содержание отчета

В отчете должны быть освещены следующие вопросы:

1. постановка задачи;
2. макеты интерфейса пользователя (созданные, например, с использо- ванием Figma, https://[www.figma.com/),](http://www.figma.com/)) если есть в задании;
3. описание тестов (модульных, интеграционных), если есть в задании;
4. описание наиболее важных фрагментов кода;
5. примеры экранных форм.

В отчете рекомендуется привести:

1. UML-диаграмму последовательности запросов и переходов между формами;
2. исходные коды лабораторной работы с комментариями.

# Список рекомендуемой литературы

Гоше Х. Д. HTML5 Для профессионалов. СПб.: Питер, 2015. Симпсон К. ES6 и не только. СПб.: Питер. 2017.

Стефанов С. React.js быстрый старт. СПб.: Питер, 2017. Фримен А. Angular для профессионалов. СПб.: Питер, 2018.

Шелли П. Изучаем Node. Переходим на сторону сервера. СПб.: Питер, 2017.

# Оглавление

[Лабораторная работа 1. Тетрис на JavaScript 3](#_bookmark0)

[Лабораторная работа 2. REST-приложение управления библиотекой 9](#_bookmark1)

[Лабораторная работа 3. Модуль администрирования приложения](#_bookmark2)

[«Социальная сеть» 21](#_bookmark2)

[Лабораторная работа 4. Модуль пользователя приложения «Социальная сеть»](#_bookmark3)

[................................................................................................................................. 35](#_bookmark3)

[Лабораторная работа 5. Модуль администрирования приложения «Биржа](#_bookmark4) [акций» 58](#_bookmark4)

[Лабораторная работа 6. Модуль приложения «Покупка и продажа акций» 70](#_bookmark5)

[Содержание отчета 82](#_bookmark6)

Беляев Сергей Алексеевич

# Web-технологии

Лабораторный практикум

Редактор М. Б. Шишкова

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Подписано в печать 12.03.2019. Формат 60×84 1/16.

Бумага офсетная. Печать цифровая. Печ. л. 5,0.

Гарнитура «Times New Roman». Тираж 83 экз. Заказ .

––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

Издательство СПбГЭТУ «ЛЭТИ» 197376, С.-Петербург, ул. Проф. Попова, 5