Лабораторная работа №1

Обзор платформы NodeJS и технологии AJAX

Цель работы

Познакомиться с процессом разработки веб-приложения на платформе NodeJS и с технологией AJAX.

Задачи:

1. Реализовать приложение на NodeJS с использованием модулей
2. Развернуть сервер с помощью Express.js
3. Создать HTML-страницу с функцией отправки AJAX-запроса к серверу
4. Развернуть приложение на Heroku

Подготовка:

В качестве IDE при выполнении работы рекомендуется использовать WebStorm. Студенческая лицензия WebStorm бесплатна и предоставляется при отправке скана студенческого билета. В качестве альтернативы могут быть использованы любые редакторы (Atom, Sublime или заточенный под фронтенд задачи Brackets от Adobe) в совокупности с командной строкой.

Для выполнения работы необходимо установить NodeJS с официального сайта. Платформа устанавливается вместе с менеджером пакетов npm. Для корректного выполнения работы необходимо, чтобы node и npm попали в PATH. При дефолтных параметрах установки это произойдет автоматически, в противном случае потребуется вручную указать путь к бинарным файлам. Проверить работоспособность команд можно выполнив в консоли команды

```
> node –v
> npm -v
```

Для корректной работы менеджера пакетов необходим Git. Установить клиент можно с официального сайта. Git также должен попасть в PATH при установке. Для наглядности рекомендуется использовать любой UI-клиент, например TortoiseGit. Проверить корректность установки можно командой

```
> git --version
```

Ход работы

1. Инициализация проекта

Работу необходимо выполнить в git-репозитории. Для начала инициализируем пустой репозиторий с помощью команды

```
> git init
```

или с помощью любого UI-клиента. В данном случае придется добавлять удаленные репозитории вручную. Чтобы этого избежать, можно создать пустой репозиторий на github,

bitbucket, gitlab или любом другом провайдере, получить на него ссылку и воспользоваться командой `git clone`.

Инициализация проекта производится командой

```
> npm init
```

Менеджер предложит ответить на несколько вопросов (название проекта, версия и т.д.), и в конце создаст файл метаданных package.json. Помимо прочего, в данном файле могут быть определены пользовательские скрипты, что понадобится при выполнении следующих работ.

Создадим файл index.js со следующим содержимым:

```
console.log("Hi! This is my first NodeJS app!");
```

И запустим его на выполнение простой командой

```
> node .
```

Как мы видим, наш первый скрипт выполнился. index.js – имя для дефолтного модуля, который будет запущен в текущей папке. Аналогичным образом может быть выполнен и любой другой модуль. Создадим файл myscript.js:

```
console.log("Hi from my script!");
```

```
> node myscript
```

Данный скрипт также выполнился. Обратите внимание, что добавлять расширение “.js” в данном случае необязательно.

Перед продолжением работы зальем наши изменения в репозиторий

2. Подключение и использование модулей

Как мы выяснили в предыдущем пункте, модуль – это отдельный js-файл. Модули могут импортировать друг друга, что позволяет разделять логику в архитектурно сложных проектах.

Изменим index.js следующим образом и посмотрим, как поменялся вывод

```
require("./myscript")
console.log("Hi! This is my first NodeJS app!");
```

Приведенные примеры больше являются отдельными скриптами, нежели полноценными модулями. Скрипт является интерпретируемым. Инструкции в скрипте выполняются сверху вниз. Для выполнения инструкции по нашему требованию она оборачиваются в функцию:

```
function sayHello(name) {
    console.log("Hi, " + name + "! This is my first NodeJS app!");
}
```

И вызывается: `sayHello("John")`
Изменим файл index.js и проверим вывод.

Все объявленные функции доступны только в приватном контексте модуля. Для экспорта функций (а также констант или переменных) из модуля используется дефолтный объект module.exports.

Этот объект уже существует при инициализации модуля, соответственно можно инициализировать его поля:

```
module.exports.myFunction = function() {
    console.log('Inside module.exports.myFunction')
}
```

или переопределять объект целиком:

```
module.exports = {
    sayHello: function(name) {
        console.log("Hi from my script, " + name + "!");
    }
}
```

или же вообще экспортировать только одну функцию, а не объект:

```
module.exports = function() {
    console.log('I am module.exports')
}
```

Иными словами, module.exports может являться чем угодно – как объектом со своими полями, так функцией или обычной константой.

При импорте модуля этот объект присваивается в ту переменную, которую мы задаем и все его поля становятся доступны через эту переменную:

```
var myscript = require("./myscript");

function sayHello(name) {
    console.log("Hi, " + name + "! This is my first NodeJS app!");
}

sayHello("John");
myscript.sayHello("John");
```

Изменив вышеуказанным образом наши файлы, проверим вывод index.js

3. Использование сторонних библиотек

moment.js (https://momentjs.com) – библиотека для работы с DateTime. Javascript (VanillaJS) предоставляет нативные возможности для работы со временем, а moment.js предоставляет для них удобную обертку.

Установим этот пакет с помощью команды

```
> npm install moment --save
```

Npm – это менеджер, хранящий базу публичных и приватных пакетов. Пакет, упрощенно, представляет собой копию git-репозитория, сохраненную в базе npm. Пакет можно установить либо по имени, либо по прямой ссылке на git-репозиторий или tarball.

Пакеты могут быть установлены глобально (с флагом –g) и локально в проекте. Глобальные пакеты будут доступны во всех других проектах, локальные – только в текущем

Список текущих зависимостей хранится в package.json. Когда другой разработчик скачает наш репозиторий, ему будет достаточно выполнить команду `npm install`, после чего npm самостоятельно просмотрит список зависимостей и скачает их

Чтобы поддерживать этот список актуальным, при установке необходимо использовать флаг `--save`

После установки подключим наш новый пакет:

```
var moment = require("moment");
```

Задание: создать отдельный модуль timeProvider. С помощью документации с официального сайта moment.js вывести текущее время в произвольном формате (в родительском модуле)

Прежде чем заливать изменения обратим внимание на папку node_modules. Она содержит библиотеки, установленные через npm – нет смысла заливать ее в наш репозиторий. Добавим ее в git ignore и закоммитим изменения

4. Веб-сервер

В стеке MEAN для реализации веб-сервера используется Express.js. Создадим файл server.js и добавим следующий код:

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
    res.send('Hello World!')
})

app.listen(server.listen(process.env.PORT || 3000), function () {
    console.log('Server started!')
})
```

Примечание: конструкция `process.env.PORT || 3000` *представляет собой логическую операцию, которая вернет* `process.env.PORT`*, если он будет содержать значение и* `3000` *в противном случае.* `process.env.PORT` *задается в качестве переменной окружения и автоматически определяется на Heroku, что понадобится нам при выполнении последнего раздела работы.*

Убедимся, что у нас установлены все зависимости и запустим файл. Для проверки работоспособности зайдем на 3000 порт в браузере. Как мы видим, веб-сервер работает и отдает ответ

Подобным образом можно было бы возвращать верстку страниц, однако наиболее грамотный способ – публикация статичных ресурсов.

Добавим простой HTML файл, содержащий текст “Hello world”.

Изменим код сервера:

```
var express = require('express');
var app = express();

app.use(express.static(__dirname + '/'));

app.listen(process.env.PORT || 3000, function () {
    console.log('Server started!')
})
```

Обратите внимание, что в качестве папки со статичными ресурсами выбрана корневая папка “/”. На следующих шагах она будет изменена.

Перезапустим сервер и укажем путь к нашему файлу. Проверим, все ли работает

5. Маршрутизация

В сложных приложениях у нас может быть множество нестатичных путей, которые будут обслуживаться разными способами. Для управлениям ими используется Router.

Примем, что API у нас будет доступно по адресу /api. По адресу /api/time будет доступно текущее время

Импортируем модуль `timeProvider` и пробросим функцию now(), объявленную в его module.exports, внутрь обработчика для роутера:

```
var router = express.Router();
router.get('/time', function(req, res) {
    res.json({
        time: timeProvider.now()
    });
});
app.use('/api', router);
```

Зайдем по указанному адресу и проверим, что наш обработчик корректно работает. Обратите внимание: для передачи данных между сервером и браузером по умолчанию используется формат JSON. Это стандарт де-факто для REST-сервисов. В SOAP-сервисах используется формат XML. Вы можете реализовать абсолютно любой формат данных, вплоть до передачи текста, склеенного через “;” или “|”, но наиболее адекватным решением будет использование дефолтного JSON-формата.

Перед переходом к следующему пункту убедимся, что закоммитили последние изменения.

6. Структура проекта

Упорядочим структуру проекта. Вынесем index.html в папку “frontend” (не забудем изменить адрес для статичных ресурсов в конфигурации сервера), создадим папку “api”, куда поместим наш timeProvider.js

Данная операция может быть выполнена вручную или с помощью инструмента рефакторинга в WebStorm, что позволит автоматически найти все ссылки на перемещаемые файлы и обновить их

7. AJAX

Asynchronous Javascript And Xml – технология, позволяющая обратиться к серверу без перезагрузки страницы. При реализации классических приложений мы сталкиваемся с проблемой полной перезагрузки страницы при любом действии. AJAX позволяет в фоновом режиме запросить и обработать любой ресурс, расположенный на сервере.

В качестве ресурса может выступать REST-эндпоинт, поставляющий данные, либо CSS-файл, который мы будем динамически подцеплять на странице, либо скрипт, который нужно загрузить по требованию

Для работы с AJAX разумнее всего использовать какой-либо фреймворк. В качестве демонстрации подключим в HTML файл jQuery через CDN

В папке “frontend” создадим и подключим скрипт frontend.js. Изменим содержимое тега body нашей страницы:

```
<body>
    <h1>Hello from my HTML</h1>
    <button id="getTimeButton">Get Time</button>
    <div id="timeResult"></div>
</body>
```

Задание: по нажатию на кнопку “Get Time” отправлять AJAX-запрос на сервер, получать текущее время и отображать его внутри блока “timeResult”

Убедимся, что все работает и зальем изменения

8. Heroku

Heroku – облачный PaaS-провайдер для развертывания приложений. В рамках работы он позволяет развернуть наше приложение в Интернете, а не просто локально на ноутбуке.

Heroku предоставляет контейнер, в котором может быть запущено приложение, аддоны (от баз данных до вспомогательных сервисов) и бесплатный домен третьего уровня

Для работы с heroku необходима учетная запись, которую можно создать на официальном сайте. Для развертывания проектов необходима утилита heroku cli, которую можно скачать и установить с официального сайта. При дефолтных параметрах установки утилита попадет в PATH. Корректную работу можно проверить с помощью команды

```
> heroku -v
```

Взаимодействие построено на технологии Continuous Integration («непрерывная интеграция», которая подразумевает авторазвертывание и автотестирование проекта при любом изменении

Инициализация проекта происходит следующим образом:

```
> heroku create YOUR_PROJECT_NAME
```

После инициализации на heroku создается удаленный репозиторий, ссылка на который добавляется в наш локальный проект. Теперь мы можем выливать изменения в 2 удаленных репозитория – в свой изначальный и в heroku

После git push в удаленный репозиторий на heroku произойдет автоматический деплой вашего приложения по адресу https:// YOUR_PROJECT_NAME.herokuapp.com

В этом можно убедиться с помощью команды

```
> heroku open
```

9. Что должно быть в итоге:

1. Сервер, публикующий как статичные файлы, так и функционал отдельных модулей
2. Модуль, экспортирующий функцию
3. Веб-страница с подключенным скриптом, в котором обрабатываются DOM-события и выполняется AJAX-запрос
4. Приложение, развернутое на Heroku

Задание для выполнения

Описанным образом реализовать веб-приложение, которое выводит время с помощью momentjs в любом переданном с фронтенда формате

либо

Любое другое приложение, удовлетворяющее требованиям пункта 9

Вопросы для защиты:

1. Что такое ES? V8?
2. Как производится инициализация пустого репозитория? Как добавить существующий репозиторий с github? В чем специфика локальных и удаленных репозиториев?
3. Какой версией NodeJS Вы пользовались при выполнении работы?
4. Что такое npm? Как установить пакет с помощью npm:
   4.1. Локально в проекте?
   4.2. Глобально в системе?
5. Как обеспечить наличие установленных пакетов у других разработчиков, когда они будут смотреть наш проект?
6. Что такое module.exports? Какие значения он может принимать? Описать минимум 3 способа, как можно экспортировать функцию из метода
7. Как подключить модуль? Как обратиться к функции внутри подключенного модуля?
8. Что такое CDN? Какова их функция и преимущество? Когда нельзя использовать CDN?

9. Что такое AJAX? Привести пример. В чем преимущество? Как передать параметр в AJAX-запрос?
10. Что такое DOM? Как работать с DOM с помощью jQuery?