Университет ИТМО

Факультет программной инженерии и компьютерной техники

кафедра информатики и прикладной математики

направление подготовки 09.03.04 "Программная инженерия"

ОТЧЁТ

о учебной практике

|  |  |  |
| --- | --- | --- |
| Тема задания | | Разработка Web приложения для сети ресторанов |
| Студент | Плюхин Дмитрий Алексеевич | |

Руководитель практики: Ф.И.О. полностью, место работы и должность.

Оценка, рекомендованная руководителем: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| Практика пройдена с оценкой \_\_\_\_\_\_\_\_\_\_\_  **Подписи членов комиссии**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / к.т.н., доц. Лаздин А.В.**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Логинов И.П.**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Исаев И.В.**  **Дата: \_\_\_\_\_\_\_\_\_\_** |

Санкт-Петербург

2017

Оглавление

[Введение 2](#_Toc483760273)

[Цели 2](#_Toc483760274)

[Технологическая платформа 2](#_Toc483760275)

[Теоретические сведения 2](#_Toc483760276)

[Node.js 2](#_Toc483760277)

[MySQL 4](#_Toc483760278)

[Реализация 4](#_Toc483760279)

[Структура приложения 4](#_Toc483760280)

[Использование HTML/CSS/JavaScript 5](#_Toc483760281)

[Использование Node.js 5](#_Toc483760282)

[Использование MySQL 7](#_Toc483760283)

[Результат 8](#_Toc483760284)

[Выводы 10](#_Toc483760285)

# Введение

## Цели

Разработать Web приложение для сети ресторанов, которое:

1. Имеет UI управления поварами: создание/редактирование/удаление повара. Атрибуты повара: ФИО, аттестации на отделы, и предпочтения-ограничения.

2. Умеет автоматически составлять расписание смен на 1 месяц для каждого ресторана, так чтобы в каждый момент времени все три отдела кухни были бы обеспечены поварами. Т.е. должно быть 3 повара: один с квалификацией Русская кухня, другой – Италия и третий – Япония.

3. Имеет UI отображения составленного расписания.

## Технологическая платформа

Были предложены следующие варианты технологий для реализации компонентов приложения:

• Frontend: HTML/CSS/JavaScript

• Backend (одно из): ASP.NET, Node.js, PHP

• Database (одно из): SQL Server, MySQL, PostgreSQL

Для реализации Backend была выбрана технология Node js в силу большого количества документации и простоты изучения, для базы данных были выбраны средства MySQL как наиболее распространенные.

# Теоретические сведения

## Node.js

Node.js - это серверная JavaScript-nлaтфopмa, предназначенная для создания масштабируемых распределенных сетевых приложений, использующая событийно-ориентированную архитектуру и неблокирующее асинхронное взаимодействие. Она основана на JаvаSсriрt движке V8 и использует этот же JavaScript для создания приложений.

Например, запрос к базе данных в случае синхронно выполняемого кода выглядит следующим образом:

result = query('SELECT \* FROM posts WHERE id = 1');

do\_something\_with(result);

То есть, поток окажется заблокирован в ожидании ответа от базы данных.

Node js предлагает несколько иной способ организации обработки данных, приходящих от сервера:

query\_finished = function(result) {

do\_something\_with(result);

}

query('SELECT \* FROM posts WHERE id = 1', query\_finished);

То есть, при получении ответа от базы данных будет вызываться определенная функция, которая и осуществит обработку результата– в этом случае устраняется необходимость блокирования потока исполнения.

Другая важная концепция JavaScript вообще и Node js в частности – это использование замыканий (closures), которое делает возможным чтение и изменение переменных из внешнего блока внутри какой-либо функции (которая и называется замыканием), определенной внутри этого блока, например:

var clickCount = 0;

document.getElementById('myButton').onclick = function() {

clickCount += 1;

alert("clicked " + clickCount + " times.");

};

Такие действия становятся возможными благодаря тому, что при запуске функция создает объект LexicalEnvironment, записывает туда аргументы, функции и переменные. Процесс инициализации выполняется в том же порядке, что и для глобального объекта, который, вообще говоря, является частным случаем лексического окружения. Объект LexicalEnvironment является внутренним и скрыт от прямого доступа.

Интерпретатор, при доступе к переменной, сначала пытается найти переменную в текущем LexicalEnvironment, а затем, если её нет – ищет во внешнем объекте переменных.

Подобная концепция доступна и в некоторых других языках, в частности, в Java:

public Function<Integer, Integer> make\_fun() {

int n = 0;

return arg -> {

System.out.print(n + " " + arg + ": ");

arg += 1;

// n += arg; // Produces error message

return n + arg;

};

}

Помимо эффектной асинхронной модели работы, неблокирующих процессов, высокой производительности, Node.js делает то, что считалось принципиально невыполнимым, - дает возможность разработчику создавать как server-side/backend-, так и frоntеnd-приложения, пользуясь единой технологией! Теперь на JavaScript можно написать как обработчик httр-запросов, так и настоящий, полнофункциональный веб-сервер. Можно работать с SQL- (и NoSQL-) базами данных, сетью, файловой системой.

Node.js доказала свою состоятельность, и сейчас её «боевое» использование - не экзотика, а нормальная практика, особенно в пресловутых высоконагруженных проектах. Node.js сейчас тем или иным образом используют такие известные участники IТ-рынка, как Groupon, SAP, Linkedln, Microsoft, Yahoo!, Walmart, PayPal.

Платформа Node.js была создана в 2009 rоду Райном Далом (Ryan Dahl) в ходе исследований по созданию событийно-ориентированных серверных систем. Асинхронная модель была по причине низких накладных расходов (по сравнению с мноrопоточной моделью) и высокого быстродействия. Node была (и остается) построена на основе JavaScript-движкa V8 с открытым исходным кодом, разработанного компанией Google в процессе работы над своим браузером Google Chrome. Это была не первая реализация V8 на стороне сервера, но технология оказалась так удачно спроектирована, что сразу же обрела большое число сторонников и энтузиастов и, как следствие, множество модулей, реализующих самый разнообразный функционал. В настоящее время разработка Node.js спонсируется основанной Райном компанией Jоуеnt.

## MySQL

MySQL— свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

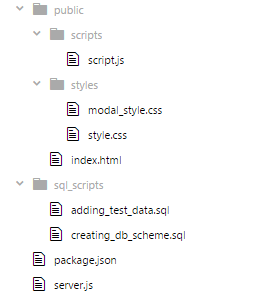
MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

# Реализация

## Структура приложения

Проект имеет стандартную для веб-приложения конфигурацию файлов:

script.js содержит сценарий, выполняющий проверку данных, введенных пользователем, отправку их на сервер, получение данных с сервера и управление их отображением.

modal\_style.css содержит стили модального окна, предназначенного для редактирования характеристик повара.

style.css включает стили для html элементов

index.html содержит разметку страницы приложения

adding\_test\_data.sql содержит набор sql запросов, выполняющих заполнение базы данных тестовыми данными

creating\_db\_scheme.sql содержит sql запрос для создания схемы базы данных, используемой приложением

package.json – файл конфигурации Node.js, в котором объявлены дополнительные модули, используемые приложением

server.js – серверный сценарий Node.js

## Использование HTML/CSS/JavaScript

Что касается frontend, ключевое значение имеют методы, извлекающие необходимые данные из html – формы и отправляющие их на сервер при помощи ajax. Действия, осуществляемые при добавлении нового повара, являются наглядной иллюстрацией применения на практике принципа асинхронного программирования – в качестве одного из свойств создаваемого объекта используется функция, которая будет вызвана в случае успешного выполнения запроса:

$.ajax({

type : "post",

url : "/addnew\_handler",

data : JSON.stringify(getRequestObject("addnew",form)),

dataTpe : "json",

contentType : "application/json",

success : function(data){

loadCooksInfo();

showActionResult(form.elements, data, "Accepted", "New cook added", "New cook wasn't added");

},

});

## Использование Node.js

Для упрощения разработки были использованы дополнительные модули Node js:

* Express для быстрой настройки и запуска сервера
* Body-parser для обмена данными между клиентской и серверной частью
* Mysql для организации работы с базой данных

В свою очередь, сервер Node js ответственен за прием данных, запись (или извлечение) их в базу данных, обработку и предоставление результата. Именно сервер занимается анализом зарегистрированных поваров и составлением корректного графика работы. Ключевую роль здесь играет следующий цикл, предназначенный для расчета всех возможных вариантов и выбора оптимального:

for (var day = 0; day < num\_of\_days; day++){//for each day in month

preferring\_constraint = 1;//firstly try to create schedule, which will be preferred for all

unbusyAll(cooks\_attributes);

clearTimeRestaurants(time\_restaurants);

while(!isAllBusy(time\_restaurants)){// while restaurants is not enough full of cooks

if (isAllCooksBusy(cooks\_attributes)) {

return time\_restaurants;//no way because can't find unbusy cook

}

for (var g = 0; g < num\_of\_kitchens; g++){//for each kitchen

for (var h = 0; h < num\_of\_restaurants; h++){// for each restaurant

if (time\_restaurants[g][h] == 24) continue;// we don't need to check handled restaurants again

broke = false;

for (var i = 0; i < cooks.length; i++){// for each cook

if(cooks\_attributes[i].busy == 1){

if (i == (cooks.length - 1)){

if (preferring\_constraint == 1){

preferring\_constraint = 0;

continue;

}

return time\_restaurants;//no way because can't find required cook

}

continue;// if cook already busy (or get rest today), skip him

}

if (handleWorkCounters(cooks\_attributes[i], 1, 5)) continue;//must we give to this cook a day rest?

if (handleWorkCounters(cooks\_attributes[i], 0, 2)) continue;//must we give to this cook a day rest?

for (var l = 0; l < cooks\_attributes[i].days.length; l++){//for each configure of the day

if ((cooks\_attributes[i].days[l].begin\_hour == time\_restaurants[g][h]) &&// if it is required time at the moment

(cooks\_attributes[i].days[l].preferred >= preferring\_constraint) &&//if it is enough preferred

(((24-cooks\_attributes[i].days[l].end\_hour) >= 4)||

(cooks\_attributes[i].days[l].end\_hour==24))){//if it is correct working day duration at the situation

if ((g==0) && (cooks[i].russian == 1)){

broke = writeCookEntry(day, h, l, cooks[i], cooks\_attributes[i], "russian", time\_restaurants[g], schedule[day][h][g]);

break;

}

if ((g==1) && (cooks[i].italian == 1)){

broke = writeCookEntry(day, h, l, cooks[i], cooks\_attributes[i], "italian", time\_restaurants[g], schedule[day][h][g]);

break;

}

if ((g==2) && (cooks[i].japanese == 1)){

broke = writeCookEntry(day, h, l, cooks[i], cooks\_attributes[i], "japanese", time\_restaurants[g], schedule[day][h][g]);

break;

}

}//large if

}//for each configure of the day

if (broke){

break;

} else if (i == (cooks.length - 1)){

if (preferring\_constraint == 1){

preferring\_constraint = 0;//try to find less preferred shifts

continue;

}

return time\_restaurants;

}

}// for each cook

}// for each restaurant

}// for each kitchen

}// while restaurants isn't filled

}// for each day in month

## Использование MySQL

Схема базы данных формируется запросом

create table cooks(

cookid int not null auto\_increment primary key,

name nvarchar(50) not null,

surname nvarchar(50) not null,

patronymic nvarchar(60) null,

russian boolean not null,

italian boolean not null,

japanese boolean not null,

morningshifts boolean not null,

eveningshifts boolean not null,

necessityshiftstime boolean not null,

dayduration int(2) not null,

necessitydayduration boolean not null,

workingmode\_5\_2 boolean not null,

workingmode\_2\_2 boolean not null

);

Запросы к базе данных формируются непосредственно на сервере при обращении клиента. Например, следующим образом выглядит добавление нового повара в базу данных:

connection.query("INSERT INTO cooks (name, surname, patronymic, russian, italian, japanese, morningshifts, eveningshifts, "+

"necessityshiftstime, dayduration, necessitydayduration, workingmode\_5\_2, workingmode\_2\_2) "+

"VALUES ('"+request.body.name+"', '"+request.body.surname+"', '"+request.body.patronymic+"', "+request.body.russian+", "+request.body.italian+", "+

request.body.japanese+", "+request.body.morningshifts+", "+request.body.eveningshifts+", "+request.body.necessityshiftstime+", "+request.body.dayduration+

", "+request.body.necessitydayduration+", "+request.body.workingmode\_5\_2+", "+request.body.workingmode\_2\_2+");",function(error,result,fields){

if(error) return response.json(error);

});

# Результат

Разработанное web-приложение, принимающее от пользователя данные о поварах – сотрудниках ресторана и генерирующее корректное расписание их работы.

|  |
| --- |
|  |
| Рисунок 1. Главное окно приложения |
| |  | | --- | |  | | Рисунок 2. Вывод приложения при невозможности генерирования расписания | |
|  |
| Рисунок 3. Модальное окно для редактирования данных повара |

|  |
| --- |
|  |
| Рисунок 4. Фрагмент сгенерированного расписания |

# Выводы

* Было разработано веб-приложение, состоящее из трех компонентов – клиентской составляющей (HTML, CSS, JavaScript), серверной составляющей (Node.js) и базы данных (MySQL).
* Был разработан алгоритм для составления расписания выполнения работ с учетом набора ограничений