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

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-платформа, предназначенная для создания масштабируемых распределенных сетевых приложений, использующая событийно-ориентированную архитектуру и неблокирующее асинхронное взаимодействие. Она основана на JavaScript 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 frontend-приложения, пользуясь единой технологией! Теперь на JavaScript можно написать как обработчик http-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 IT-рынка, как Groupon, SAP, Linkedln, Microsoft, Yahoo!, Walmart, PayPal.

Платформа Node.js была создана в 2009 году Райном Далом (Ryan Dahl) в ходе исследований по созданию событийно-ориентированных серверных систем. Асинхронная модель была по причине низких накладных расходов (по сравнению с многопоточной моделью) и высокого быстродействия. Node была (и остается) построена на основе JavaScript-движка 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 Joyent.

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.

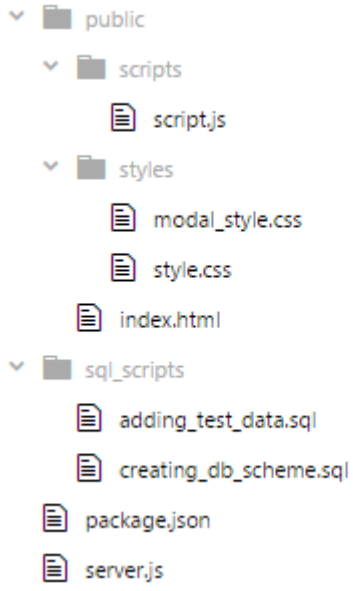
MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека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

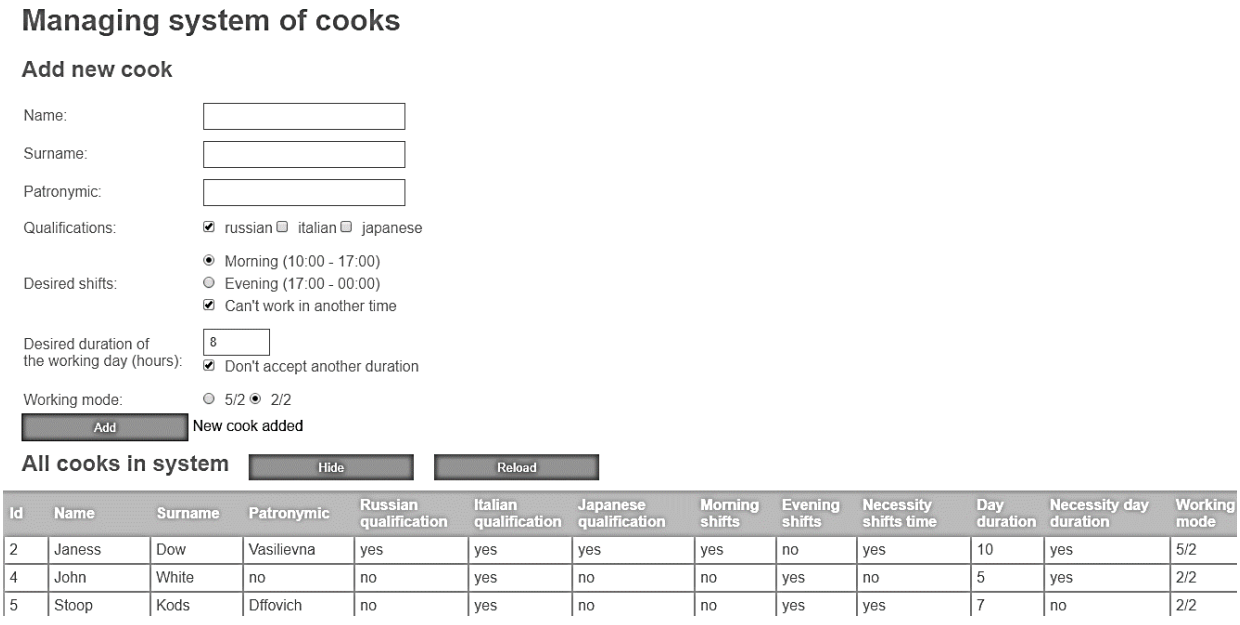
```
      workingmode_2_2 boolean not null
);
```

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.



Id	Name	Surname	Patronymic	Russian qualification	Italian qualification	Japanese qualification	Morning shifts	Evening shifts	Necessity shifts time	Day duration	Necessity day duration	Working mode
2	Janess	Dow	Vasilievna	yes	yes	yes	yes	no	yes	10	yes	5/2
4	John	White	no	no	yes	no	no	yes	no	5	yes	2/2
5	Stoop	Kods	Dffovich	no	yes	no	no	yes	yes	7	no	2/2

Рисунок 1. Главное окно приложения

Рисунок 2. Вывод приложения при невозможности генерирования расписания

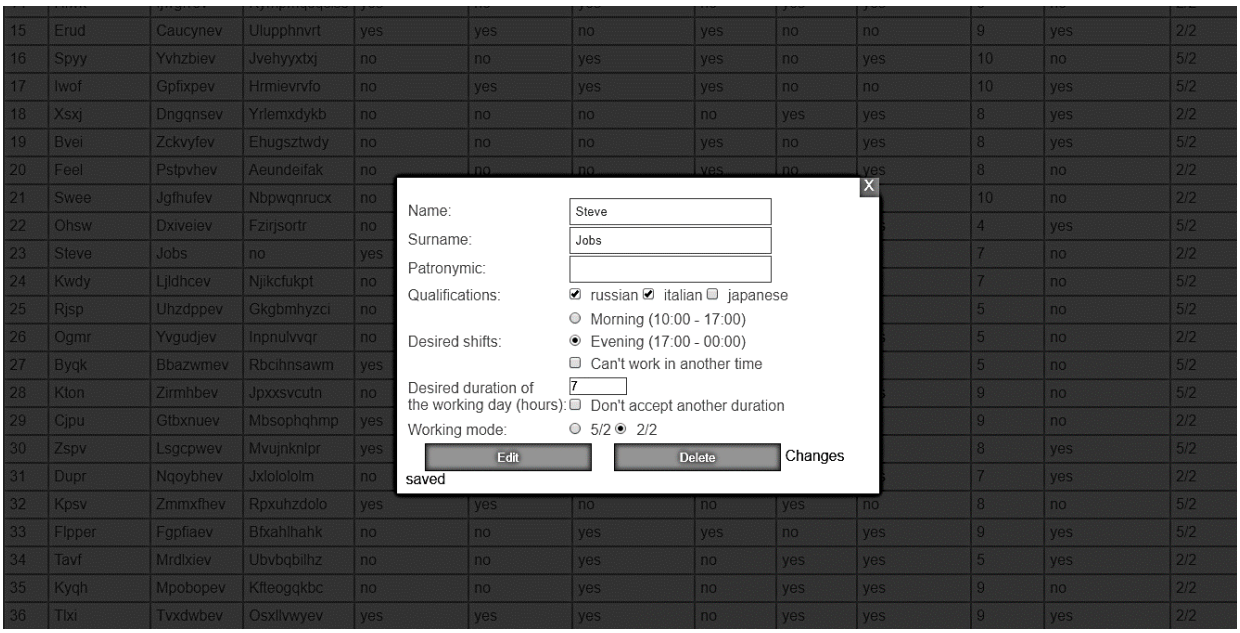


Рисунок 3. Модальное окно для редактирования данных повара

- 28 restaurant :
  - russian kitchen :
    - Qoxvltev Orso Lwcvyfdaso(id = 308) : 10:00 - 20:00
    - Tkqesgev Ilhd Zhjxobpwmu(id = 984) : 20:00 - 24:00
  - italian kitchen :
    - Sxlndeev Bgbo Ltnfnwxbhk(id = 379) : 10:00 - 18:00
    - Ufqqhnev Uqro Rwllyljkxm(id = 613) : 18:00 - 24:00
  - japanese kitchen :
    - Vbqdrbev Bvdb Fukoqoulre(id = 450) : 10:00 - 20:00
    - Cdifedev Wnak Fqqezlhycl(id = 590) : 20:00 - 24:00
- 29 restaurant :
  - russian kitchen :
    - Wxxwofev Dknm Jgeapxvjip(id = 310) : 10:00 - 14:00
    - Hbkvctev Rxql Reqveifalf(id = 598) : 14:00 - 24:00
  - italian kitchen :
    - Utluumev Eolk Bttniyufog(id = 380) : 10:00 - 18:00
    - Znohssev Ccem Memxezmncu(id = 622) : 18:00 - 24:00
  - japanese kitchen :
    - Zexzgeev Rrip Catslxjyvf(id = 458) : 10:00 - 18:00
    - Osuwakev Tylk Avdfndymnd(id = 718) : 18:00 - 24:00
- 30 restaurant :
  - russian kitchen :
    - Pgjdmbev Bbzx Snlraacgdu(id = 312) : 10:00 - 14:00
    - Kvyjwhev Djmj Udgwrttlyo(id = 420) : 14:00 - 24:00
  - italian kitchen :
    - Gvndylev Xkft Jlekooecyo(id = 383) : 10:00 - 14:00
    - Neeidsev Uyke Gybjpagebt(id = 627) : 14:00 - 24:00
  - japanese kitchen :
    - Uynpljev Bosa Mknlqvhvjg(id = 459) : 10:00 - 19:00
    - Lkurciev Zhmp Hoxvofjeum(id = 723) : 19:00 - 24:00
- 31 restaurant :
  - russian kitchen :
    - Mlsgcrev Ophr Eryubuwzho(id = 318) : 10:00 - 14:00
    - Nuncuuev Gqnp Qssmeknmtk(id = 455) : 14:00 - 24:00

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