4. алгоритм DFS;
5. алгоритм BFS;
6. алгоритм Дейкстры;
7. определение степени вершин графа;
8. файловый вывод результатов методов обхода и сохранение изображения графа.

При кодировании граф был представлен в виде списка (List) экземпляров класса вершин и списка экземпляров класса ребер. Когда происходит выбор вершины мышью, поиск нужной вершины из списка осуществляется посредством перебора всех вершин и проверки условия принадлежности точки, в месте щелчка мыши, окружности вершины с помощью уравнения окружности. На основании этих методов и структур, путем перебора списков вершин и ребер заполнялись матрицы смежности и инцидентности, а также списки смежности. Далее, исходя из этих представлений, на графе реализовывались алгоритмы обхода.

Ознакомиться с исходным кодом проекта можно на моем github https://github.com/Vlad-Senyuk.

ЛИТЕРАТУРА

1. Домнин, Л.Н. Элементы теории графов. Пенза: Изд-во Пенз. гос. ун-та, 2007. 147 с.
2. Кристофидес, Н. Теория графов. Алгоритмический подход. Москва: Мир, 1978. 432 с.
3. Nathan, A. WPF 4. Unleashed. Indianapolis. Indiana 46240 US, 2010. 844 p.

УДК 004.77

Студ. Р.И. Белькевич
Науч. рук. ст. преподаватель Е.А. Блинова
(кафедра информационных систем и технологий, БГТУ)

ОБЗОР ГОТОВЫХ РЕШЕНИЙ ДЛЯ ЭМУЛЯЦИИ И ЗАМЕЩЕНИЯ БЭКЕНДА ВЕБ-ПРИЛОЖЕНИЙ

Front-end и back-end — термины в программной инженерии, которые различают согласно принципу разделения ответственности между представительским уровнем и уровнем доступа к данным соответственно. Front-end — интерфейс взаимодействия между пользователем и основной программно-аппаратной частью (back-end). Front-end и back-end могут быть распределены между одной или

несколькими системами. В работе представлены три актуальных отобранных готовых решения (Deployd, Feathers js, LoopBack от StrongLoop) для замещения бэкенда, предназ-наченные для тестирования взаимодействия веб-интерфейса и функционала сервера, сокращения затрат на написание бэкенда, сокращения времени разработки и упрощения жизни веб-разработчикам как в сфере front-end, так и в сфере back-end разработки.

Deployd — самый простой способ создать свой API. Deployd подразумевает использование готовой коллекции как ресурса, ресурсы добавляются через веб интерфейс панели управления. Можно развернуть приложение самостоятельно на любой сервис поддерживающий Node.js и MongoDB. С другой стороны, Deployd непригоден для публикации приложений.

Для создания приложения необходимо набрать следующее:

```
dpd create hello-world
cd hello-world
dpd -d
```

После этого в браузере откроется вкладка с панелью управления. Все доступные методы изображены на рисунке 1.

Access the `/test-stuff` collection over http.

Task	Method	Route	Accepts	Returns
Listing data	GET	/test-stuff	*Nothing*	An array of objects
Creating an object	POST	/test-stuff	A single object	The saved object (or errors)
Getting an object	GET	/test-stuff/<id>	*Nothing*	A single object
Updating an object	PUT	/test-stuff/<id>	A single object	The saved object (or errors)
Deleting an object	DELETE	/test-stuff/<id>	A single object	*Nothing*

Рисунок 1 – доступные методы созданного API.

Пример использования:

```
function getAll (){
let xmlhttp=new XMLHttpRequest();
xmlhttp.open('GET','http://localhost:2403//test-stuff',false);
xmlhttp.send(null);
if(xmlhttp.status==200){
let result=JSON.parse(xmlhttp.responseText)
  //ToDo  }
}
```

Feathers.js —очень мощный и удобный фреймворк для создания серверных приложений на Node.js. Feathers.js предназначен для создания сервисов (REST, Socket.io и Primus), требует минимум усилий и доработки кода. Минусом является высокий порог вхождения.

Для создания приложения необходимо набрать следующее:

```
mkdir feathers-app
cd feathers-app/
feathers generate
```

После этого запустится генератор, с помощью которого будут установлены все необходимые настройки. Для использования необходимо сделать следующее:

Сервер:

```
const app = feathers();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.configure(rest());
app.configure(socketio());
app.listen(3030);
```

Клиент (браузер):

```
var socket = io('http://localhost:3030');
var app = feathers();
app.configure(feathers.hooks());
app.configure(feathers.socketio(socket));
app.service('users').find({}).then(function(users){
        console.log('Users!', users);});
        })   });
```

LoopBack от StrongLoop — это полноценный Node.js-фреймворк для соединения приложений с данными через REST API. Позволяет разработчикам сосредоточиться на бизнес-логике, т.к. код для API генерируется автоматически. LoopBack последует идеологии open source и обладает высокой степенью расширяемости.

Для создания приложения необходимо набрать следующее:

```
npm install -g apiconnect
apic loopback
```

После завершения работы генератора по адресу localhost:8080/explorer получаем готовую документацию (рисунок 2):

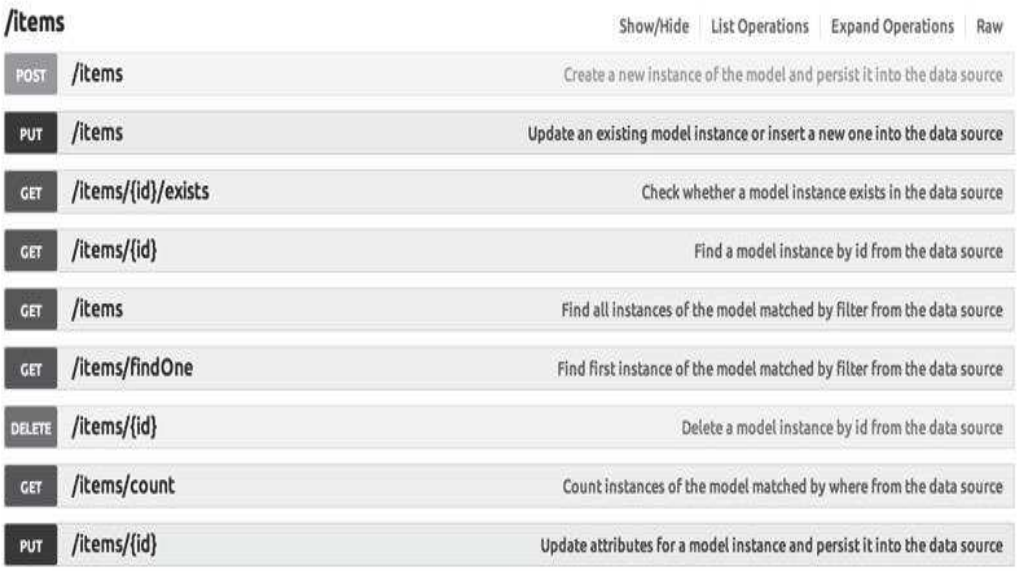

Рисунок 2 – доступные методы.

Пример использования:

```
function getAll (){
let xmlhttp=new XMLHttpRequest();
xmlhttp.open('GET','http://localhost:8080/items',false);
xmlhttp.send(null);
if(xmlhttp.status==200){
let result=JSON.parse(xmlhttp.responseText)
   //ToDo    }
}
```

Если надо проверить, как будут выгружаться из хранилища данные и каким образом это будет отображаться на странице, достаточно использовать Deployd. Если планируется расширение проекта и требуется высокая надежность, лучше выбрать LoopBack. Если создаётся проект, не требующий серьёзной разработки, но все таки возможна какая-то кастомизация, то лучшим выбором является Feathers.js.

ЛИТЕРАТУРА

1. feathersjs.com [Электронный ресурс] Режим доступа: https://docs.feathersjs.com/ — Дата доступа: 18.03.2017.

2. deployd.com [Электронный ресурс] Режим доступа: http://docs.deployd.com/— Дата доступа: 18.03.2017.
3. loopback.io [Электронный ресурс] Режим доступа: http://loopback.io/doc/— Дата доступа: 18.03.2017.

УДК 004.9

Студ. А.Н. Зайцев
Науч. рук.: ст. преп. Е.А. Блинова
(кафедра информационных систем и технологий, БГТУ)

ВЫБОР ОПТИМАЛЬНОГО АЛГОРИТМА ПОИСКА ПОДСТРОКИ В СТРОКЕ ДЛЯ РАЗЛИЧНЫХ ВИДОВ ДАННЫХ

В настоящее время довольно большой пласт информации представлен в текстовом виде: огромные библиотеки книг, архивы данных научных исследований. Одной из важнейших задач является поиск нужной информации за приемлемое время в этих огромных массивах данных. Целью данной научной работы является подбор оптимальных алгоритмов поиска текстовой информации в больших массивах данных для различных типов входных данных.

В ходе научной работы будут использоваться следующие обозначения: S — искомый текст (паттерн), T — текст, в котором ведётся поиск, N — длина S, M — длина T.

Самый простой существующий алгоритм работает по следующему принципу: идём слева направо по T и пытаемся найти такую позицию pos, где T[pos… pos + N] = S. Сравнение производится в цикле посимвольно. Асимптотика данного метода в худшем случае составляет O(N * M). Надо отметить, что этот худший случай на реальных данных очень и очень редкий, так как для его выполнения требуется выполнение следующего условия: На каждом шаге алгоритма S должна полностью совпадать с подстрокой T, за исключением последнего символа. На самом деле алгоритм работает с гораздо лучшим ожиданием.

Вполне очевидным кажется развитие метода — сравнение строки не посимвольно, а с помощью хеш-функций, которые позволяют за O(1) (в данной работе принимается, что асимптотика сравнения чисел процессором составляет O(1)) проверить на идентичность два числа. Данный алгоритм называется методом Рабина-Карпа [3]. Недостатком данного алгоритма является то, что подсчёт хеш-функции является довольно медленной операцией.

Следующий алгоритм поиска называется алгоритм Кнута-Морриса-Пратта (КМП) [2]. Данный алгоритм имеет асимптотику O(N