



Faculty of Computer Science

Prof. Alexander Adam, Daniel Richter

Datenbanken und Web Techniken
Project Report (SS 2018)

DownTown Diner
Restaurant Hunter

Chemnitz, 1. July 2018

Team members

Shyam Ashish Agrawal (Master in Web Engineering)
Pallavi Singh (Master in Web Engineering)

Matriculation no.

479026
485344

Table of content

1. Introduction.....	4
2. Resources.....	5
2.1. Technologies.....	5
2.2. Database Selection.....	7
3. Project Flow.....	8
3.1. Crawling and Database.....	8
3.2. REST API.....	9
3.3. Frontend.....	11
4. Practical Demonstration.....	12
5. Conclusion.....	16
6. Appendix.....	17

Work Split Details:

Team Member	Work Details for the Project
Shyam Ashish Agrawal	<ol style="list-style-type: none">1) Project Structure and Overview (Backend)2) Web Crawling3) Database4) REST API
Pallavi Singh	<ol style="list-style-type: none">1) Website Design of the Project (front-end)2) Handling Data From API

1. Introduction

Manually finding the Restaurant Dishes and its Price is time consuming and difficult nowadays. The aim of this project is to provide a system which finds Dishes from different Restaurants and provides necessary information about name of the Restaurant, address and prices of each dish available.

The goal of this task is to create a website that is able to fetch, store and visualization data of Downtown Dinner. The benefits of this service are able to fetch the real time data from the three different restaurants.

Customers can search the dishes based on keywords and will get the information about the Ingredients, Price and Description. Customers also get the details of all the available dishes and opening hours of restaurants by clicking any of the dishes.

2. Resources

2.1 Technologies

The System is Developed Using Node.js Technology. Node.js is an open source, cross-platform runtime environment for developing server-side and networking applications. Node.js applications are written in JavaScript and can be run within the Node.js runtime on OS X, Microsoft Windows, and Linux.

Following are some of the important features that make Node.js the first choice of web development:

- Asynchronous and Event Driven
- Very Fast
- Single Threaded but Highly Scalable
- No Buffering

Why We have selected Node.js

Firstly, using Node.js as our server technology gives our team a great boost that comes from using the same language on both the front end and the back end. This means that our team is more efficient and cross-functional, which, in turn, leads to lower development costs.

We have also used different Node.js Frameworks. Below are the details:

1. Request - Package for HTTP

Request provides an HTTP request API on the client and server. To use these functions, add the Request package to your project by using your terminal.

2. Cheerio - Package for jQuery-Traversing

jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

3. Mongoose - Package for Mongo DB

Mongoose is an Object Document Mapper (ODM). This means that Mongoose allows you to define objects with a strongly-typed schema that is mapped to a MongoDB document.

4. Express - Package for Web Framework, Rest API

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

5. body-parser – Package for Middleware

In order to read HTTP POST data, we have to use "body-parser" node module. Body-parser is a piece of express middleware that reads a form's input and stores it as a JavaScript object accessible through "req.body".

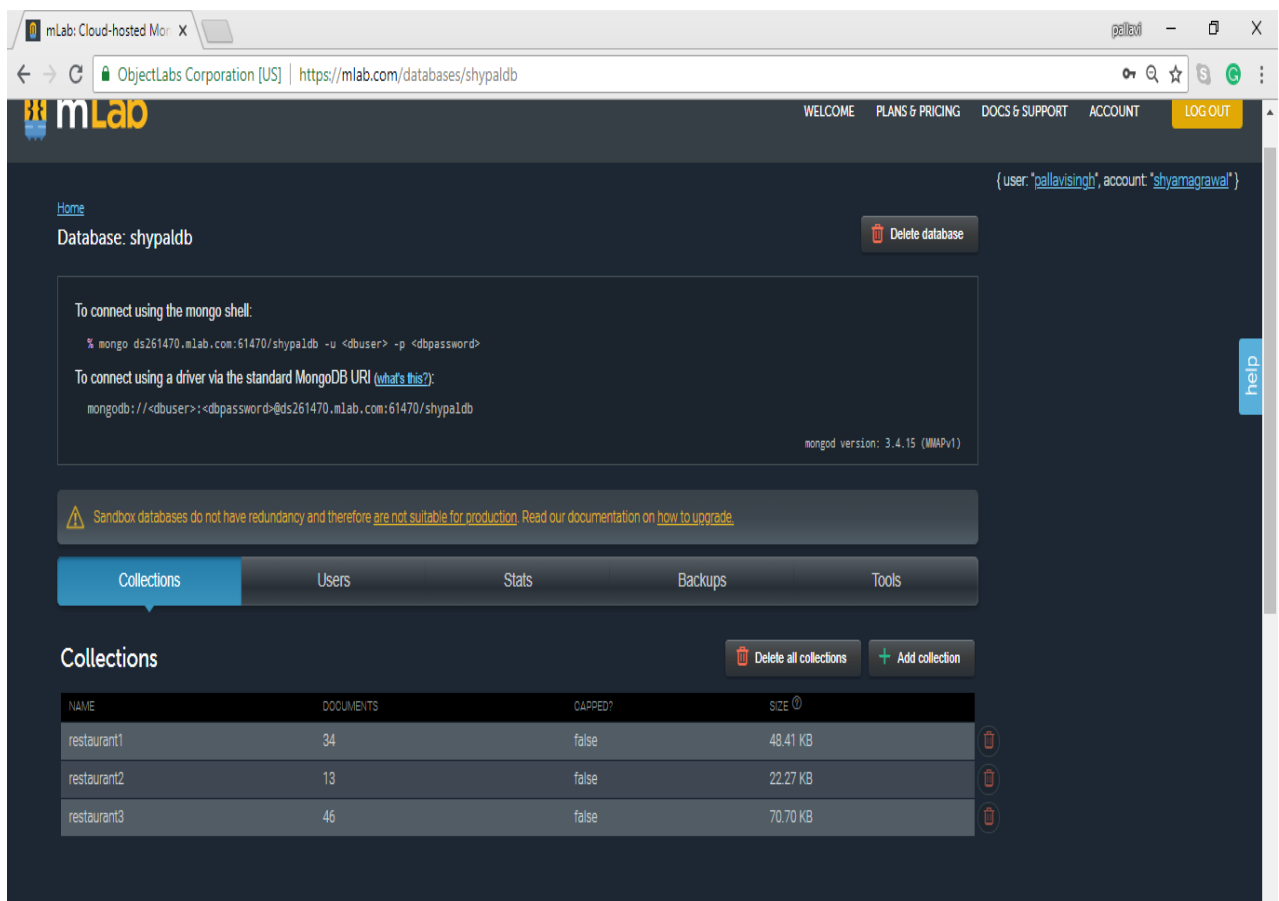
We have used scripting language like JavaScript, AJAX. For styling, the web pages we have used Bootstrap and CSS.

For Integrated Development Environment, we have used Microsoft Visual Studio Code 2018.

2.2 Database Selection

The System is using MongoDB as the database. MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas.

In the project, System is using mLab: Database-as-a-Service for MongoDB. mLab is a fully managed cloud database service featuring automated provisioning and scaling of MongoDB databases, backup and recovery, 24/7 monitoring and alerting, web-based management tools, and expert support. mLab's Database-as-a-Service platform powers hundreds of thousands of databases across AWS, Azure, and Google and allows developers to focus their attention on product development instead of operations.



The screenshot displays the mLab web interface for a MongoDB database named 'shypaldb'. The interface includes a navigation bar with links for 'WELCOME', 'PLANS & PRICING', 'DOCS & SUPPORT', 'ACCOUNT', and 'LOG OUT'. The user is logged in as 'pallavisingh' on the 'shyamagrawal' account. The main content area shows the database name and a 'Delete database' button. Below this, there are instructions for connecting to the database using the mongo shell and a standard MongoDB URI. A warning message states: 'Sandbox databases do not have redundancy and therefore are not suitable for production. Read our documentation on how to upgrade.' The 'Collections' tab is selected, showing a table with three collections: 'restaurant1' (34 documents, 48.41 KB), 'restaurant2' (13 documents, 22.27 KB), and 'restaurant3' (46 documents, 70.70 KB). There are buttons for 'Delete all collections' and 'Add collection'.

NAME	DOCUMENTS	CLAPPED?	SIZE
restaurant1	34	false	48.41 KB
restaurant2	13	false	22.27 KB
restaurant3	46	false	70.70 KB

3. Project Flow

3.1 Crawling & Database

A crawler is a program that visits Web sites and reads their pages and other information in order to create entries for a search engine index. Crawlers apparently gained the name because they crawl through a site a page at a time. In the Project, System crawls the data from three different restaurants and the crawled data is being saved in the database. And at the same time, it will replace the existing data with the new data.

```
function crawlRestTwoData()
{

    var url = 'http://www.schroedingers.de/speisekarte-schroedingers'
    request(url, function(err, response, html)
    {
        if(!err)
        {
            // Removing The Existing Data Inorder To Update the Database With
            // New Data
            restTwoModel.collection.remove({}, function(err)
            {
                if(err){ throw err; }
                // else {console.log('Restaurant TWO Data removed from db');}
            });
            // restTwoModel.collection.drop();

            // Crawling RestaurantTwo Data
            var $ = cheerio.load(html);
            var dataList2 = $('<div>.pmtitle');
            dataList2.each(function(index)
            {
                restaurantTwo.itemName =
                $('<div>.pmtitle').eq(index).text().trim();
                restaurantTwo.price = $('<div>.pmprice').eq(index).text().trim();
                restaurantTwo.restName = 'Bistro Schroeders';
            }
            }
            }
    }
}
```



```

        restaurantTwo.restDetails = 'Bistro Schroeders
Helmholtzstrasse 23 10587 Berlin. Opening hours: Mon - Fri: 11:00 - 16:00.
Phone: 0160 34 855 38';
        restaurantTwo.timeStamp = new Date();
        // console.log(restaurantTwo);

        // Saving Crawled Data to Database
        restTwoModel.create(restaurantTwo, function(err)
        {
            if(err) { throw err; }
        });

    });
    console.log('Restaurant Two Data Saved Into DB');
}
});
}

```

3.2 REST API

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. Here System is working with GET and POST Methods.

GET Method

For Retrieving information from the server, System uses GET method. When User is performing a `GET` request, the REST API retrieves all data in the database and sends it back to User. In other words, a `GET` request performs a `READ` operation.

```

router.get('/allDishes', function(req, res)
{
    //Combining Data From all Models
    Promise.all(
        [
            restOneModel.find(),

```

```

restTwoModel.find(),
restThreeModel.find()
]).then(allData =>
{
    // Concatinating the data into Single Array
    var combinedData =
allData[0].concat(allData[1]).concat(allData[2]);
    console.log('User Requested For all the Dishes from Database');
    res.json(combinedData);
}).catch(err =>
{
    console.error("something is wrong",err);
})
});

```

POST Method

System is using POST Method in order to accept an input String from the User and send the String inside body of POST Request. REST API receives the string and searches the Database based on the String and returns the Result of the Query to the User

```

router.post('/search', function(req, res)
{
    if(req.body.name)
    {
        var srcitem = req.body.name; //Input String From User
        console.log("The Submitted String is: " + srcitem);

        //Query For Searching Based on Letters
        /* var query = {
            $or: [
                {"itemName": {$regex: "." + srcitem + "."},
$options:"i"}},
                {"itemDesc": {$regex: "." + srcitem + "."},
$options:"i"}},
            ]
        }
        */

        //Query For Searching Based on Keywords

```

```

var query = { $text: {$search: srcitem} } ;

// Combining Data from all models based on query
Promise.all([
    restOneModel.find(query),
    restTwoModel.find(query),
    restThreeModel.find(query)
]).then(allData =>
    {
        // Concatinating the data into Single Array
        var combinedData =
allData[0].concat(allData[1]).concat(allData[2]);
        // console.log(combinedData);
        if(combinedData == '') { console.log('keywords did
not Match'); }

        res.json(combinedData);
    }).catch(err =>
    {
        console.error("something is wrong",err);
    })
}
else { console.log('Empty Data! Please Insert a String'); }
});

```

3.3 Frontend

At the Frontend, System uses JavaScript to send AJAX GET and POST request to REST API.

System also uses JavaScript to handle chunks of data received from the REST API.

4. Practical Demonstration

In DownTown Diner, Main page consist of a Search Box along with a Button to Search the Database. This Web Application is easy to use by writing the Keyword in the Search box and User will get the Dishes from the Restaurants, Prices and Ingredients. Thereafter User can click one of the Dishes to get the Restaurant Details. Apart from this, User can also get a list of all the Available Dishes by clicking on "All Available Dishes" button from the Database.

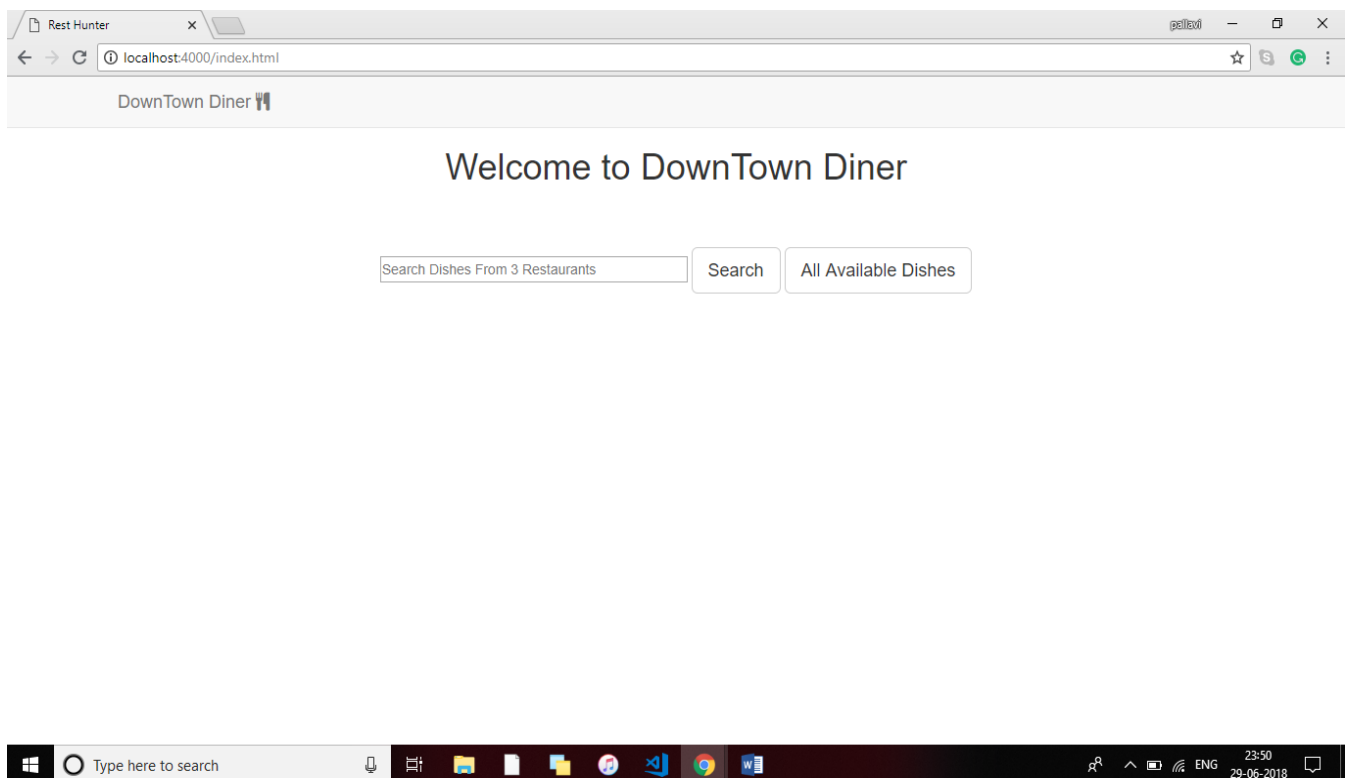


Fig.1 Displaying the Main Page of DownTown Diner

We have added event listener to “Search” and “All Available Dishes” buttons.

```
document.querySelector('form').addEventListener('submit',searchString);  
document.getElementById('allDishes').addEventListener('click',getAllDishes);
```

We are using AJAX method for fetching all the dishes from the database and sending the request.

```
var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function()  
    {  
        if(this.readyState == 4 && this.status == 200)  
        {  
            myArr = JSON.parse(this.responseText);  
            myFunc(myArr); // Sending the Received data to a  
function to process further  
        }  
    }  
}
```

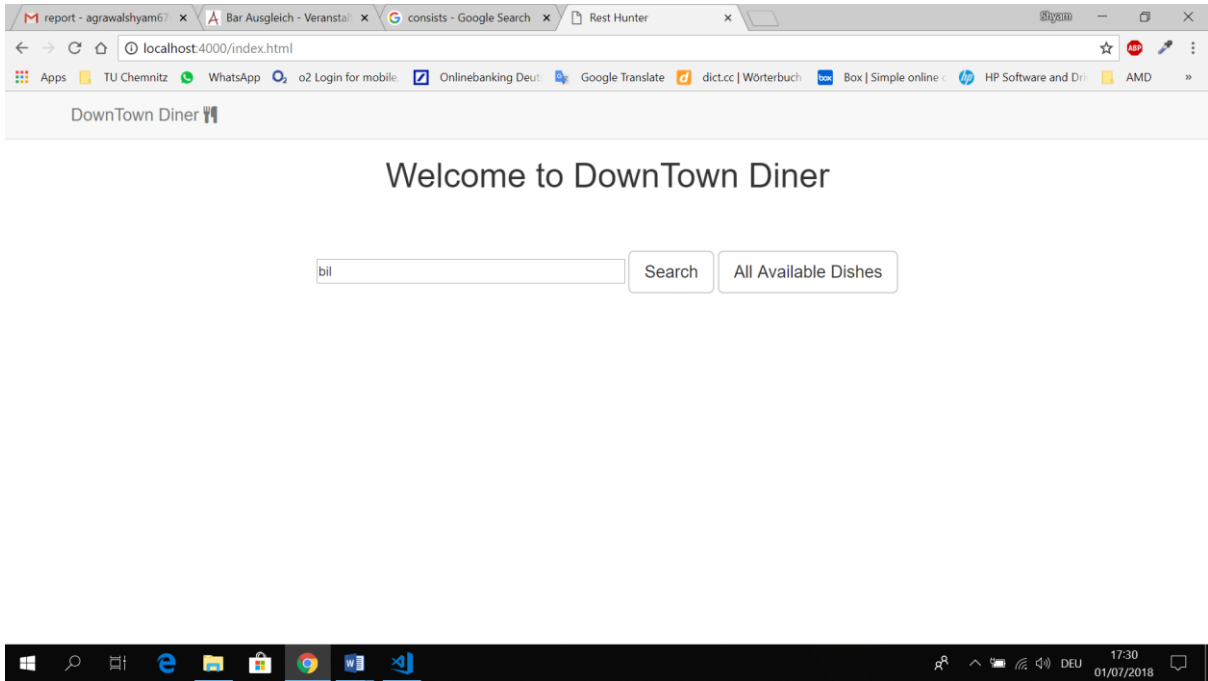


Fig.2 Searching Dishes based on Keywords

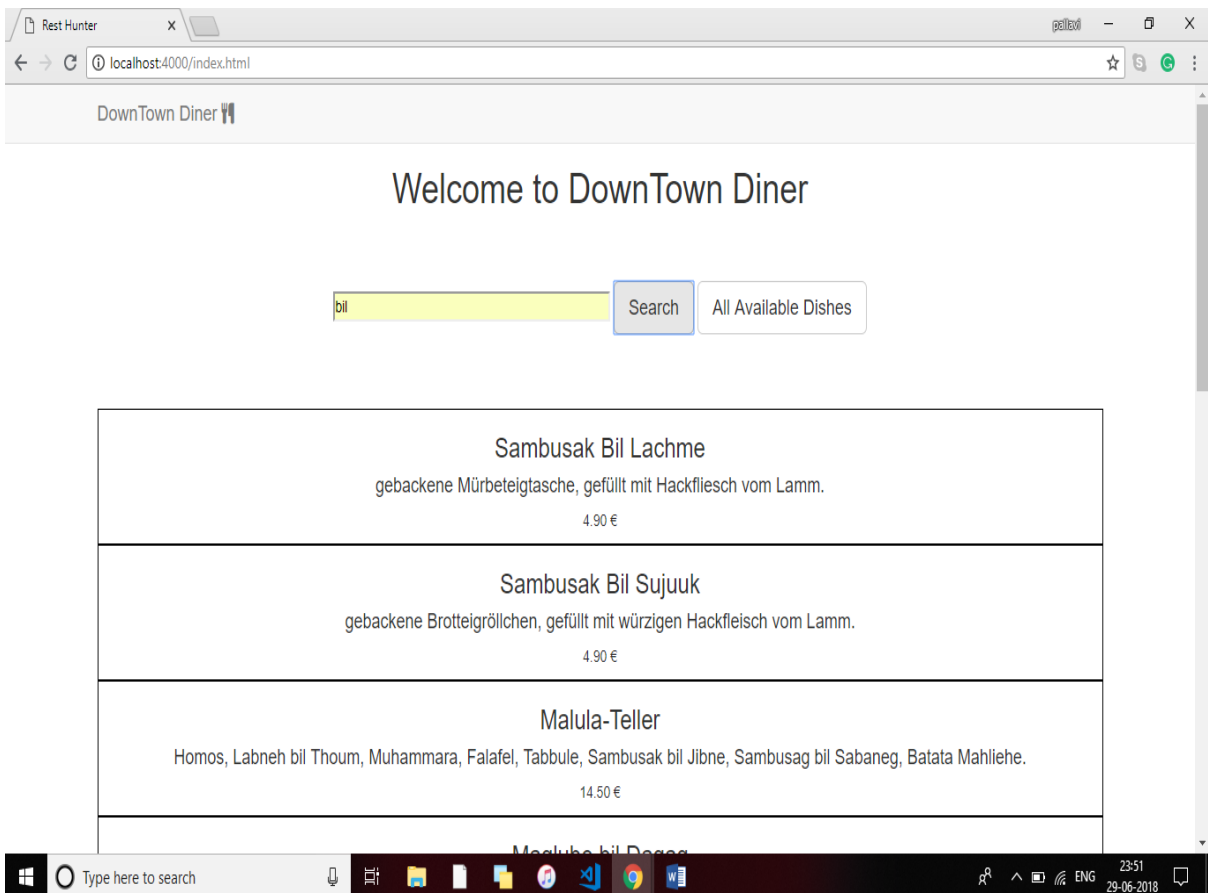


Fig.3 Displays the Dishes from all Restaurant

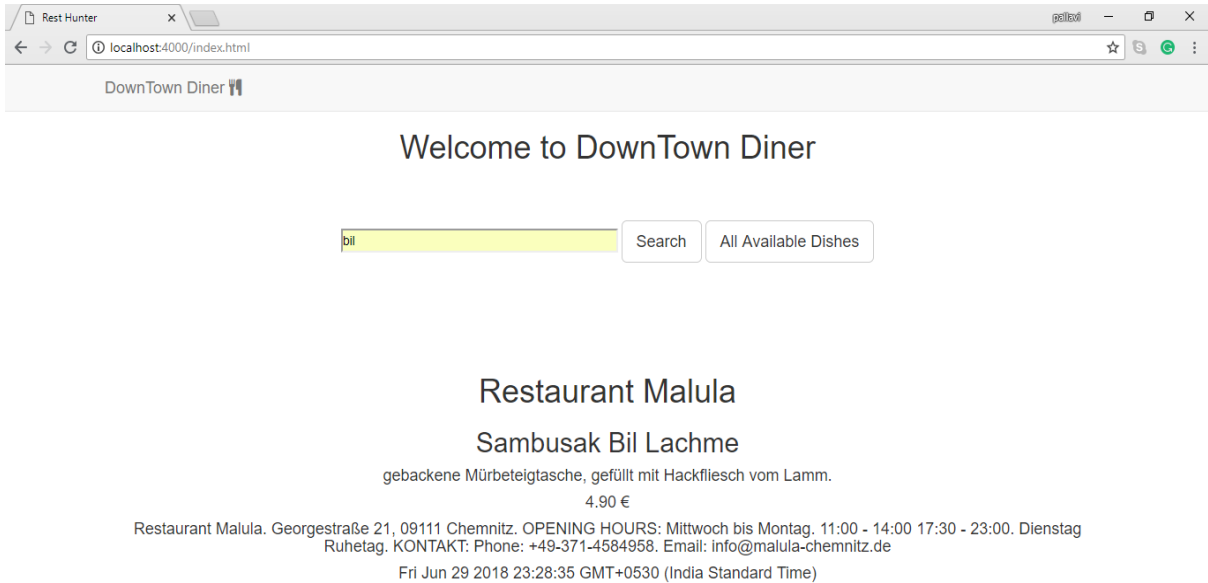


Fig.4 Clicking the Dish Displays the Restaurant Details

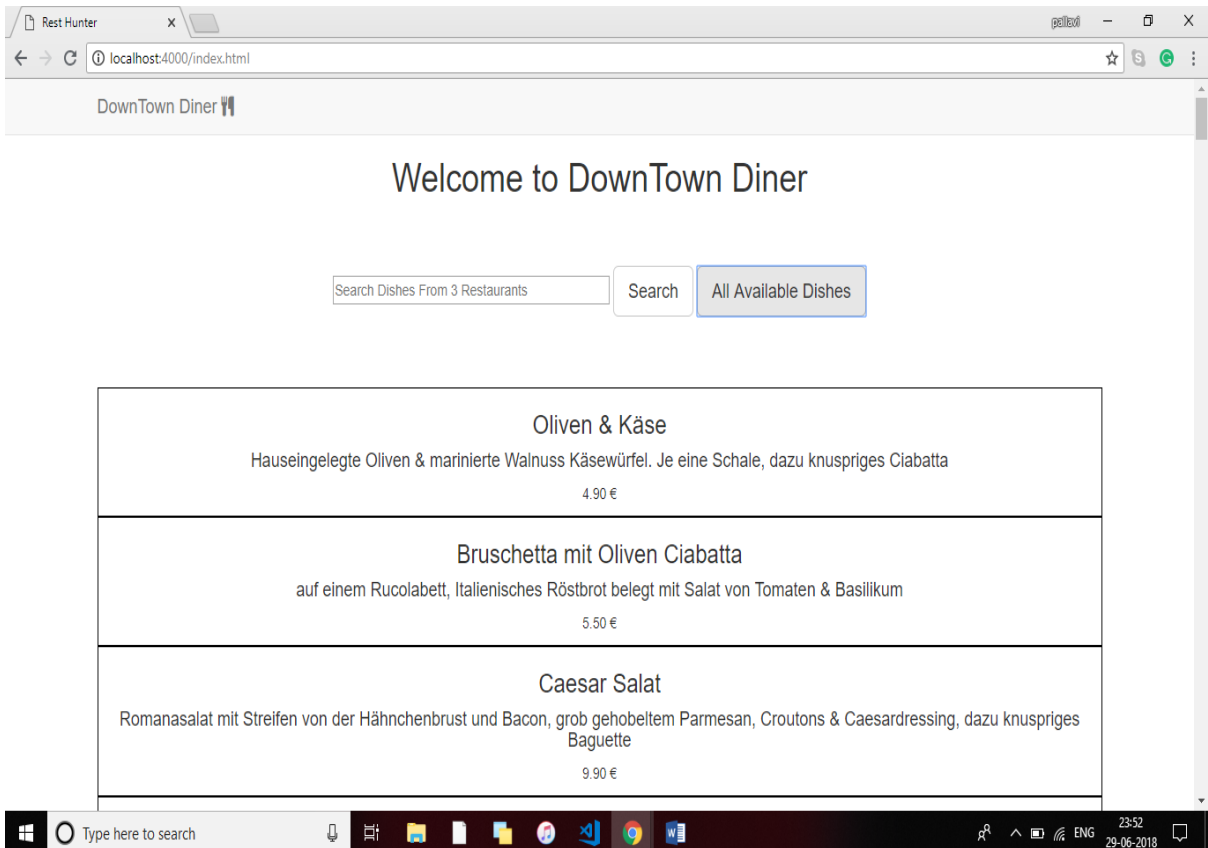


Fig.5 Displays All Available Dishes

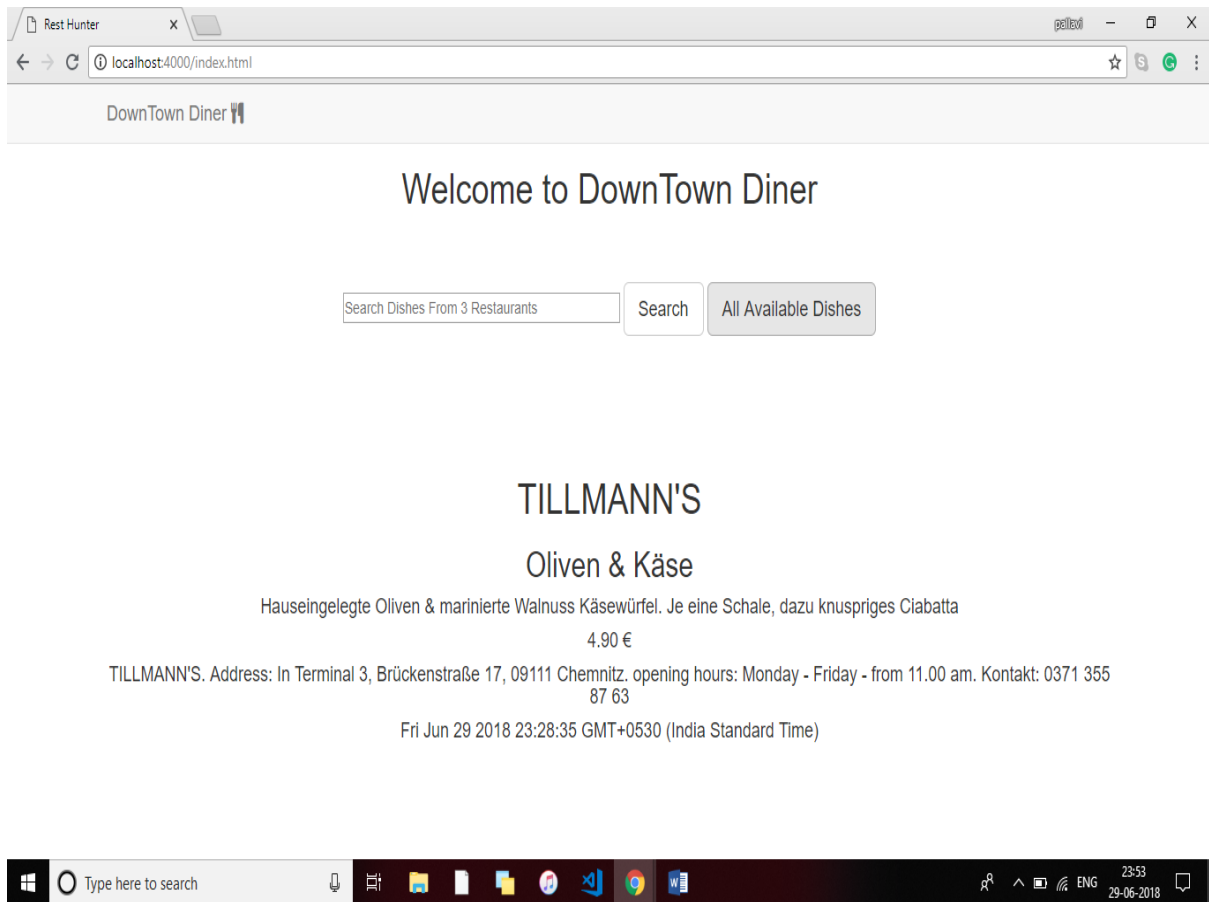


Fig.6 Clicking the Dish Displays the Restaurant Details

5. Conclusion

The Web Application helps to get Dishes from different Restaurants along with their Descriptions. It helps the User to Differentiate between Dishes from different Restaurants.

The Application Crawls and Stores data in Real Time. Overall the Application provides all the necessary Information to the User in a more Efficient way.

6. Appendix

Show All Dishes

Returns an array of JSON data about all the available dishes

- **URL**
/allDishes
- **Method**
GET
- **URL Params**
None
- **Data Params**
None
- **Success Response:**
 - Code: 200
Content:

```
[  
  {  
    "_id": "5b3777774f0f4d155061adef",  
    "itemName": "Oliven & Käse",  
    "itemDesc": "Hauseingelegte Oliven &  
    marinierte Walnuss Käsewürfel. \nJe eine  
    Schale, dazu knuspriges Ciabatta",  
    "price": "4.90 €",  
    "restName": "TILLMANN'S",
```

```
"restDetails": "TILLMANN'S. Address: In  
Terminal 3, Brückenstraße 17, 09111  
Chemnitz. opening hours: Monday - Friday -  
from 11.00 am. Kontakt: 0371 355 87 63 ",  
"timeStamp": "Sat Jun 30 2018 14:28:38  
GMT+0200 (W. Europe Summer Time)"  
},  
{}, {}, {}, ....so on  
]
```

- **Error Response:**

None

- **Sample Call:**

```
var xhttp = new XMLHttpRequest();  
xhttp.onreadystatechange = function()  
{ if(this.readyState == 4 && this.status == 200)  
{  
    Console.log(JSON.parse(this.responseText));  
}  
}  
xhttp.open('GET', '/allDishes', true);  
// xhttp.setRequestHeader('Content-Type',  
'application/json');  
xhttp.send();
```

Show Dishes Based on Search

Returns Array of JSON data about dishes from the Database based on keyword Provided by the User.

- **URL**
/search
- **Method**
POST
- **URL Params**
None
- **Data Params**
{“name”: “salat”}
- **Success Response:**
 - Code: 200
Content:
[
 {
 "_id": "5b3777774f0f4d155061adf4",
 "itemName": "Rucola-Salat",
 "itemDesc": "Bunter Salat von Rucola, mit
Walnüssen, \nTomaten und einer Orangen-
Vinaigrette getoppt, dazu Oliven Ciabatta",
 "price": "7.90 €",
 "restName": "TILLMANN'S",
 "restDetails": "TILLMANN'S. Address: In
Terminal 3, Brückenstraße 17, 09111
Chemnitz. opening hours: Monday - Friday -
from 11.00 am. Kontakt: 0371 355 87 63 ",

```
        "timeStamp": "Sat Jun 30 2018 14:28:39  
        GMT+0200 (W. Europe Summer Time)"  
    },  
    {}, {}, {}, .....so on  
]
```

- **Error Response:**

None

- **Sample Call:**

```
var input1 = document.getElementById('data1').value;  
    console.log(input1);  
    var newEntry = { 'name': input1 };  
    console.log(newEntry);  
  
    var xhttp = new XMLHttpRequest();  
    xhttp.onreadystatechange = function()  
    { if(this.readyState == 4 && this.status == 200)  
      {  
        Console.log(JSON.parse(this.responseText));  
      }  
    }  
  
    xhttp.open('POST', '/search', true);  
    xhttp.setRequestHeader('Content-Type', 'application/json');  
    xhttp.send(JSON.stringify(newEntry));
```