ABIYAANTRIX & SAPIENCE ACADEMY
INTERNSHIP + TRAINING

Internship Mini Project on

"BOOK DATABASE"

Submitted in partial fulfillment towards Mini Project work of Internship

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

SUBMITTED BY

Vivek Pawar	**4CA15CS021**
Shantha Raman S	**4PS15CS100**
Anushree G	**4GW15CS009**
Bhavini Pahwa	**1RN16CS020**
Komal B R	**4GW15CS040**
Prakruthi S	**4GW16CS415**
Akash V	**01JST16IS051**

UNDER THE GUIDANCE OF

Mrs. ANJANA SHASHIKIRAN
DIRECTOR

Mr. SRIKRISHNA S KASHYAP
INTERNSHIP TRAINER

ACKNOWLEDGMENT

We sincerely owe our gratitude to all the persons who helped and guided us in completing this mini-project.

We are thankful to **Mrs. Anjana Shashi Kiran**, *Honorary Director*, Abiyaantrix Tech Solutions, Mysuru, for having supported in our academic endeavours.

We are thankful to **Mr. Srikrishna S Kashyap**, *Trainer*, Abiyaantrix Tech Solutions, Mysuru, for all the support he has rendered.

We are extremely pleased to thank our parents, family members and friends for their continuous support, inspiration and encouragement, for their helping hand and also last but not the least, We thank all the members who supported directly or indirectly in this internship process.

Vivek Pawar
Shantha Raman S
Anushree G
Bhavini Pahwa
Komal B R
Prakruthi S
Akash V

ABSTRACT

It is the process by which order, structure and meaning are given to the data (information).

It consists in transforming the collected data into useful and true conclusions and or lessons.

From the pre-established topics, the data are processed, looking for trends, differences and variations in the information obtained.

The processes, techniques and tools used are based on certain assumptions and as such have limitations.

The process is used to describe and summarize the data, identify the relationships and differences between variables, compare variables and make predictions.

CONTENTS

INTRODUCTION

A **database** is an organized collection of data, stored and accessed electronically. Database designers typically organize the data to model aspects of reality in a way that supports processes requiring information, such as (for example) modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

A **database management system** (**DBMS**) is a softwareapplication that interacts with end users, other applications, and the database itself to capture and analyse data. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases. A database is generally stored in a DBMS-specific format which is not portable, but different DBMSs can share data by using standards such as SQL and ODBC or JDBC. Sometimes a DBMS is loosely referred to as a "database".

Computer scientists may classify database-management systems according to the database models that they support. Relational databases became dominant in the 1980s. These model data as rows and columns in a series of tables, and the vast majority use SQL for writing and querying data. In the 2000s, non-relational databases became popular, referred to as NoSQL because they use different query languages.

Database features

Because of the critical importance of database technology to the smooth running of an enterprise, database systems include complex mechanisms to deliver the required performance, security, and availability, and allow database administrators to control the use of these features.

Storage

Database storage is the container of the physical materialization of a database. It comprises the *internal* (physical) *level* in the database architecture. It also contains all the information needed (e.g., metadata, "data about the data", and internal data structures) to reconstruct the *conceptual level* and *external level* from the internal level when needed. Putting data into permanent storage is generally the responsibility of the database engine a.k.a. "storage engine". Though typically accessed by a DBMS through the underlying operating system (and often utilizing the operating systems' file systems as intermediates for storage layout), storage properties and configuration setting are extremely important for the efficient operation of the DBMS, and thus are closely maintained by database administrators. A DBMS, while in operation, always has its database residing in several types of storage (e.g., memory and external storage). The database data and the additional needed information, possibly in very large amounts, are coded into bits. Data typically reside in the storage in structures that look completely different from the way the data look in the conceptual and external levels, but in ways that attempt to optimize (the best possible) these levels' reconstruction when needed by users and programs, as well as for computing additional types of needed information from the data (e.g., when querying the database).

Some DBMSs support specifying which character encoding was used to store data, so multiple encodings can be used in the same database.

Various low-level database storage structures are used by the storage engine to serialize the data model so it can be written to the medium of choice. Techniques such as indexing may be used to improve performance. Conventional storage is row-oriented, but there are also column-oriented and correlation databases.

Materializedviews

Often storage redundancy is employed to increase performance. A common example is storing *materialized views*, which consist of frequently needed *external views* or query results. Storing such views saves the expensive computing of them each time they are needed. The downsides of materialized views are the overhead incurred when updating them to keep them synchronized with their original updated database data, and the cost of storage redundancy.

Replication

Occasionally a database employs storage redundancy by database objects replication (with one or more copies) to increase data availability (both to improve performance of simultaneous multiple end-user accesses to a same database object, and to provide resiliency in a case of partial failure of a distributed database). Updates of a replicated object need to be synchronized across the object copies. In many cases, the entire database is replicated.

Security

Database security deals with all various aspects of protecting the database content, its owners, and its users. It ranges from protection from intentional unauthorized database uses to unintentional database accesses by unauthorized entities (e.g., a person or a computer program).

Database access control deals with controlling who (a person or a certain computer program) is allowed to access what information in the database. The information may comprise specific database objects (e.g., record types, specific records, data structures), certain computations over certain objects (e.g., query types, or specific queries), or utilizing specific access paths to the former (e.g., using specific indexes or other data structures to access information). Database access controls are set by special authorized (by the database owner) personnel that uses dedicated protected security DBMS interfaces.

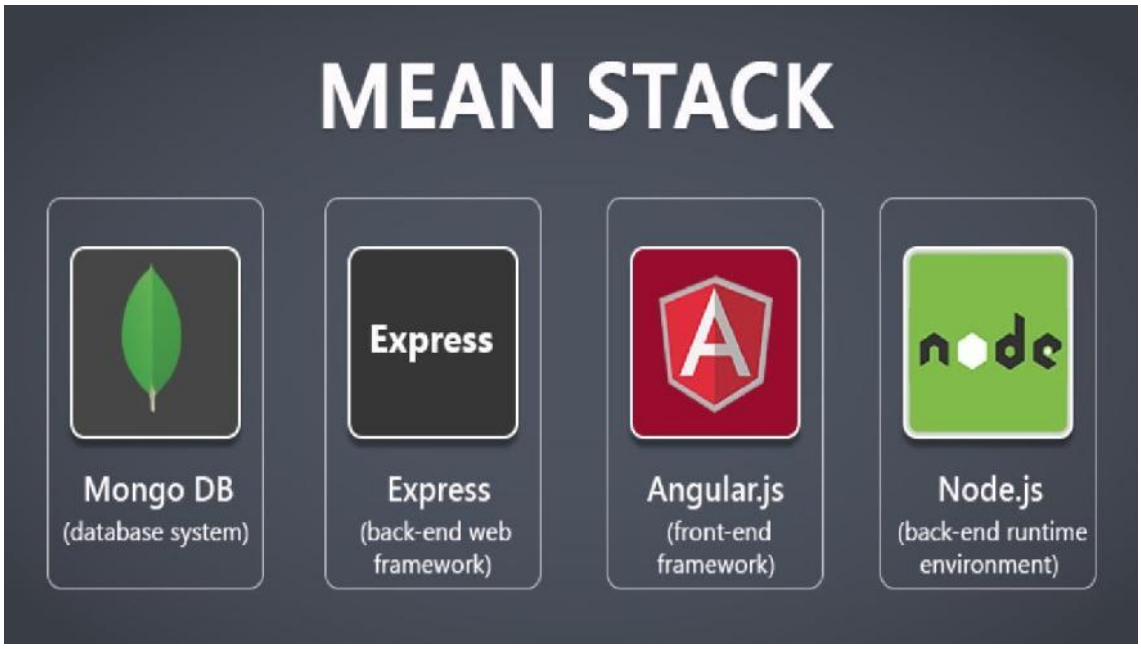
This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called "subschemas".

For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases.

Data security in general deals with protecting specific chunks of data, both physically (i.e., from corruption, or destruction, or removal; e.g., see physical security), or the interpretation of them, or parts of them to meaningful information (e.g., by looking at the strings of bits that they comprise, concluding specific valid credit-card numbers; e.g., see data encryption).

Change and access logging records who accessed which attributes, what was changed, and when it was changed. Logging services allow for a forensic database audit later by keeping a record of access occurrences and changes. Sometimes application-level code is used to record changes rather than leaving this to the database. Monitoring can be set up to attempt to detect security breaches.

MEAN STACK



MEAN is a free and open-sourceJavaScriptsoftware stack for building dynamic web sites and web applications.

The MEAN stack is MongoDB, Express.js, AngularJS (or Angular), and Node.js. Because all components of the MEAN stack support programs are written in JavaScript, MEAN applications can be written in one language for both server-side and client-side execution environments.

SYSTEMREQUIREMENTANDSPECIFICATION

2.1HardwareRequirements:

- Intel® Pentium 4 CPU and higher versions
- 256 MB RAM,
- 80GBHDD Mouse
- QWERTY Keyboard
- Standard VGA Monitor

2.2SoftwareRequirements:

- OPERATING SYSTEM : WINDOWS 10, 7
- LANGUAGE :JS , HTML
- TOOL : MEAN STACK

TESTING AND RESULTS

The full creating and implementing Book Database. Storing Book details such as ISBN(International Standard Book Number) , Book title, Author name ,Book Published Year and publishing house. Using Angular The following tools, frameworks, and modules are: Node.js (recommended version), Angular CLI 1.5, Angular 5, Express.js, mongoDB, mongoose.js

UNIT TESTING

Here the individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

MODULE TESTING

Module is a collection of dependent components such as procedures and functions. Since the module encapsulates related components can be tested with our other system modules. The testing process is concerned with finding errors which results from erroneous function calls from the main function to various individual functions.

SYSTEM TESING

The Modules are integrated to make up the entire system. The testing process is concerned with finding errors with the results from unanticipated interactions between module and system components. It is also concerned with validating that the system meets its functional and non-functional requirements.

Implementation Code

#to create server

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('mean-app:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
```

```
/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
```

```
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
"scripts": {
  "ng": "ng",
  "start": "ng build && node ./bin/www",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
```

#app.js

```
var express = require('express');
var path = require('path');
var favicon = require('serve-favicon');
var logger = require('morgan');
var bodyParser = require('body-parser');
var mongoose = require('mongoose');
mongoose.Promise = require('bluebird');
mongoose.connect('mongodb://localhost/mean-angular5', { useMongoClient: true,
promiseLibrary: require('bluebird') })
  .then(() =>  console.log('connection succesful'))
  .catch((err) => console.error(err));

var book = require('./routes/book');
var app = express();

app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({'extended':'false'}));
app.use(express.static(path.join(__dirname, 'dist')));
app.use('/books', express.static(path.join(__dirname, 'dist')));
app.use('/book', book);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  var err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
```

```
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
#book.js
var express = require('express');
var mongoose = require('mongoose');
var router = express.Router();

/* GET home page. */
router.get('/', function(req, res, next) {
  res.send('Express RESTful API');
});
var BookSchema = new mongoose.Schema({
  isbn: String,
  title: String,
  author: String,
  description: String,
  published_year: String,
  publisher: String,
  updated_date: { type: Date, default: Date.now },
});

module.exports = router;
#route/book.js
var express = require('express');
var router = express.Router();
var mongoose = require('mongoose');
var Book = require('../models/Book.js');
```

```
/* GET ALL BOOKS */
router.get('/', function(req, res, next) {
  Book.find(function (err, products) {
    if (err) return next(err);
    res.json(products);
  });
});

/* GET SINGLE BOOK BY ID */
router.get('/:id', function(req, res, next) {
  Book.findById(req.params.id, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

/* SAVE BOOK */
router.post('/', function(req, res, next) {
  Book.create(req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

/* UPDATE BOOK */
router.put('/:id', function(req, res, next) {
  Book.findByIdAndUpdate(req.params.id, req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});
```

```
/* DELETE BOOK */
router.delete('/:id', function(req, res, next) {
  Book.findByIdAndRemove(req.params.id, req.body, function (err, post) {
    if (err) return next(err);
    res.json(post);
  });
});

module.exports = router;
```

#src/app/book/book.component.html

```
<div class="container">
  <h1>Book List</h1>
  <table class="table">
    <thead>
      <tr>
        <th>Title</th>
        <th>Author</th>
        <th>Action</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let book of books">
        <td>{{ book.title }}</td>
        <td>{{ book.author }}</td>
        <td>Show Detail</td>
      </tr>
    </tbody>
  </table>
</div>
```

#index . HTML

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
```

```
  <title>MeanAngular5</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <!-- Latest compiled and minified CSS -->
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
  <!-- Optional theme -->
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-
theme.min.css" integrity="sha384-
rHyoN1iRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwl/Sp"
crossorigin="anonymous">
</head>
<body>
  <app-root></app-root>
  <!-- Latest compiled and minified JavaScript -->
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
crossorigin="anonymous"></script>
</body>
</html>
#app.module.ts
const appRoutes: Routes = [
  {
    path: 'books',
    component: BookComponent,
    data: { title: 'Book List' }
  },
  {
```

```
    path: 'book-details/:id',
    component: BookDetailComponent,
    data: { title: 'Book Details' }
  },
  { path: '',
    redirectTo: '/books',
    pathMatch: 'full'
  }
];
const appRoutes: Routes = [
  {
    path: 'books',
    component: BookComponent,
    data: { title: 'Book List' }
  },
  {
    path: 'book-details/:id',
    component: BookDetailComponent,
    data: { title: 'Book Details' }
  },
  {
    path: 'book-create',
    component: BookCreateComponent,
    data: { title: 'Create Book' }
  },
  { path: '',
    redirectTo: '/books',
    pathMatch: 'full'
  }
];
```

#src/app/book-detail/book-detail.component.ts

```
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { ActivatedRoute } from '@angular/router';
```

```
@Component({
  selector: 'app-book-detail',
  templateUrl: './book-detail.component.html',
  styleUrls: ['./book-detail.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class BookDetailComponent implements OnInit {

  book = {};

  constructor(private route: ActivatedRoute, private http: HttpClient) { }

  ngOnInit() {
    this.getBookDetail(this.route.snapshot.params['id']);
  }

  getBookDetail(id) {
    this.http.get('/book/'+id).subscribe(data => {
      this.book = data;
    });
  }

}
```

#src/app/book-detail/book-detail.component.html

```
<div class="container">
  <h1>{{ book.title }}</h1>
  <dl class="list">
    <dt>ISBN</dt>
    <dd>{{ book.isbn }}</dd>
    <dt>Author</dt>
    <dd>{{ book.author }}</dd>
    <dt>Publisher</dt>
    <dd>{{ book.publisher }}</dd>
```

```
    <dt>Price</dt>
    <dd>{{ book.price }}</dd>
    <dt>Update Date</dt>
    <dd>{{ book.updated_at }}</dd>
  </dl>
</div>
```

#Src/app/book/book-create.component.ts

```
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-book-create',
  templateUrl: './book-create.component.html',
  styleUrls: ['./book-create.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class BookCreateComponent implements OnInit {

  book = {};

  constructor(private http: HttpClient, private router: Router) { }

  ngOnInit() {
  }

  saveBook() {
    this.http.post('/book', this.book)
      .subscribe(res => {
          let id = res['_id'];
          this.router.navigate(['/book-details', id]);
        }, (err) => {
          console.log(err);
        }
```

```
    );
  }

}
```

#src/app/book-create/book-create.component.html

```
<div class="container">
  <h1>Add New Book</h1>
  <div class="row">
    <div class="col-md-6">
      <form (ngSubmit)="saveBook()" #bookForm="ngForm">
        <div class="form-group">
          <label for="name">ISBN</label>
          <input type="text" class="form-control" [(ngModel)]="book.isbn" name="isbn" required>
        </div>
        <div class="form-group">
          <label for="name">Title</label>
          <input type="text" class="form-control" [(ngModel)]="book.title" name="title" required>
        </div>
        <div class="form-group">
          <label for="name">Author</label>
          <input type="text" class="form-control" [(ngModel)]="book.author" name="author" required>
        </div>
        <div class="form-group">
          <label for="name">Published Year</label>
          <input type="number" class="form-control" [(ngModel)]="book.published_year" name="published_year" required>
        </div>
        <div class="form-group">
          <label for="name">Publisher</label>
          <input type="text" class="form-control" [(ngModel)]="book.publisher" name="publisher" required>
```

```
      </div>
      <div class="form-group">
        <button type="submit" class="btn btn-success"
[disabled]="!bookForm.form.valid">Save</button>
      </div>
    </form>
   </div>
  </div>
</div>
```

#src/app/book-edit/book-edit.component.ts

```
import { Component, OnInit, ViewEncapsulation } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-book-edit',
  templateUrl: './book-edit.component.html',
  styleUrls: ['./book-edit.component.css'],
  encapsulation: ViewEncapsulation.None
})
export class BookEditComponent implements OnInit {

  book = {};

  constructor(private http: HttpClient, private router: Router, private route: ActivatedRoute) {
}

  ngOnInit() {
    this.getBook(this.route.snapshot.params['id']);
  }

  getBook(id) {
    this.http.get('/book/'+id).subscribe(data => {
```

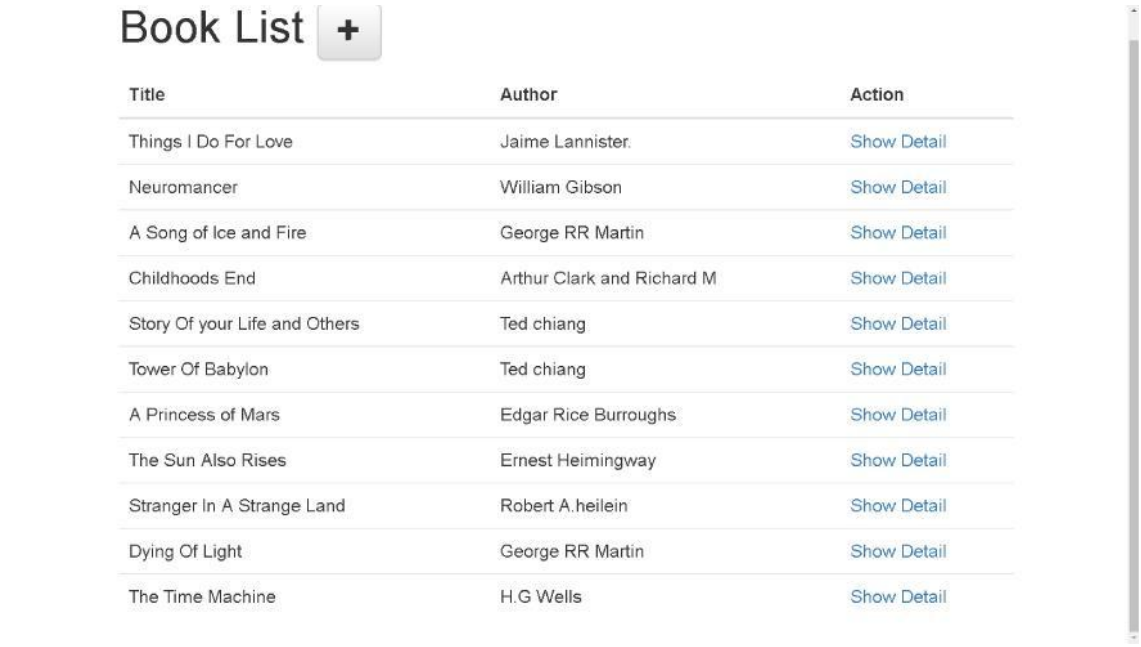
```
      this.book = data;
    });
  }

  updateBook(id, data) {
    this.http.put('/book/'+id, data)
      .subscribe(res => {
          let id = res['_id'];
          this.router.navigate(['/book-details', id]);
        }, (err) => {
          console.log(err);
        }
      );
  }

}
```

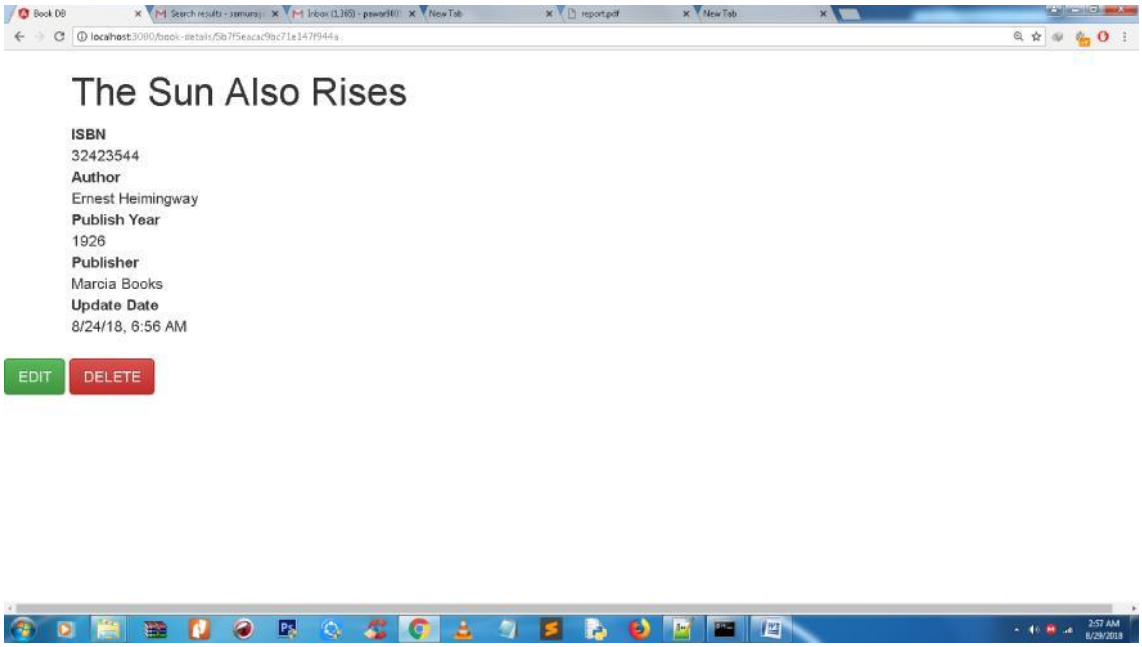
#Src/app/book-edit/book-edit.component.html

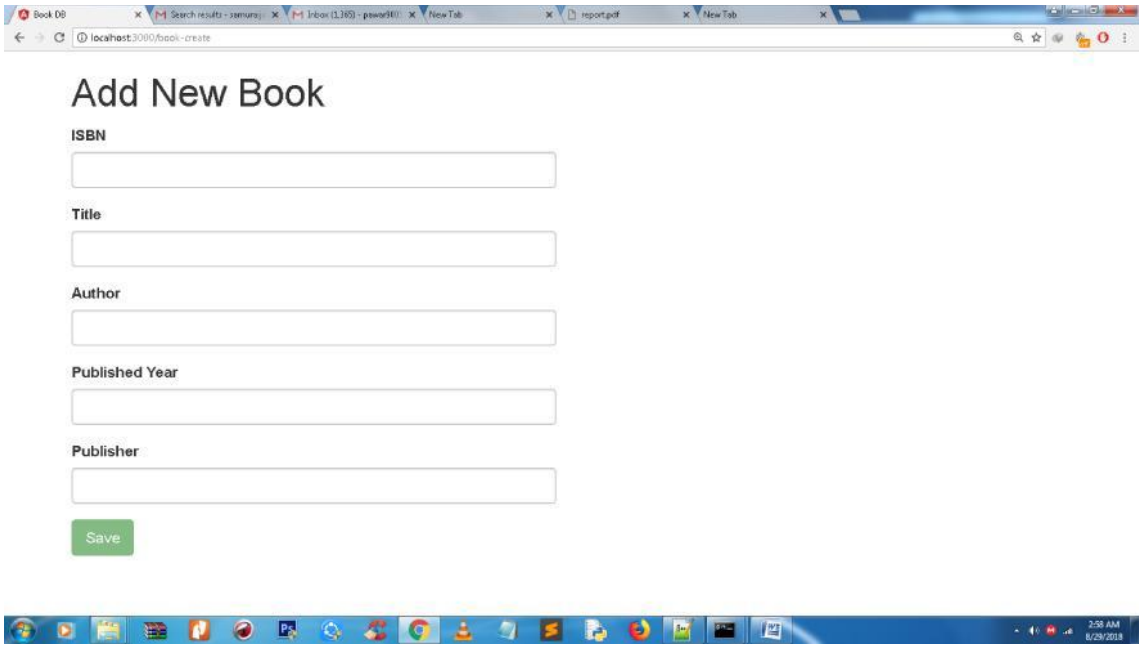
```
<div class="container">
  <h1>Edit Book</h1>
  <div class="row">
    <div class="col-md-6">
      <form (ngSubmit)="updateBook(book._id)" #bookForm="ngForm">
        <div class="form-group">
          <label for="name">ISBN</label>
          <input type="text" class="form-control" [(ngModel)]="book.isbn" name="isbn"
required>
        </div>
        <div class="form-group">
          <label for="name">Title</label>
          <input type="text" class="form-control" [(ngModel)]="book.title" name="title"
required>
        </div>
```

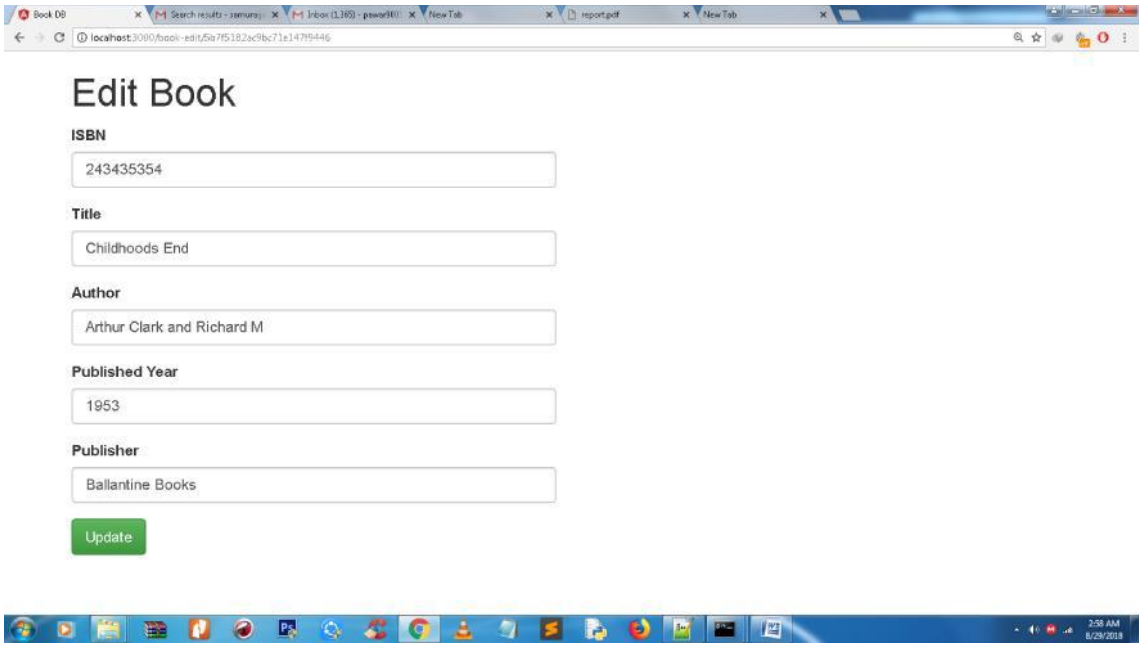
```
      <div class="form-group">
        <label for="name">Author</label>
        <input type="text" class="form-control" [(ngModel)]="book.author" name="author" required>
      </div>
      <div class="form-group">
        <label for="name">Published Year</label>
        <input type="number" class="form-control" [(ngModel)]="book.published_year" name="published_year" required>
      </div>
      <div class="form-group">
        <label for="name">Publisher</label>
        <input type="text" class="form-control" [(ngModel)]="book.publisher" name="publisher" required>
      </div>
      <div class="form-group">
        <button type="submit" class="btn btn-success" [disabled]="!bookForm.form.valid">Update</button>
      </div>
    </form>
  </div>
 </div>
</div>
```

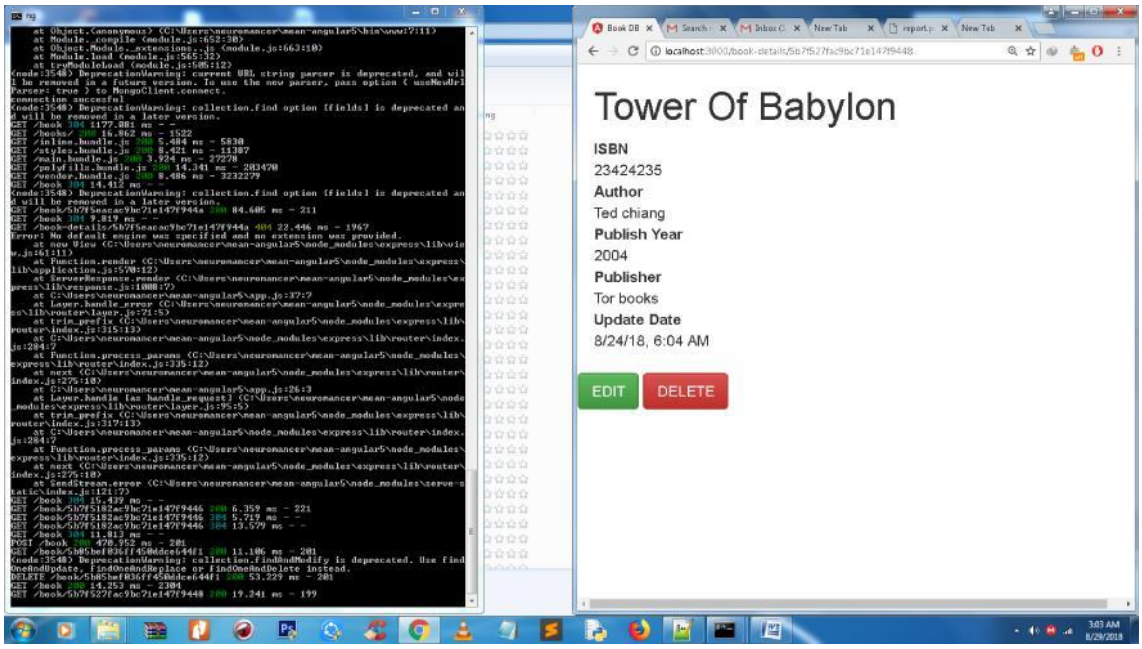
SNAPSHOTS

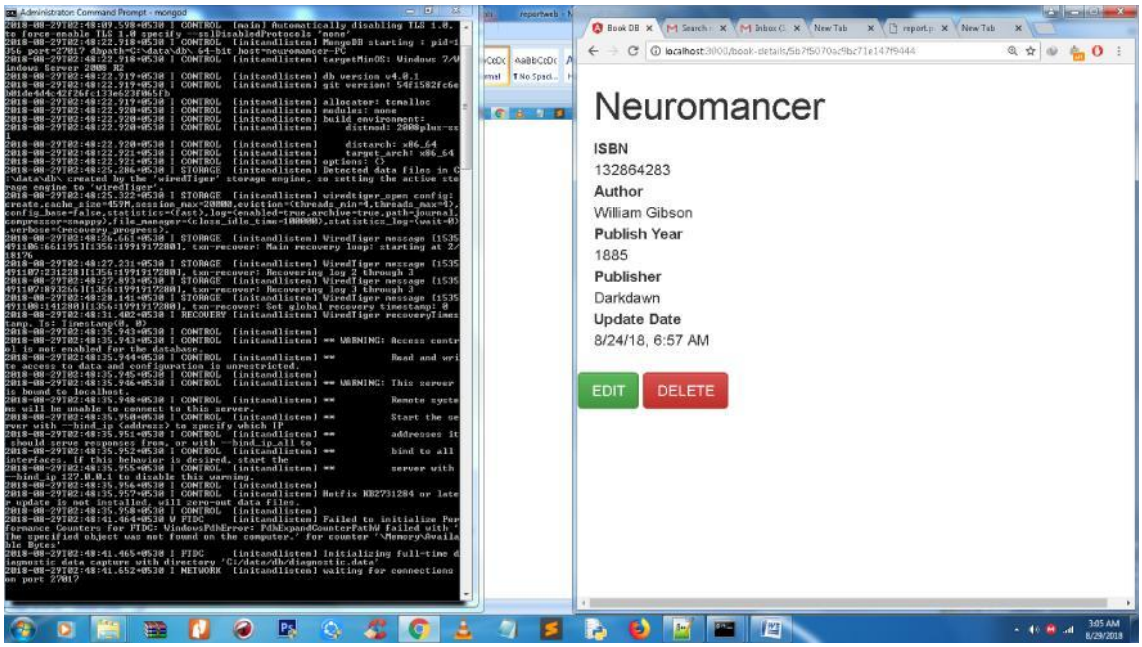

Book List +
Title
Author
Action
Things I Do For Love
Jaime Lannister.
Show Detail
Neuromancer
William Gibson
Show Detail
A Song of Ice and Fire
George RR Martin
Show Detail
Childhoods End
Arthur Clark and Richard M
Show Detail
Story Of your Life and Others
Ted chiang
Show Detail
Tower Of Babylon
Ted chiang
Show Detail
A Princess of Mars
Edgar Rice Burroughs
Show Detail
The Sun Also Rises
Ernest Heimingway
Show Detail
Stranger In A Strange Land
Robert A.heilein
Show Detail
Dying Of Light
George RR Martin
Show Detail
The Time Machine
H.G Wells
Show Detail


The Sun Also Rises
ISBN
32423544
Author
Ernest Heimingway
Publish Year
1926
Publisher
Marcia Books
Update Date
8/24/18, 6:56 AM
EDIT
DELETE


Book DB
localhost:3000/book-create
Add New Book
ISBN
Title
Author
Published Year
Publisher
Save


Book DB
localhost:3000/book-edit/5a7f5182ac9bc71e147f9446
Edit Book
ISBN
243435354
Title
Childhoods End
Author
Arthur Clark and Richard M
Published Year
1953
Publisher
Ballantine Books
Update


Tower Of Babylon
ISBN
23424235
Author
Ted chiang
Publish Year
2004
Publisher
Tor books
Update Date
8/24/18, 6:04 AM
EDIT
DELETE


Neuromancer
ISBN
132864283
Author
William Gibson
Publish Year
1885
Publisher
Darkdawn
Update Date
8/24/18, 6:57 AM
EDIT
DELETE

FUTURE ENHANCEMENTS

We have successfully implemented the book database. We can enhance the experience of this application by using various interfaces, data formats.

We can add images of book cover and store it in database, fetch when needed

Add short summary description of the story or book context.

CONCLUSION

A book database keeps all the information together and it can be easily moved around on a laptop or pen-drive or memory card. It stores a large amount of information in easily retrievable way. It can be accessed simultaneously by multiple users over a network and it can also be made very secure (using the relevant security protocols and applications).

BIBLIOGRAPHY

- Newspaper article
- Wikipedia.org
- Techopedia.com
- Stackoverflow.com
- Quora.com
- Elitedatascience.com/data-cleaning
- Trifecta.com
- Mean.io
- Searchbusinessanalytics.techtarget.com
- Optimizely.com
- Geeksforgeeks.com
- Youtube.com
- Studytonight.com
- Techterms.com
- Google images
- Codingdojo.com
- Github.com
- Hackermoon.com