Express + Mongo

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.
www.fandsindia.com

Ground Rules

- Turn off cell phone. If you cannot please keep it on silent mode. You can go out and attend your call.
- If you have any questions or issues please let me know immediately.
- Let us be punctual.

Agenda

TypeScript

Typescript

- Why
  - JavaScript is dynamic type
    - + Can hold any object, type on the fly
    - - Can get messy over the time
  - Hard to manage, difficult to ensure property types
- What
  - Any valid JavaScript is a Typescript
  - Typescriptlang.org
    - Typescript lets you write JavaScript the way you really want to.
    - Typescript is a typed superset of JavaScript that compiles to plain JavaScript.
    - Any browser. Any host. Any OS. Open Source.

Typescript Key Features

- Supports standard JavaScript code
- Provide static typing
- Encapsulation through classes and modules
- Support for constructors, properties, functions
- Define interfaces
- Lambda style function support
- Intellisense and syntax checking

Typescript Basic Data Types

- Boolean
- Number
- String
- Array
- Enum
- Enum as Bit Flag 1,2,4,8,16,32,64,128 and so on
- Any
- Void

TypeScript Functions

- Named / Anonymous Functions
- Functions with Parameter Types
- Optional(lastName?: string)
- Default Parameters (lastName = "Smith“)

Rest Parameters

- Required, optional, and default parameters all have one thing in common: they talk about one parameter at a time. Sometimes, you want to work with multiple parameters as a group, or you may not know how many parameters a function will ultimately take.

```
function buildName(firstName: string, ...restOfName: string[]) {
    return firstName + " " + restOfName.join(" ");
}

// employeeName will be "Joseph Samuel Lucas MacKinzie"
let employeeName = buildName("Joseph", "Samuel", "Lucas", "MacKinzie");
```

Interfaces

```
interface LabeledValue {
    label: string;
}
```

- Interfaces are used at design time to provide auto completion and at compile time to provide type checking
- Supported features
  - Optional properties
  - Function Types
  - Array Types
  - Class Types
  - Extending Interface
  - Hybrid Types

```
interface ClockInterface {
    currentTime: Date;
    setTime(d: Date): void;
}

class Clock implements ClockInterface {
    currentTime: Date = new Date();
    setTime(d: Date) {
        this.currentTime = d;
    }
    constructor(h: number, m: number) { }
}
```

Classes

- Class with variables and functions
- Inheritance
- Public, private, and protected modifiers
  - By default public

```
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");
```

```
class Animal {
    move(distanceInMeters: number = 0) {
        console.log(`Animal moved ${distanceInMeters}m.`);
    }
}

class Dog extends Animal {
    bark() {
        console.log('Woof! Woof!');
    }
}
```

Enums

- Enums allow us to define a set of named constants. Using enums can make it easier to document intent, or create a set of distinct cases. TypeScript provides both numeric and string-based enums.

```
enum Direction {
    Up = 1,
    Down,
    Left,
    Right,
}
```

Array & Tuple

- Array
  - let list: number[] = [1, 2, 3];
  - let list: Array<number> = [1, 2, 3];
- Tuple
  - allows you to express an array with a fixed number of elements whose types are known, but need not be the same
  - **let** x: [string, number];
  - x = ["hello", 10];

Assignment - TypeScript

- Create a Connection interface with open and close method
  - open (url, login , password)--> password as optional parameter
- Create classes OracleConnection & SQLConnection implementing connection
- Create Emp (empno, ename, salary)
- Create EmpDAO class
  - constructor to accept Connection object
  - Variable - arr:Array<Emp> = new Array()
  - Methods
    - Insert - accept Emp object -> con.open(); console.log("in insert"); con.close()
    - List - list the employees.

Express

Express

- Fast, unopinionated, minimalist web framework for Node
- Like any abstraction, Express hides difficult bits and says "don't worry, you don't need to understand this part".
- Base for many other frameworks
- Develop
  - Web Applications
    - Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.
  - APIs
    - With a variety of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Installation (JavaScript)

- In a new folder
  - npm init
    - To create package.json
  - npm install express –save
    - Modify package.json to include dependency
  - Create index.js

```
const express = require('express')
const app = express()
app.get('/', (req, res) => res.send('Hello World!'))
app.listen(3000, () => console.log('Example app listening at http://localhost:3000'))
```

  - Node index.js
  - Invoke code from postman

Installation (TypeScript)

- In a new folder
  1) Create package.json
     npm init or npm init -y(no questions)
  2) Install dependencies (local/global)
     npm install --save-dev typescript@3.3.3
     npm install --save-dev tslint@5.12.1
     npm install --save express@4.16.4
     npm install --save-dev @types/express@4.16.1
  3) Create tsconfig.json
     node_modules\.bin\tsc –init

Installation (TypeScript)

- In a same folder
  1) Create tslint.json
     ./node_modules/.bin/tslint –init
  2) Modify package.json
     To include script line
     "start": "tsc && node dist/index.js“
  3) Create index.ts
     On next page..
  4) Run the code
     Npm start
  5) Test from postman

Index.ts

```
import express from 'express';

const app = express();
app.get('/', (req, res) => {
  res.send('Hello World !!');
});
app.listen(3000, err => {
  if (err) {
    return console.error(err);
  }
  return console.log(`server is listening on 3000`);
});
```

Assignment + Things to Check

- Implement the same
- Generated file
  - Index.js
- Creating different folders for source code and generated code
  - Modify package.json
  - Modify tsconfig.json

Express Routing

Routing

- Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).
- Each route can have one or more handler functions, which are executed when the route is matched.

Routing Method

- app.METHOD(PATH, HANDLER)
- Where:
  - app is an instance of express.
  - METHOD is an HTTP request method, in lowercase.
  - PATH is a path on the server.
  - HANDLER is the function executed when the route is matched.
- E.g.

```
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

Routing Examples

- Simple Get/Post/Put/Delete

```
app.get('/', (req, res)=> res.send('Get Request Handled'))
app.post('/',(req, res) => res.send('Post Request Handled'))
app.put('/', (req,res)=> res.send('Put request Handled'))
app.delete('/',(req, res) => res.send('Delete request Handled'))
```

- Catch All

```
app.get('/*',(req, res) => res.send('CatchAll request Handled'))
```

- Regular Expression
  - The characters ?, +, *, and () are subsets of their regular expression counterparts. The hyphen (-) and the dot (.) are interpreted literally by string-based paths.

Simple Parameters (REST)

```
app.get("/hello/:who", (req, res)=> {
    res.send("Hello, " + req.params.who + ".");
});

app.get("/add/:n1/:n2", (req, res)=> {
    const s1 = +req.params.n1+ +req.params.n2;
    res.send("Hello, " + s1  + ".");
});
```

Get Request Parameters

- To pass parameters
  - http://localhost:8080/process?username=test&password=W124
- To access parameters
  - var user_id = request.query.username;

Post Request Parameters

for express > 4.16, for older verson - use body-parser

```
app.use(express.urlencoded({ extended: true }))
app.use(express.json())

app.post('/', (req, res)=>{
        var user_id = req.body.uid;
        var user_password = req.body.pass;
        res.send('Post Request Handled for ' +
user_id + ", "+ user_password)
        })
```

Assignment

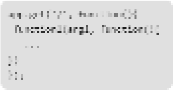
- Implement
  - # get all items
    - GET items/
  - # get a single item using an id parameter
    - GET items/:id
  - # create an item POST
    - items/
  - # update an item
    - PUT items/
  - # delete an item using an id parameter
    - DELETE items/:id

Other Response Methods

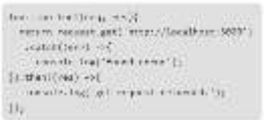
Method	Description
res.download()	Prompt a file to be downloaded.
res.end()	End the response process.
res.json()	Send a JSON response.
res.jsonp()	Send a JSON response with JSONP support.
res.redirect()	Redirect a request.
res.render()	Render a view template.
res.send()	Send a response of various types.
res.sendFile()	Send a file as an octet stream.
res.sendStatus()	Set the response status code and send its string representation as the response body.

Need of Async/Await in Node

- Older version – callbacks
  - Callback hell



- Promise API
  - More like function chaining



Async/Await

- With Node v8, the async/await feature was officially rolled out by the Node to deal with Promises and function chaining. The functions need not to be chained one after another, simply **await** the function that returns the Promise. But the function **async** needs to be declared before **awaiting** a function returning a Promise
- This suddenly (and magically) makes asynchronous programming as easy as synchronous programming. Three things needed for this thought experiment are:
  - Ability to *pause function* execution.
  - Ability to *put a value inside* the function.
  - Ability to *throw an exception inside* the function.

Async/Await

- When the promis
  continues,
- if it was fulfilled th
  value,
- if it's rejected an
  synchronously which we can catch.

NodeMon

Automatic restart NodeJs

- Any changes done to code, we need to stop and start Node Server
- Nodemon is a package for handling this restart process automatically when changes occur in the project file.

Assignment – NodeMon

- Copy demo1 as demo2
- Npm install nodemon@1.18.10 -g
- Modify package.json to include dev-dependency for “nodemon”: “^1.18.10”
- Create nodemon.json
- Modify package.json start script to “nodemon src/index.ts”

MongoDB

What is MongoDB?

- The current SQL/NoSQL landscape
- Document-oriented vs. other types of storage
- Mongo's feature set
- Common use-cases
- Introduction to JSON

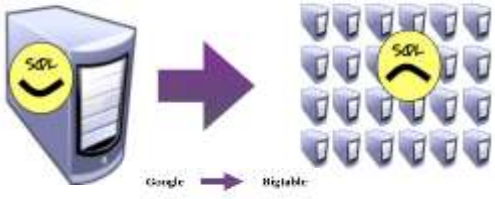
SQL has ruled for two decades

- Store persistent data
  Storing large amounts of data on disk, while allowing applications to grab the bits they need through queries
- Application integration
  Many applications in an enterprise need to share information. By getting all applications to use the database, we ensure all these applications have consistent, up-to-date data
- Mostly Standard
  The relational model is widely used and understood. Interaction with the database is done with SQL, which is a (mostly) standard language. This degree of standardization is enough to keep things familiar so people don't need to learn new things
- Concurrency Control
  Many users access the same information at the same time. Handling this concurrency is difficult to program, so databases provide transactions to help ensure consistent interaction.
- Reporting
  SQL's simple data model and standardization has made it a foundation for many reporting tools

SQL's dominance is cracking

Relational databases are designed to run on a single machine, so to scale, you need buy a bigger machine

But it's cheaper and more effective to scale horizontally by buying lots of machines



Google → Bigtable

Amazon → Dynamo

NoSQL Databases

- There is no standard definition of what NoSQL means.
- The term began with a workshop organized in 2009, but there is much argument about what databases can truly be called NoSQL.

NoSQL Databases

- While there is no formal definition, there are some common characteristics
  - they don't use the relational data model, and thus don't use the SQL language
  - they tend to be designed to run on a cluster
  - they tend to be Open Source
  - they don't have a fixed schema, allowing you to store any data in any record

NoSQL Databases

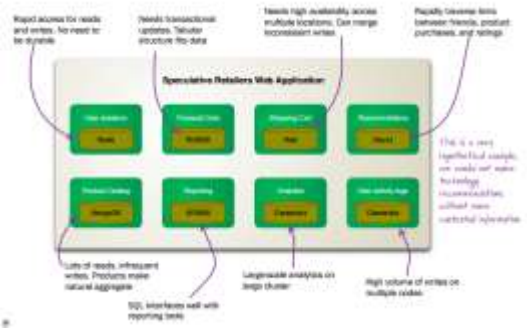
NoSQL Databases

- Main Advantages
  - Reduce Development Drag
  - Embrace Large Scale
- This does not mean Relational is dead
  - the relational model is still relevant
  - ACID transactions
  - Tools
  - Familiarity

Polyglot Persistence

- Using multiple data storage technologies, chosen based upon the way data is being used by individual applications. Why store binary images in relational database, when there are better storage systems?

What might Polyglot Persistence look like?

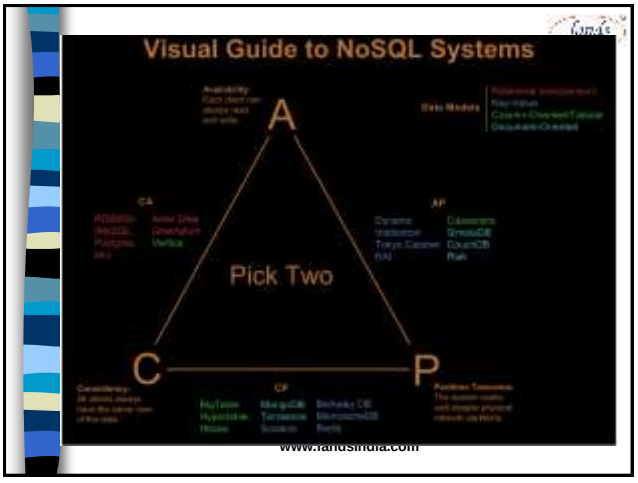


http://db-engines.com/en/ranking

Rank	DBMS	Database Model	Score
1.	Oracle	Relational, Multi-model	1340.28
2.	MySQL	Relational, Multi-model	1268.51
3.	Microsoft SQL Server	Relational, Multi-model	1059.72
4.	PostgreSQL	Relational, Multi-model	527.00
5.	MongoDB	Document, Multi-model	443.48
6.	IBM Db2	Relational, Multi-model	163.17
7.	Elasticsearch	Search engine, Multi-model	151.99
8.	Redis	Key-value, Multi-model	150.05
9.	SQLite	Relational	127.45
10.	Cassandra	Wide column	121.09

CAP theorem(wikipedia.org)

- In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:
  - Consistency (all nodes see the same data at the same time)
  - Availability (a guarantee that every request receives a response about whether it succeeded or failed)
  - Partition tolerance (the system continues to operate despite arbitrary partitioning due to network failures)



Insert

The following operation inserts a new document into the users collection. The new document has four fields name, age, and status, and an _id field. MongoDB always adds the _id field to the new document if that field does not exist.

```
db.users.insert (          <-- collection
   {
     name: "sue",          <-- field: value
     age: 26,              <-- field: value
     status: "A"           <-- field: value
   }                                          } document
)
```

The following diagram shows the same query in SQL:

```
INSERT INTO users                <-- table
            ( name, age, status ) <-- columns
VALUES      ( "sue", 26, "A" )    <-- values/row
```


Update

db.collection.update() method modifies existing documents in a collection. Can accept query criteria to determine which documents to update as well as an options document that affects its behavior, such as the multi option to update multiple documents.

save() method replaces the existing document with new doc.

```
db.users.update(                  <-- collection
   { age: { $gt: 18 } },          <-- update criteria
   { $set: { status: "A" } },     <-- update action
   { multi: true }                <-- update option
)
```

The following diagram shows the same query in SQL:

```
UPDATE users          <-- table
SET    status = 'A'   <-- update action
WHERE  age > 18       <-- update criteria
```

Remove

db.collection.remove() method deletes documents from a collection. The db.collection.remove() method accepts a query criteria to determine which documents to remove

```
db.users.remove(      <-- collection
   { status: "D" }    <-- remove criteria
)
```

The following diagram shows the same query in SQL:

```
DELETE FROM users     <-- table
WHERE  status = 'D'   <-- delete criteria
```

Find

Read operations, or queries, retrieve data stored in the database. In MongoDB, queries select documents from a single collection.

```
db.users.find(                          <-- collection
    { age: { $gt: 18 } },               <-- query criteria
    { name: 1, address: 1 }             <-- projection
).limit(5)                              <-- cursor modifier
```

The next diagram shows the same query in SQL:

```
SELECT _id, name, address   <-- projection
FROM   users                <-- table
WHERE  age > 18             <-- select criteria
LIMIT  5                    <-- cursor modifier
```

www.fandsindia.com

Assignment – Connecting to MongoDB

- Open Robo3T
  - Direct Connection
    - Address - xpanxioncluster-shard-00-01.m7uzk.mongodb.net
    - Port – 27017
  - Authentication
    - Database – admin
    - Username – testuser
    - Password – testuser
  - SSL
    - Use SSL Protocol
    - Self Signed Certificate

Node→MongoDB

- Modules
  - Mongoskin
    - the promise wrapper for node-mongodb-native.
  - Mongooes
    - MongoDB object modeling designed to work in an asynchronous environment

Required Vocabulary

- Database Schema: is close to the implementation details and tells the database (and developer) how an entity (e.g. user entity) looks like in a database table whereas every instance of an entity is represented by a table row. Basically a schema is the blueprint for an entity.
- Database Model: is a more abstract perspective on the schema. It offers the developer a conceptual framework on what models are available and how to use models as interfaces to connect an application to a database to interact with the entities. Often models are implemented with ORMs.
- Database Entity: is actual instance of a stored item in the database that is created with a database schema. Each database entity uses a row in the database table whereas each field of the entity is defined by a column.

Assignment..

- npm install mongoose –save
- Create EmpSchema

```
const empSchema = new mongoose.Schema(
  {
    empno: {
      type: Number,
      unique: true,
      required: true,
    },
    ename: {
        type: String,
        unique: false,
        required: true,
      },
    salary: {
        type: Number,
        unique: false,
        required: false,
      },
  },
  { timestamps: true },
);
```

Assignment..

- Create EmpModel

  const EmpModel = mongoose.model('myusers', empSchema);
- Open Connection

  let url =“mongodb+srv://testuser:testuser@host/mydb”

  mongoose.connect(url);
- Insert a document

  var empobj = new EmpModel({"empno": 10, "ename":"aa", "salary": 100});

  empobj.save();

Assignment

- List all documents

```
await EmpModel.find(
        (err, docarr)=>
        {if (err)
            console.log("Problem");
          else{
            console.log(docarr);
            res.send(docarr);}
          });

          mongoose.disconnect()
```

CRUD Mapping

Mongoose Queries

- Model.deleteMany()
- Model.deleteOne()
- Model.find()
- Model.findById()
- Model.findByIdAndDelete()
- Model.findByIdAndRemove()
- Model.findByIdAndUpdate()
- Model.findOne()
- Model.findOneAndDelete()
- Model.findOneAndRemove()
- Model.findOneAndReplace()
- Model.findOneAndUpdate()
- Model.replaceOne()
- Model.updateMany()
- Model.updateOne()

Query

- Query Creation

```
const query = MyModel.find();
// `query` is an instance of `Query`
query.collection(MyModel.collection);
query.where('age').gte(21).exec(callback);
```

- Operators

```
query.$where('this.comments.length === 10 ||
this.name.length === 5')
```

Assignment

- CRUD API development for UserModel
  - Get /users – all users
  - Get /users/:id – single user or 404
  - Post /users – insert user
  - Put /users – modify user based on key
  - Delete /users – delete all users
  - Delete /users/:id – delete user with key

QUESTION / ANSWERS

www.fandsindia.com

THANKING YOU !

www.fandsindia.com