Backend

Sridhar Alagar

Node.js

- Node.js is a runtime environment that allows you to run JavaScript on the server side
- It runs on Google's V8 JavaScript engine, outside of browser.
  - Makes it performant
- Node.js app runs in a single thread process
- Node.js uses event-driven, non-blocking I/O to handle many connections simultaneously
  - Ideal for scalable applications like chat apps, APIs, etc.

Node.js – other features

- Several built-in APIs with wide range of functionalities
  - File system, HTTP, Timers, Stream, Process, etc
- Node Package Manager (NPM)
  - Several millions of packages in the repository
- Server-side scripting

Express.js - A web framework for Node.js

- Fast, unopinionated, minimalist web framework
  - Relatively thin layer on top of the base Node.js functionality

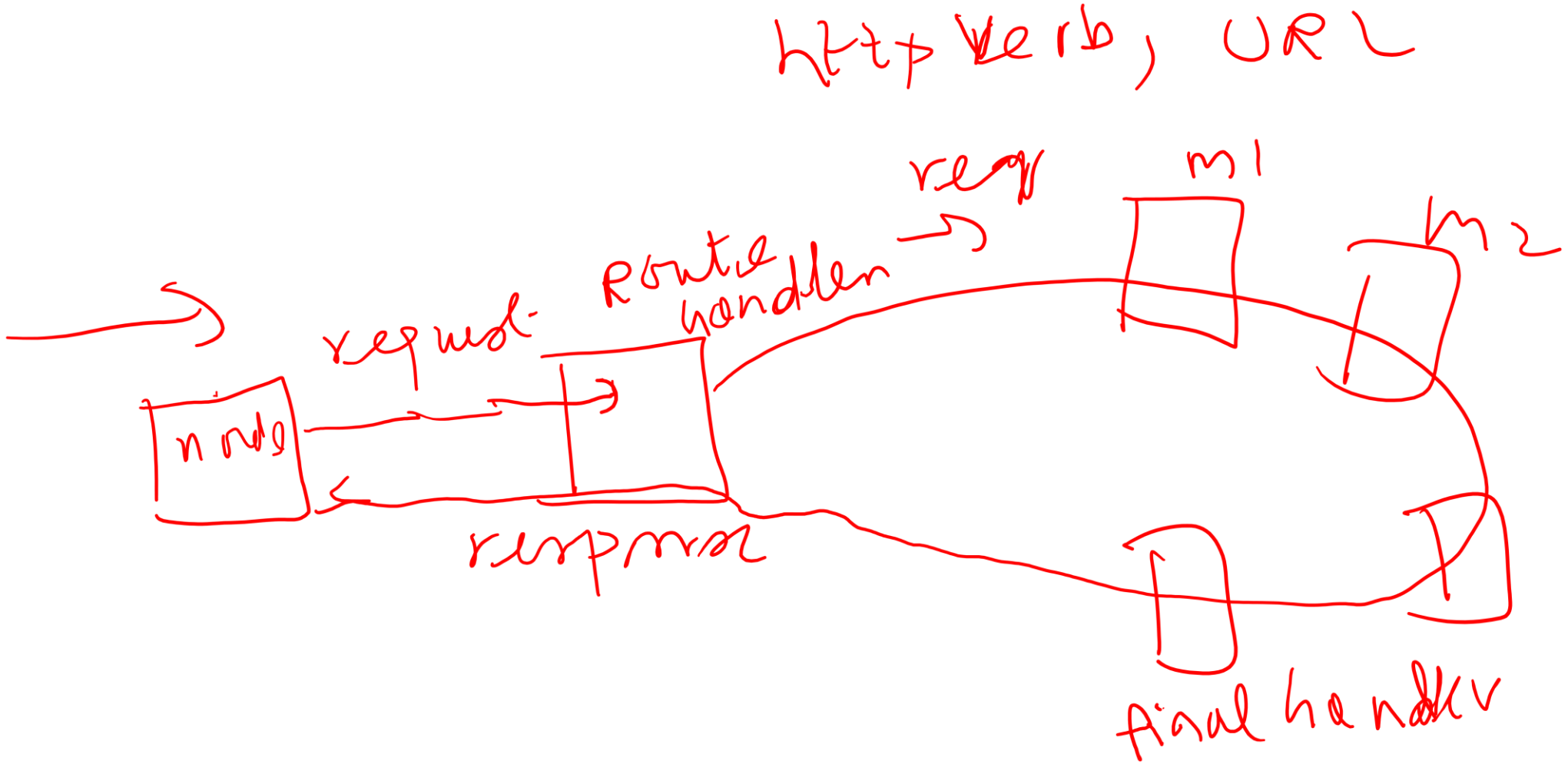
- Key benefits include
  - Simplified routing – mapping endpoints to webserver functions
  - Middleware support – allow request processing layers to added in
  - Integrate with view rendering engines – generate responses by inserting data into templates
  - Extensibility - A large ecosystem of plugins and tools via NPM

- Express is widely used for building scalable, maintainable web servers.

Simple express example – app.js

```
const express = require("express");
const app = express();
const port = 3000;

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

app.listen(port, function () {
  console.log(`Example app listening on port ${port}!`);
});
```

> node ./app.js


Http Verb, URL
req
m1
m2
Route handler
request
response
final handler

Creating Route handlers

```
// wiki.js - Wiki route module
const express = require("express");
const router = express.Router();

// Home page route
router.get("/", function (req, res) {
  res.send("Wiki home page");
});

// About page route
router.get("/about", function (req, res) {
  res.send("About this wiki");
});
module.exports = router;
```

```
const wiki =
require("./wiki.js");
// …
app.use("/wiki", wiki);
```

Using Middleware

```
const express = require("express");
const app = express();
// An example middleware function
const a_middleware_function = function (req, res, next) {
  // Perform some operations
  next(); // Call next() so Express will call the next middleware func in the chain.
};
// Function added with use() for all routes and verbs
app.use(a_middleware_function);
// Function added with use() for a specific route
app.use("/someroute", a_middleware_function);

// A middleware function added for a specific HTTP verb and route
app.get("/", a_middleware_function);
```

Using Thirdparty Middleware

```
const express = require("express");
const logger = require("morgan");
const app = express();
app.use(logger("dev"));
```

Serving static files

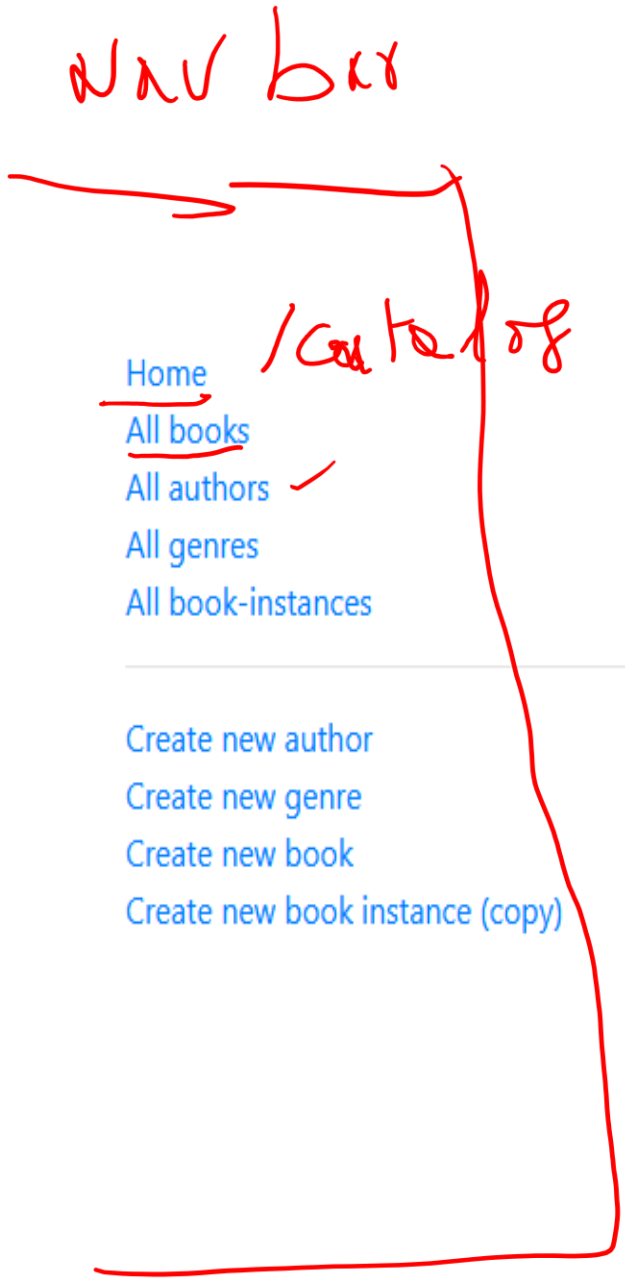
```
app.use(express.static("public"));
```

Any file in the /public folder is served

```
http://localhost:3000/images/dog.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/about.html
```

Rendering data using Template engines

```
const express = require("express");
const path = require("path");
const app = express();

// Set directory to contain the templates ('views')
app.set("views", path.join(__dirname, "views"));

// Set view engine to use, in this case 'some_template_engine_name'
app.set("view engine", "some_template_engine_name");
```



Local Library Home

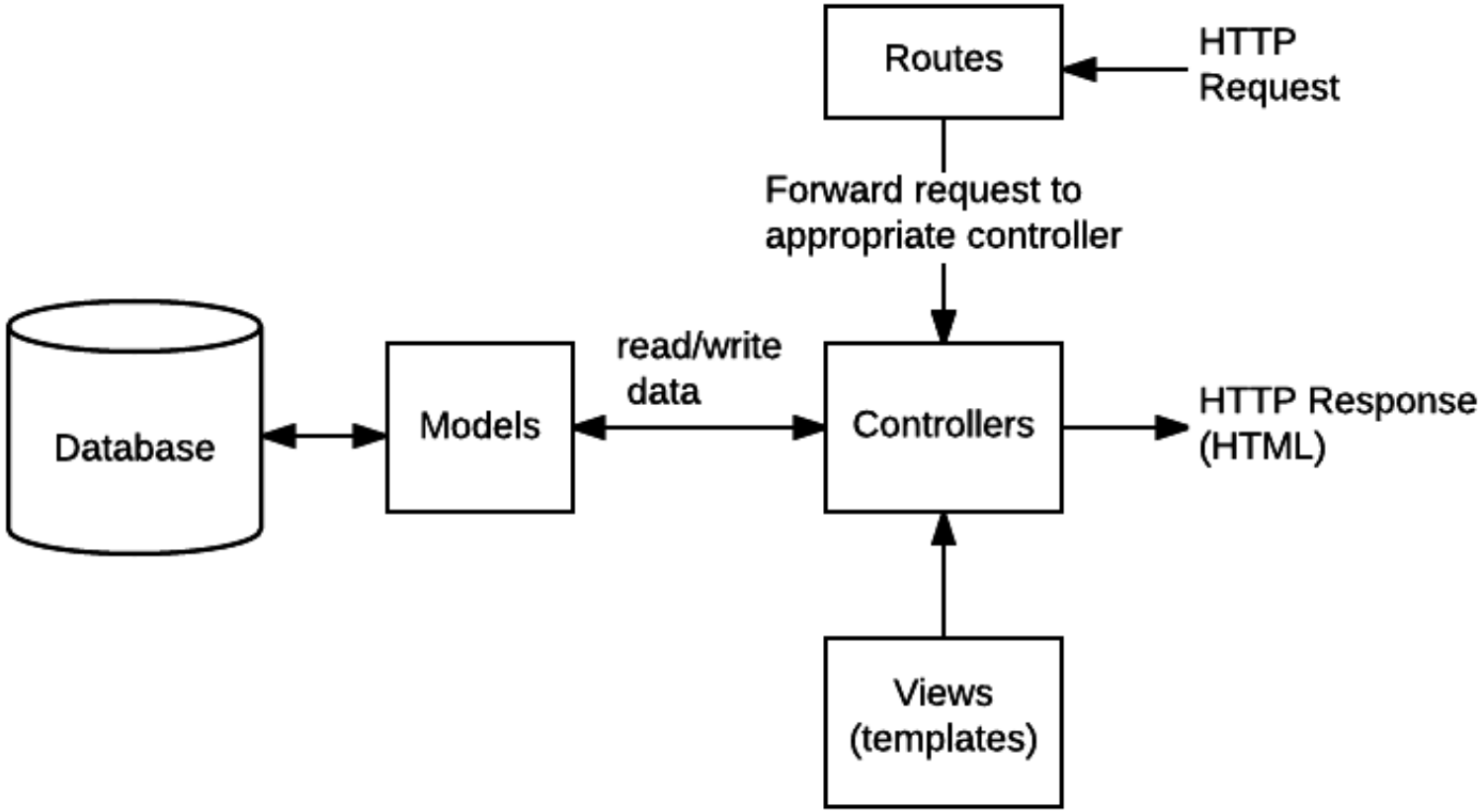
Welcome to *LocalLibrary*, a very basic Express website developed as a tutorial example on the Mozilla Developer Network.

Dynamic content

The library has the following record counts:

- **Books:** 7
- **Copies:** 11
- **Copies available:** 5
- **Authors:** 5
- **Genres:** 6

Data flow in our (MVC) model



Mongoose

- Mongoose is an ODM (Object Data Modeling) library for MongoDB designed to work in asynchronous environment (Node.js)

- It provides a structured way to interact with MongoDB, using schemas to define the shape of documents

- It offers data validation, type casting, middleware support, and more

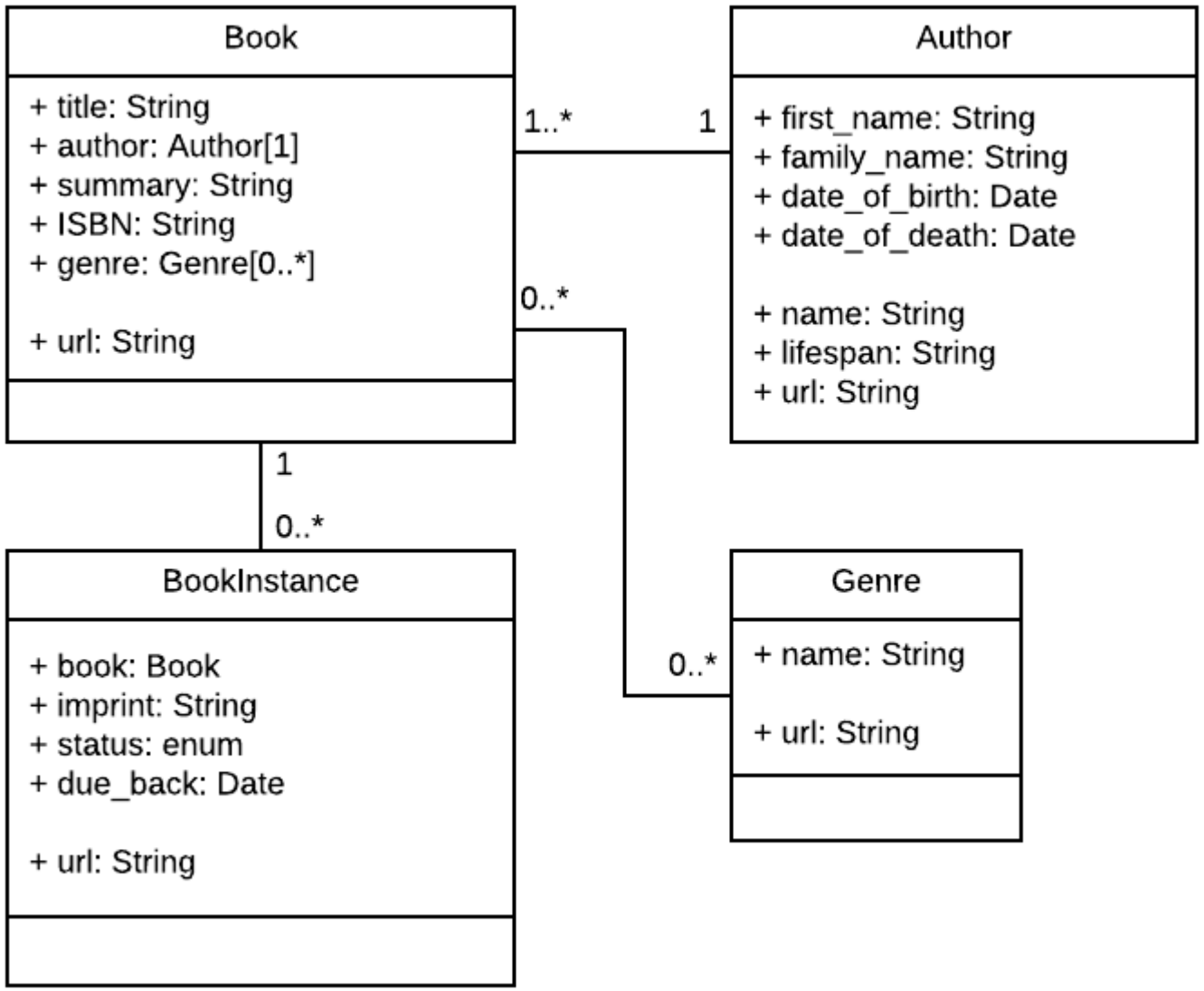
Mongoose

- Schema: Defines the structure of documents (fields, types, etc.).
- Model: Used to create and query documents. A constructor built from a schema.
- Documents: Analogous to rows/records in a relational database.

Simple example

```
const mongoose = require('mongoose’);

const Schema = mongoose.Schema;
const userSchema = new Schema({  name: String,  age: Number});

const User = mongoose.model('User', userSchema);

const user = new User({ name: 'John', age: 30 });

await user.save();
```

UML for our local library models



Genre model challenge

- The model should have a String SchemaType called name to describe the genre
- This name should be required and have between 3 and 100 characters.
- Declare a virtual for the genre's URL, named url.
- Export the model.

Handling exceptions in route functions

```
exports.get(
  "/users",
  async function (req, res, next) {
   try {
    const successfulResult = await Users.find({}).exec();
    res.render("users_view", { title: "Users", list: successfulResult });
   } catch (error) {
   return next(error);
   }
  }
);
```

express-async-handler to hide try/catch

```
// Import the module
const asyncHandler = require("express-async-handler");


exports.get(
  "/users",
  asyncHandler(async (req, res, next) => {
    const successfulResult = await Users.find({}).exec();
    res.render("users_view", { title: "Users", list: successfulResult });
  }),
);
```

Request object contains properties and methods to handle http requests

```
// For a URL like /search?term=node
req.query.term; // 'node'

// For a route like /user/:id
req.params.id; // The value of :id in the URL

// For a POST request with JSON body { "username": "john" }
req.body.username; // 'john'
// Available if middleware express.json is used

// contains cookies sent by the client
req.cookies.session_id; // Value of 'session_id' cookie
```

Request Object contains properties and methods to handle http requests

```
req.method; // 'GET', 'POST', etc.

req.url; // Full URL path, e.g., '/user/123?active=true'
req.path; // Just the path, e.g., '/user/123'

req.ip; // e.g., '127.0.0.1'

// holds session data
req.session.user; // Session data for the user
```

Response object contains properties and methods to handle http requests

```
res.send('Hello World!');

res.json({ message: 'Success' }); //sends a JSON response

res.status(404).send('Not Found'); // sets status code

res.redirect('/login'); // redirects to a different URL

res.render('profile', { user: req.user }); //renders a view template

res.end(); // ends the response process without sending any data
```

Response object contains properties and methods to handle http requests

```
res.set('Content-Type', 'text/html’);

res.cookie('name', 'value', { httpOnly: true });
res.clearCookie('name’);

// Sends different responses based on the client’s Accept header.
res.format({
  'text/plain': () => res.send('Plain text response'),
  'text/html': () => res.send('<p>HTML response</p>'),
  'application/json': () => res.json({ message: 'JSON response' })
});
```

Sources

1. MDN Web docs - Server side programming
2. CS142 Lectures