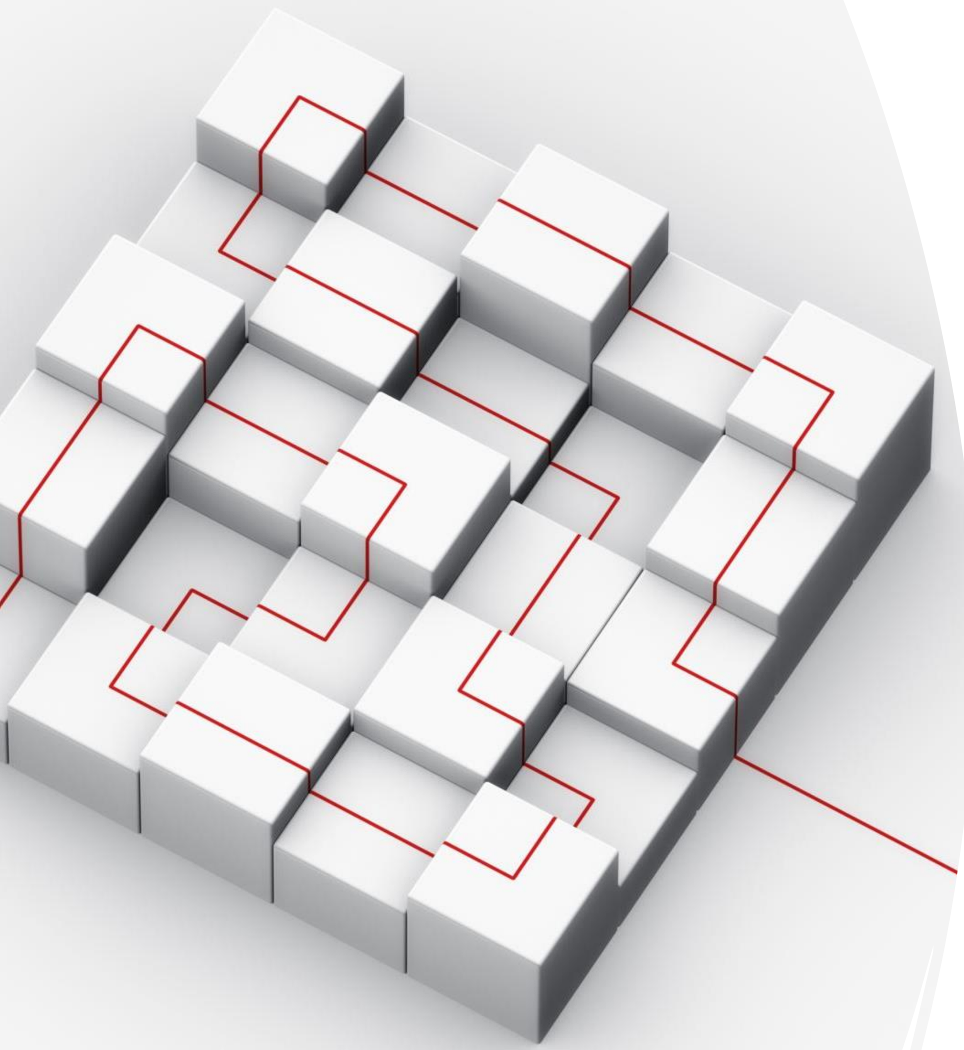
INT 161

Express JS

Basic Backend Development

Week-03

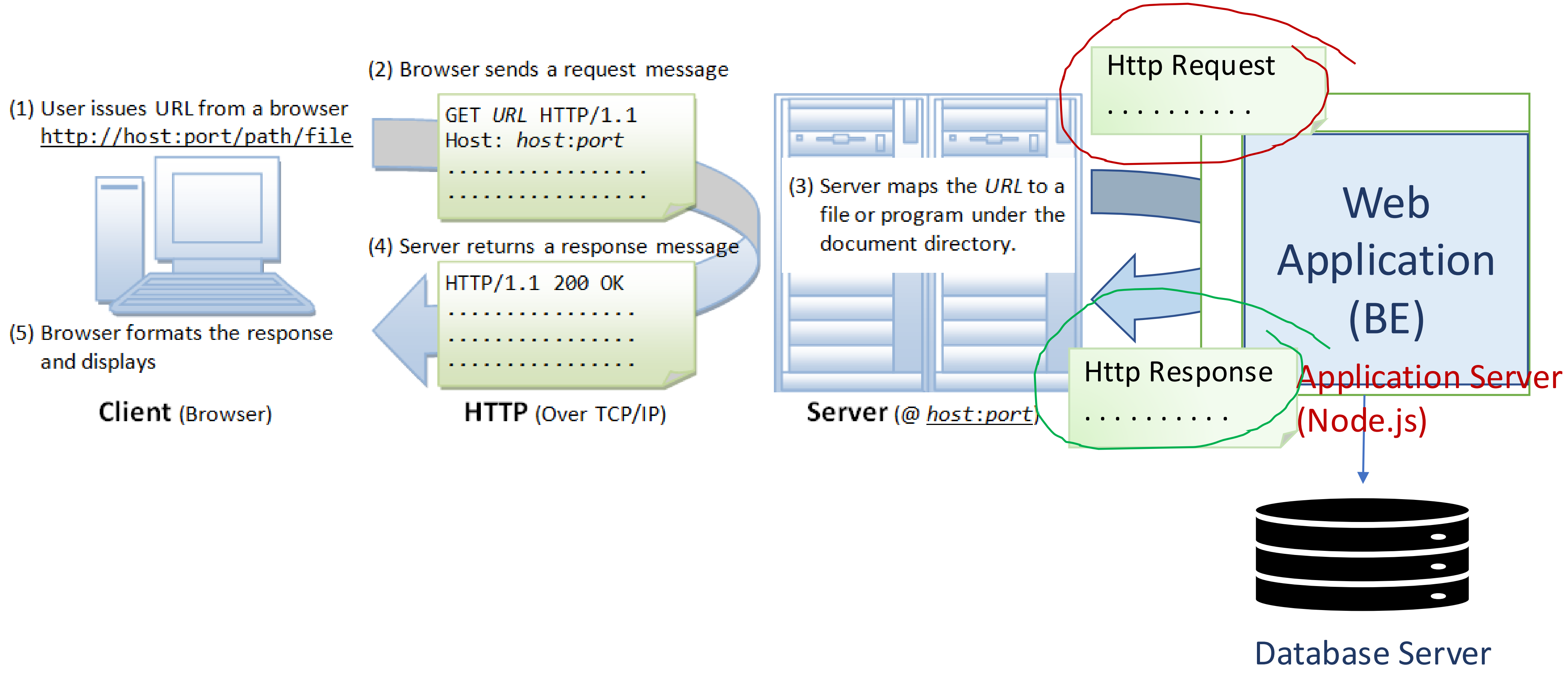
Express.js Framework



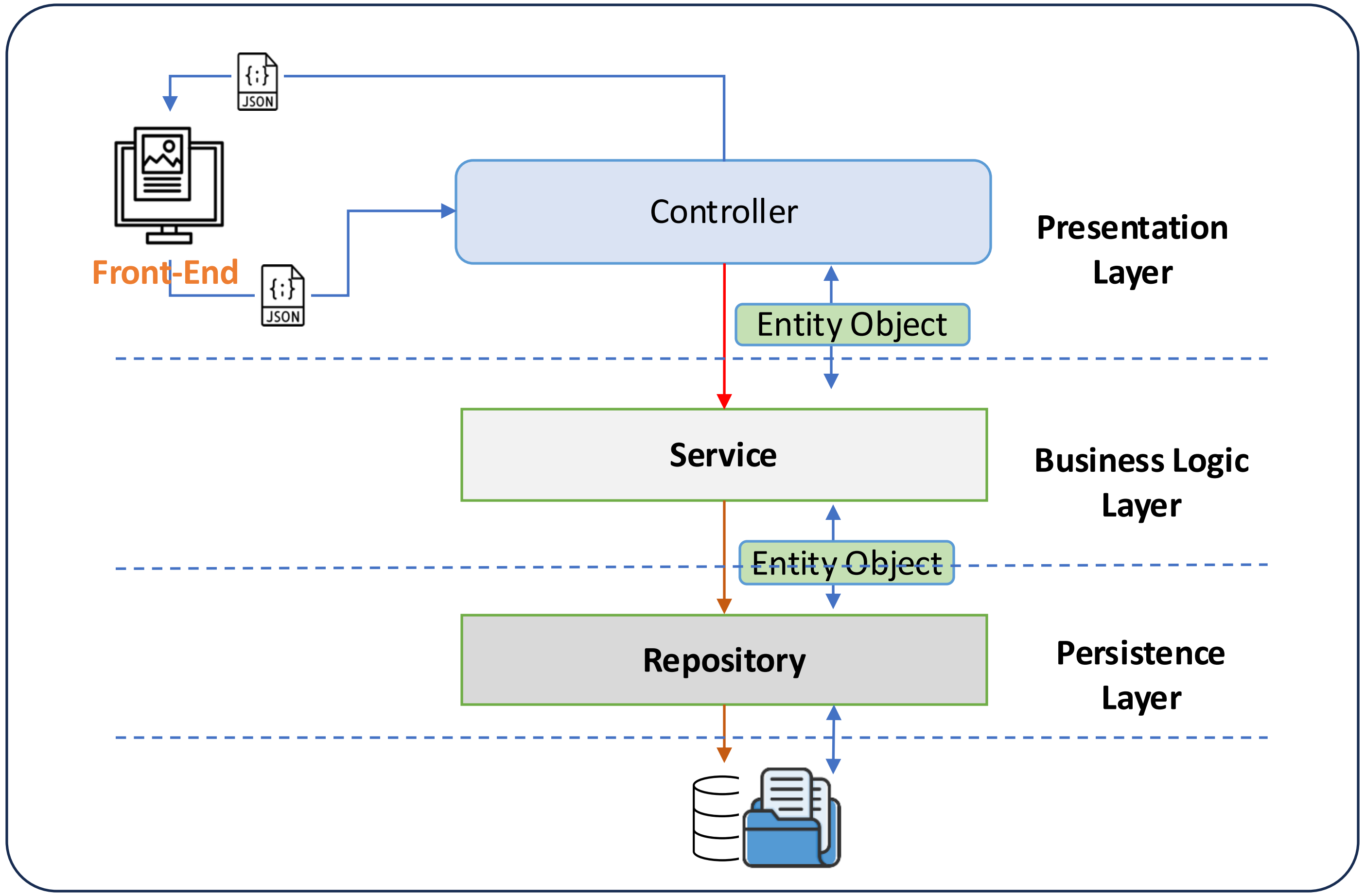
Unit Objectives

- After completing this unit, you should be able to:
  - Create Node.js Project with Express js framework
  - Explain basic concept of express framework
  - Explain the layer system of RESTful API principle
  - Create basic CRUD REST API follow the layer system

Web Application



Layered System



The **Layered System** principle means that a REST API is designed as a set of layers, where each layer has a specific role, and a client does not need to know whether it's communicating directly with the end server or through intermediaries.

This allows **scalability, flexibility, and separation of concerns**.

What is Express JS?

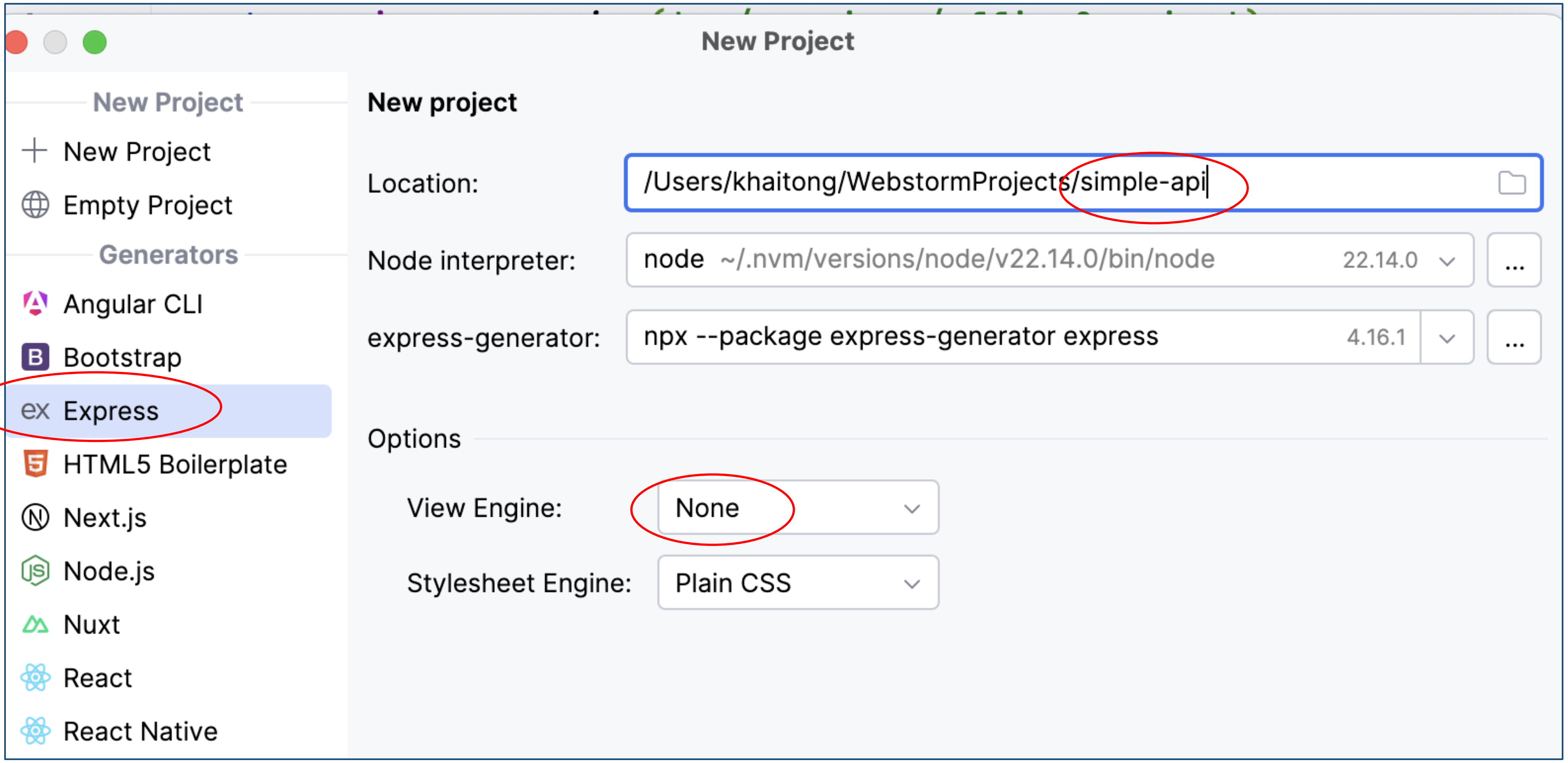
- Express.js is a Node js web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications.
- Express is the backend part of something known as the MEAN stack.
    - **M**ongoDB **E**xpress.js **A**ngular.js **N**ode.js
- The Express.js framework makes it very easy to develop an application which can be used tohandle multiple types of requests like the GET, PUT, and POST and DELETE requests.

Installation & Using Express

- **Install Express:** Run `npm install express` to create folder in node_modules and install the latest stable express to project .
- **Specify Version:** Add `@version` after the package name to install a specific Express version.
- **Import Express:** Use `var express = require('express');` to include Express in your code.

```
create : public/
create : public/javascripts/
create : public/images/
create : public/stylesheets/
create : public/stylesheets/style.css
create : routes/
create : routes/index.js
create : routes/users.js
create : public/index.html
create : app.js
create : package.json
create : bin/
create : bin/www
```

Create express project with WebStorm



New project

Location: /Users/khaitong/WebstormProjects/simple-api

Node interpreter: node ~/.nvm/versions/node/v22.14.0/bin/node 22.14.0

express-generator: npx --package express-generator express 4.16.1

Options

View Engine: None

Stylesheet Engine: Plain CSS

Key Parts of Express

- **Application (app)**
  - The app runs your Express project. It handles requests, sends responses, sets up routes, and adds middleware to process data or fix issues.
- **Request (req)**
  - The request holds details about what the user wants, like the URL, data sent, and extra info (headers). It helps your app know what to do.
- **Response (res)**
  - The response sends back web pages, data, or messages. You choose the format and set status codes to show success or errors.
- **Router (express.Router)**
  - The router organizes routes into groups, keeping code tidy and easier to manage, especially in bigger apps.

Routing in Express

- Routing decides how an app handles client requests to a specific URL and HTTP method (GET, POST, etc.).
- Syntax:

```
router.METHOD(PATH, HANDLER);
```

  - METHOD can be GET, POST, PUT, DELETE, etc.
- Example:

```
router.get('/home', (req, res) => {
    res.send('Welcome to the homepage!’);
});
```

- Route patterns can be strings or regex.
- Strings may include parameters.
- Example:

```
router.get('/api/user/:id', (req, res)
```

Understanding Request Objects

Request objects provide access to various components of HTTP requests, including:

- **req.params**: Extracts parameters embedded within the URL path.
- **req.query**: Retrieves parameters from the query string portion of the URL.
- **req.get()**: Fetches the value of a specified HTTP header.
- **req.body()**: Contains the parsed body of the request, typically used with middleware.
- **req.is()**: Checks the MIME type of the request body content.
- **req.url**: The URL path that matched the current route handler.
- **req.originalUrl**: The full original URL as received from the client.
- **req.protocol**: Indicates whether the request was made over HTTP or HTTPS.
- **req.secure**: Boolean flag that is true if the connection uses HTTPS.
- **req.host**: The value specified in the Host header of the request.
- **req.path**: The path portion of the URL.
- **req.xhr**: True if the request was initiated via an XMLHttpRequest (AJAX).

Handling Responses

There are multiple methods to craft responses to client requests:

- res.set() – Assign specific headers to the response.
- res.redirect() – Send a 301 or 302 redirect to a designated URL.
- res.send() – Deliver a status code along with a string, array, object, or buffer.
- **res.json() – Convert a JavaScript object or value into a JSON string for the response.**
- res.jsonp() – Wrap a JavaScript value in a callback function for JSONP responses.
- res.sendFile() – Stream the contents of a file directly to the client.
- res.download() – Stream a file with headers prompting the client to download it as an attachment.

Middleware in Express

- Middleware acts as an intermediary step that processes requests before they reach the router.
- For instance, it can verify user authentication prior to granting access to admin api.
- To pass control to the subsequent middleware function, use the next() method.
- Common categories of middleware include:
  - Application-level Middleware
  - Router-level Middleware
  - Error-handling Middleware
  - Built-in Middleware
  - Third-party Middleware

Modify user router routes/users.js & Test them with Postman (1/2)

```
router.get('/', function (req, res, next) {
   res.send(`url = ${req.originalUrl}, Response from ${req.method} method`);
});
router.get('/:id', function (req, res, next) {
   res.send(`url = ${req.originalUrl}, Response from ${req.method} method with id =
   ${req.params.id}`);
});
router.post('/', function (req, res, next) {
   respBody = (`url = ${req.originalUrl}, Response from ${req.method} method\n`);
   respBody += JSON.stringify(req.body, null, 2);
   res.send(respBody);
});
```

Modify user router routes/users.js & Test them with Postman (2/2)

```
router.put('/:id', function (req, res, next) {
    respBody = (`url = ${req.originalUrl}, Response from ${req.method} method\n`);
    respBody += JSON.stringify(req.body, null, 2);
    res.send(respBody);
});
router.delete('/:id', function (req, res, next) {
    res.send(`url = ${req.url}, Response from ${req.method} method with id =
    ${req.params.id}`);
});
```

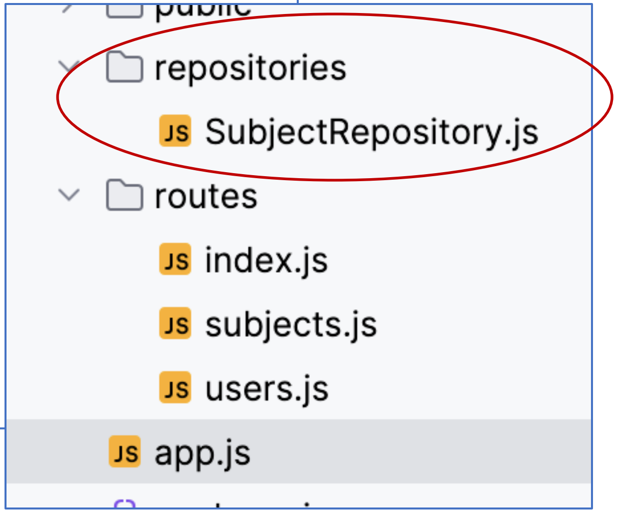
Create new router subjects.js & Copy SubjectRepository.js to project

```
var express = require('express');
var router = express.Router();
const repo = require('../repositories/SubjectRepository.js');

router.get('/', function (req, res, next) {
    res.json(repo.getSubjects());
});

router.get('/:id', function (req, res, next) {
    res.json(repo.getSubject(req.params.id));
});

module.exports = router;
```



Add router to app.js

```
     :
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var subjectsRouter = require('./routes/subjects');
     :
app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/subjects', subjectsRouter);
     :
```

Modify SubjectRepository.js

```
function addSubject(subject) {
    subjects.push(subject);
    return getSubject(subject.id);
}

function updateSubject(id, subject) {
    const idx = subjects.findIndex(s => s.id === id);
    if (idx === -1) return null;
    subject.id = id;
    subjects[idx] = subject;
    return subject;
}
```

Modify router subjects.js

```
router.put('/:id', function (req, res, next) {
    newSubject = repo.updateSubject(req.params.id, req.body);
    if (newSubject != null) {
        res.json(newSubject);
    } else {
        res.status(404).send('Subject ID '+ req.params.id+ ' not found');
    }
});

router.delete('/:id', function(req, res, next) {
    if (repo.removeSubject(req.params.id)) {
        res.status(204).send();
    } else {
        res.status(404).send('Subject ID '+ req.params.id+ ' not found');
    }
});
```

```
router.post('/', function (req, res, next) {
    newSubject = req.body;
    res.json(repo.addSubject(newSubject));
});
```

Layer System Functions (1/2)

- Controller (API Layer)
  - Acts as the entry point for HTTP requests (GET, POST, PUT, DELETE).
  - Converts incoming requests → objects that the service layer can process.
  - Sends back HTTP responses (status codes + JSON body).
  - Does not contain business logic, only delegates.
  - Analogy: Reception desk at a hospital — takes the patient info and directs them to the right doctor.
- Service Layer (Business Logic Layer)
  - Contains the core business logic of the application.
  - Applies rules, validations, calculations.
  - Coordinates between Controller and Repository.
  - Can call multiple repositories or external APIs to fulfill business processes.
  - Analogy: The doctor at the hospital — examines, diagnoses, and decides treatment.

Layer System Functions (2/2)

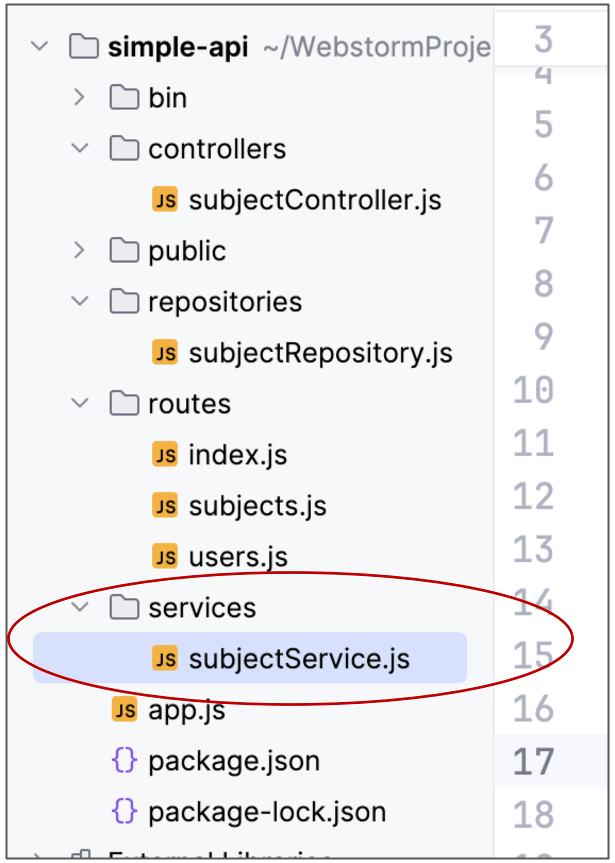
- Repository Layer (Data Access Layer)
  - Responsible for data persistence and retrieval.
  - Provides an abstraction over the database (CRUD operations).
  - No business logic — only raw data access.
  - Analogy: The hospital records office — stores and retrieves patient files.

Layer	**Responsibility**	**Should Contain**	**Should Not Contain**
Controller	Handle HTTP requests/responses	Routing, mapping, validation	Business logic, DB code
Service	Business rules & process coordination	Business logic, calculations	HTTP or DB details
Repository	Data access (CRUD)	Queries, persistence	Business logic, HTTP

Create Subjects service, subjectService.js (1/2)

```
const repo  = require('../repositories/subjectRepository');

module.exports = {
   getAllSubjects: function() {
      return repo.getSubjects();
   },
   findById: function(id) {
      subject = repo.getSubject(id);
      if (!subject) {
         throw new Error(`Subject not found for ID ${id}`);
      }
      return subject;
   },
```



```
simple-api ~/WebstormProje
  bin
  controllers
    subjectController.js
  public
  repositories
    subjectRepository.js
  routes
    index.js
    subjects.js
    users.js
  services
    subjectService.js
  app.js
  package.json
  package-lock.json
```

Create Subjects service, subjectService.js (2/2)

```
addSubject: function(newSubject) {
        if (newSubject.id===undefined || newSubject===null || newSubject.id==="") {
            throw new Error("Bad Request: missing id");;
        }
        if (repo.getSubject(newSubject.id)) {
            throw new Error(`Bad Request: Duplicate id ${newSubject.id}`);
        }
        return repo.addSubject(newSubject);
    },
    updateSubject: function(id, subject) {},
    removeSubject: function(id) {}
}
```

Create Controller, subjectController.js (1/2)

```
var service = require('../services/subjectService');

function error(req, error, message, statusCode) {
    return {
        error: error,
        statusCode: statusCode,
        message: message,
        path: req.originalUrl,
        timestamp: new Date().toLocaleString()
    };
}
module.exports = {
    list: function (req, res) {
        res.json(service.getAllSubjects());
    },
```

Modify Controller, subjectController.js (2/2)

```
 get: function (req, res) {
        if(req.params.id.trim() === "")
            return res.status(400).json(error(req,"Bad Request",
                "Bad Request: empty id", 400));
        res.json(service.findById(req.params.id));
    },
    create: function (req, res) {
        newSubject = req.body;
        if (Object.keys(newSubject).length===0)
            return res.status(400).json("Bad Request: empty body");
        res.json(service.addSubject(newSubject));
    },
    update: function (req, res) {
    },
    remove: function (req, res) {
    }
}
```