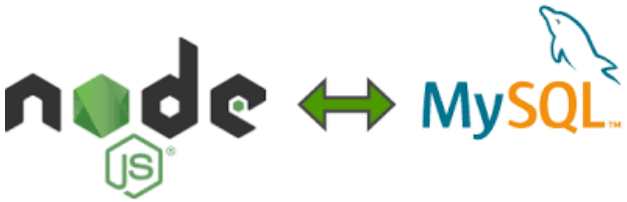
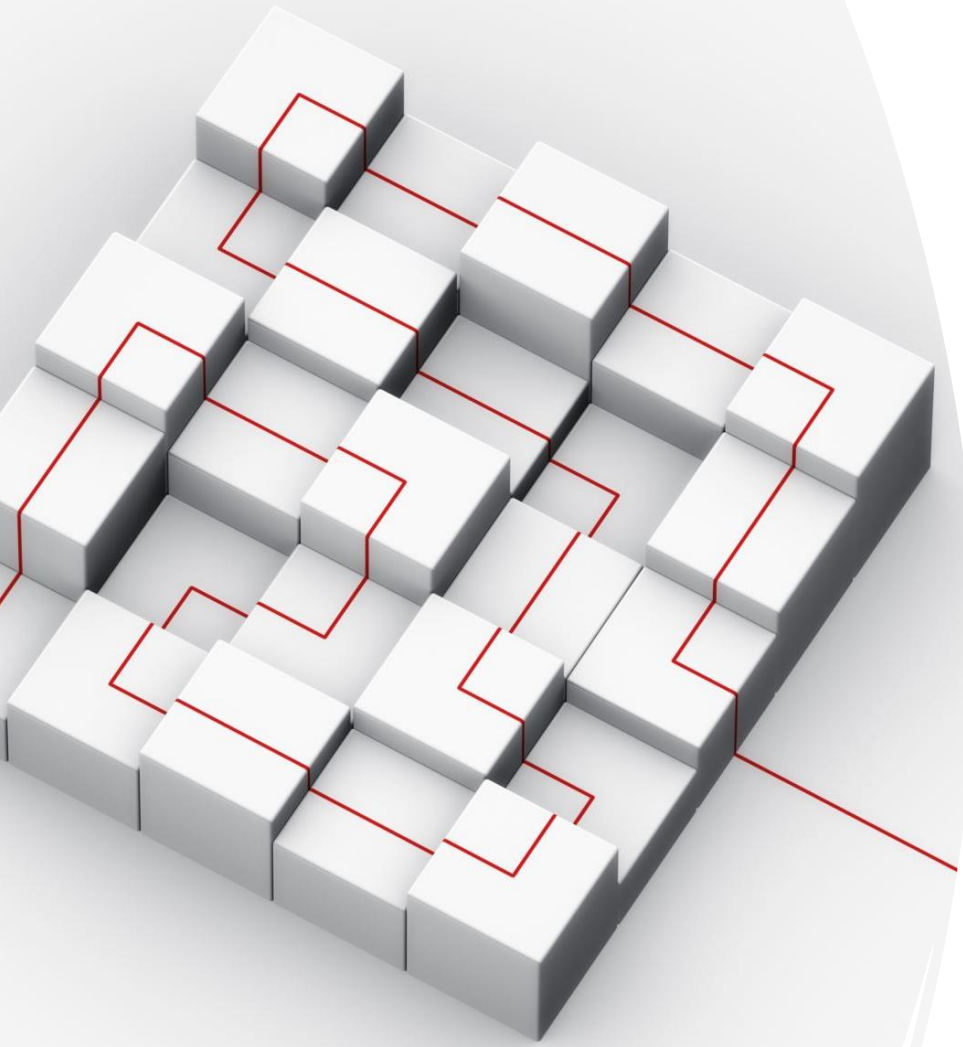
INT 161



Basic Backend Development

Week-04

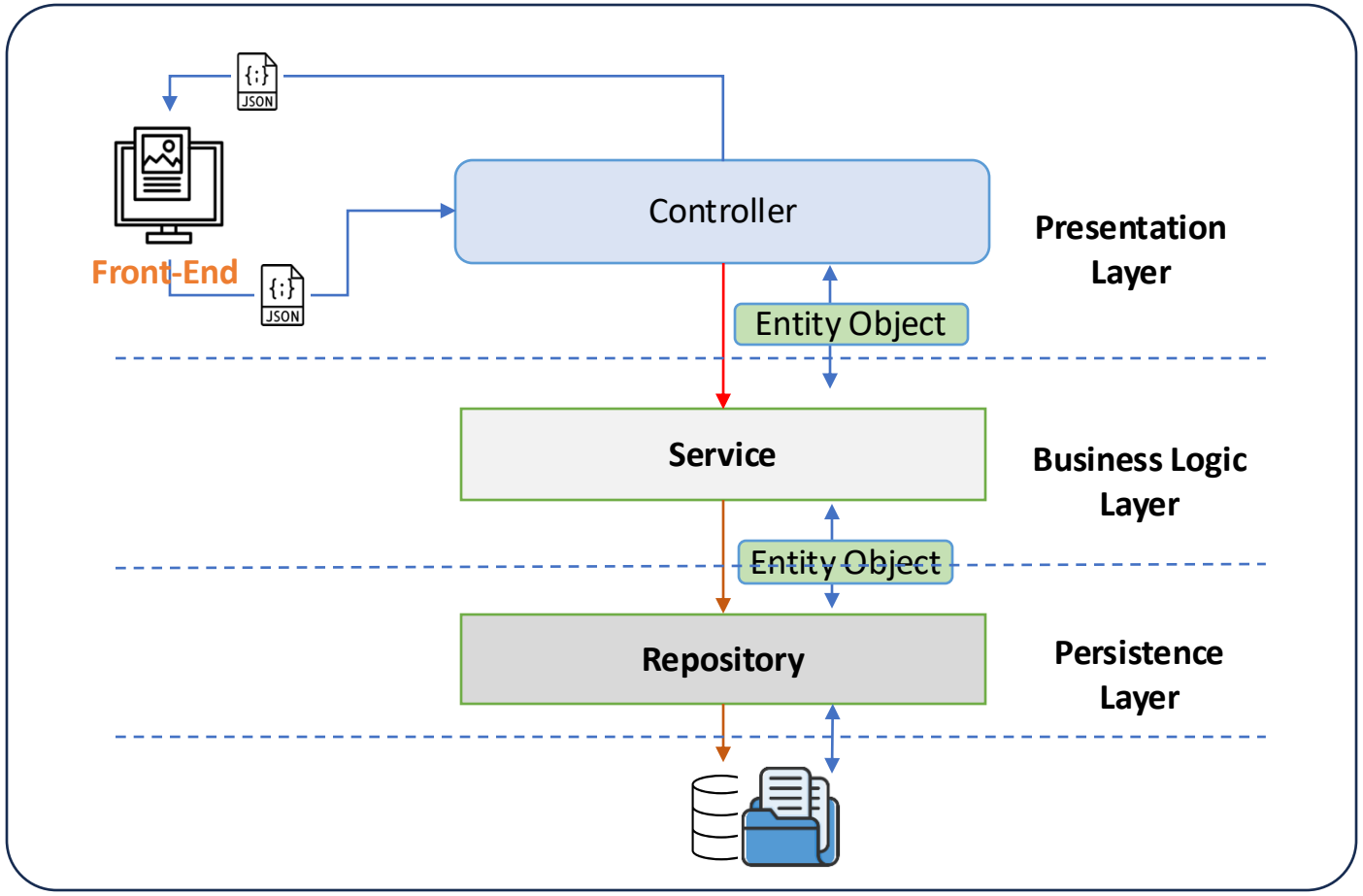
`Database Connection`



Unit Objectives

- After completing this unit, you should be able to:
  - Using Express.js to connecting with MySQL
  - Create CRUD API
  - Using Prepared Statement to prevent SQL Injection
  - Understanding layer system code management

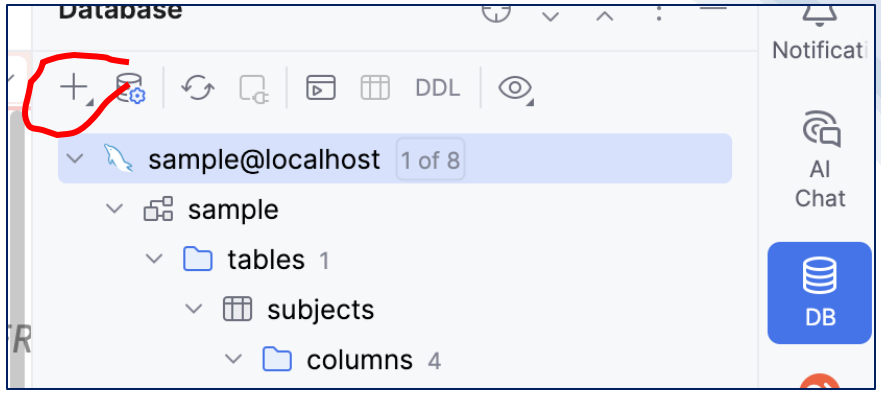
Layered System

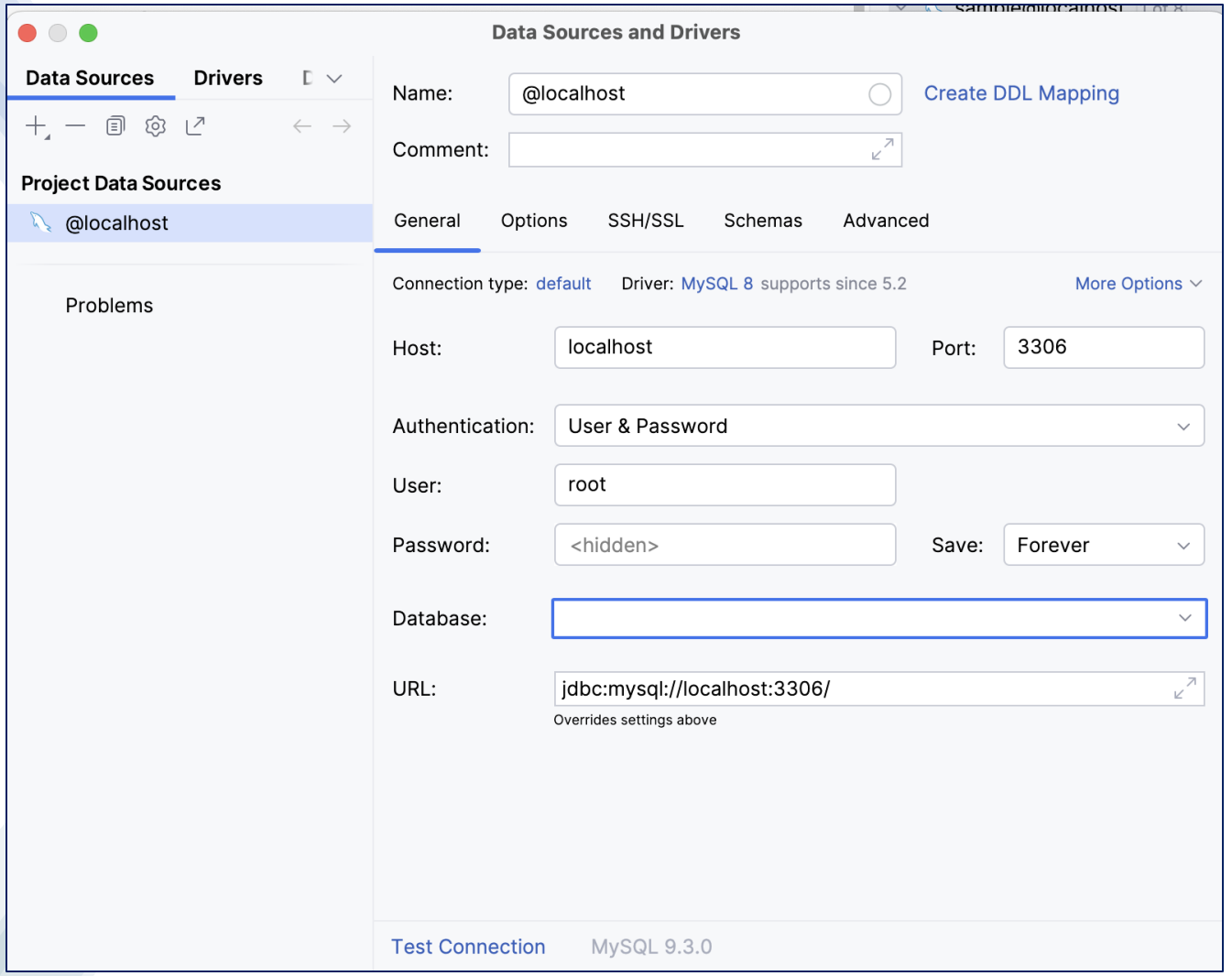


The **Layered System** principle means that a REST API is designed as a set of layers, where each layer has a specific role, and a client does not need to know whether it's communicating directly with the end server or through intermediaries.

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

Using DB Client Tool in WebStorm





Create Database and Table

```
use sample;
create table subjects
(
    id int primary key auto_increment,
    subject_code  varchar(8) not null unique,
    subject_title varchar(60),
    credit int  not null
);
```

```
insert into subjects (subject_code, subject_title, credit)
values ('INT 100', 'IT Fundamentals', 3),
       ('INT 101', 'Programming Fundamentals', 3),
       ('INT 102', 'Web Technology', 1),
       ('INT 114', 'Discrete Mathematics', 3),
       ('GEN 101', 'Physical Education', 1),
       ('GEN 111', 'Man and Ethics of Living', 3),
       ('LNG 120', 'General English', 3),
       ('LNG 220', 'Academic English', 3),
       ('INT 103', 'Advanced Programming', 3),
       ('INT 104', 'User Experience Design', 3),
       ('INT 105', 'Basic SQL', 1),
       ('INT 107', 'Computing Platforms Technology', 3),
       ('INT 200', 'Data Structures and Algorithms', 1),
       ('INT 201', 'Client-Side Programming I', 2),
       ('INT 202', 'Server-Side Programming I', 2),
       ('INT 205', 'Database Management System', 3),
       ('INT 207', 'Network I', 3);
```

Mysql2 Library > `npm install mysql2`

- Asynchronous (non-blocking I/O) using event loop
- Pure Node.js + native bindings using MySQL protocol
- Connection
  - Instance for connecting to DBMS
- Pool
  - Connection pool manager
- Query
  - Use to send SQL statement to DBMS
- Result
  - Array [rows, fields]
- Using
  - callback API (async style)
  - Promise API (async/await)

Mysql2 Library

App → mysql2 Pool/Connection → Query (async via libuv) → DBMS

- Asynchronous (non-blocking I/O) using event loop
- Pure Node.js + native bindings using MySQL protocol
- Connection
  - Instance for connecting to DBMS
- Pool
  - Connection pool manager
- Query
  - Use to send SQL statement to DBMS
- Result
  - Array [rows, fields]
- Using
  - callback API (async style)
  - Promise API (async/await)

Creating Connection Pool

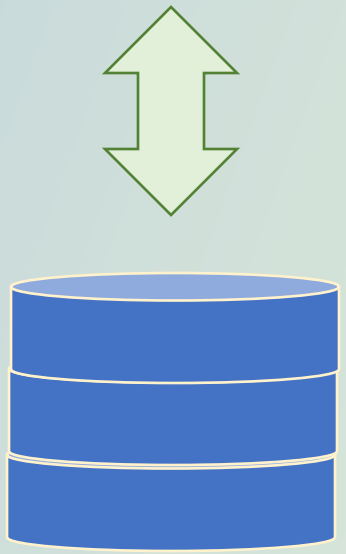
```
const mysql = require('mysql2/promise');
const pool = mysql.createPool({
    host: 'localhost',
    port: 3306,
    user: 'root',
    password: '143900',
    database: 'sample',
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0,
});
module.exports = pool;
```

- createConnection() provides a direct, one-to-one relationship with the database for single operations.
- createPool() offers a more robust and scalable solution by managing a collection of reusable connections, optimizing performance and resource utilization in demanding applications.

Send SQL to DBMS (Call Back vs Asyn/Await)

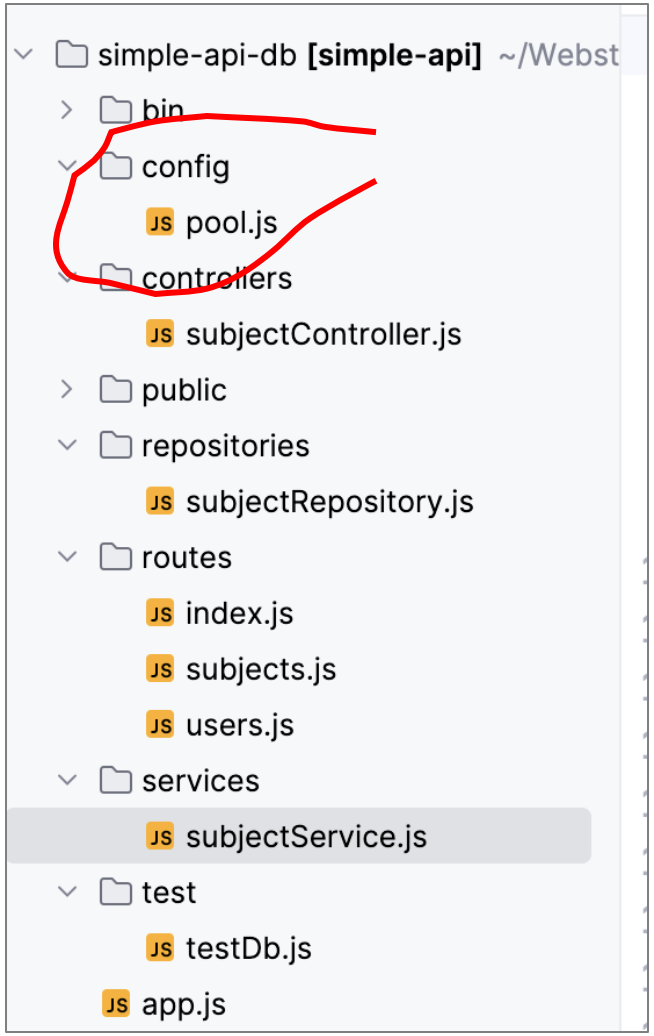
```
const pool = require('./db/pool');

pool.query (
  'SELECT * FROM students WHERE age > ?', [18], (err, results, fields) => {
    if (err) {
         console.error(' Query error:', err);
         return;
    }
    console.log('Results:', results);
  }
);
-------------------------------------------------------------------------

async function main() {
  const [rows] = await pool.query('SELECT * FROM students WHERE age > ?', [18]);
  console.log(rows[0].message);
}

main();
```

Test CRUD with Connection Pool



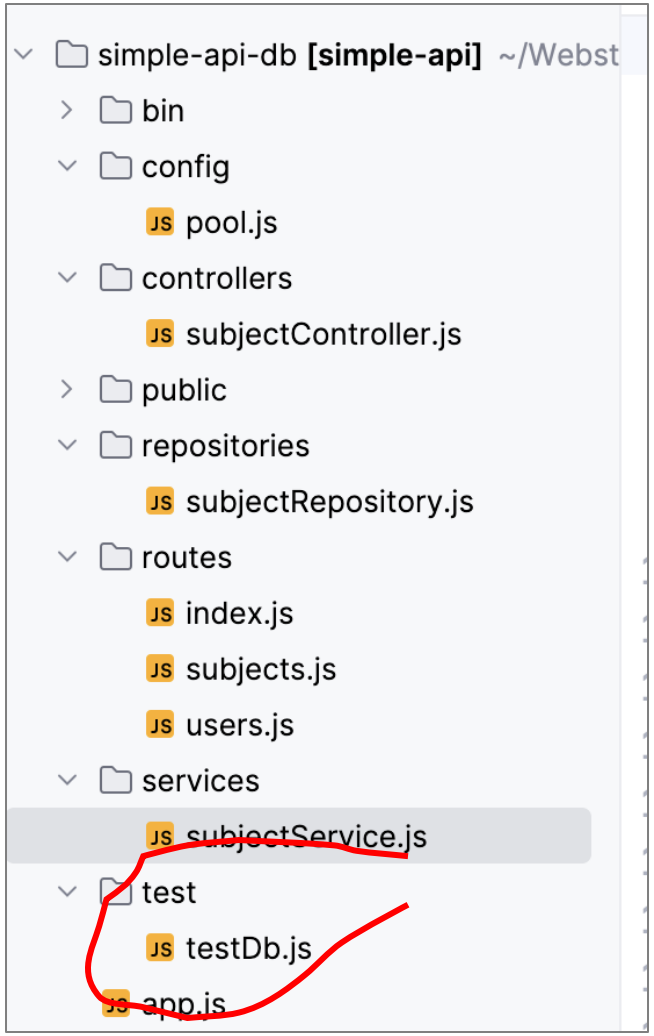
Create pool.js in folder ./config

```
const mysql = require('mysql2/promise');
const pool = mysql.createPool({
    host: 'localhost',
    port: 3306,
    user: 'root',
    password: ‘mysql@sit',
    database: 'sample',
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0,
});
module.exports = pool;
```



Create testDb in folder ./test, then Run this file

```
const conn = require('../config/pool');

// Use async/await with the promise-based pool
async function getAllSubjects() {
   const [result] = await conn.query('SELECT * FROM subjects');
   console.log(result);
}

(async () => {
   try {
      await getAllSubjects();
   } catch (err) {
      console.error(err);
   } finally {
      // If this is a standalone test script, close the pool
      await conn.end();
   }
})();
```



Add function getSubjectById to testDb, Run again

```
async function getSubjectById(id) {
   console.log('Search for id:' + id);
   console.log('----------------------');
   const oneRow = await conn.query('SELECT * FROM subjects where id = ?',[id]);
   console.log(oneRow);
   console.log('----------------------');
   console.log(oneRow[0][0]);
}

(async () => {
   try {
      // await getAllSubjects();
      await getSubjectById(1);
      await getSubjectById(99);

  }
   :
   :
```

Add function addSubject to testDb, Run again

```
async function addSubject(subject) {
   //preparing data to be inserted
   const added = await conn.query(
      "INSERT INTO subjects (subject_code, subject_title, credit) " +
      "VALUES (?, ?, ?)", [subject.code, subject.title, subject.credit]
   );
   if (added[0].insertId > 0) {
      console.log("Added successfully");
      console.log(added);
   }
}
```

```
(async () => {
   try {
      await addSubject(
         {
            code: 'INT-959',
            title: 'SQL for Data Science',
            credit: 3
         }
      );
      // await getAllSubjects();
      // await getSubjectById(1);
      // await getSubjectById(99);
   } catch (err) {
      console.error(err);
```

Add function updateSubject to testDb, Run again

```
async function updateSubject(id, subject) {
   const updated = await conn.query(
      "UPDATE subjects SET subject_code = ?, subject_title = ?, "
   + " credit = ? WHERE id = ?",
      [subject.code, subject.title, subject.credit, id]);

   console.log(updated);
   if (updated.affectedRows > 0)
      console.log("Updated successfully");
   else
      console.log("No record updated");
}
```

```
(async () => {
   try {
      await updateSubject(19,
         {
            code: 'INT-900',
            title: 'SQL for Data Science',
            credit: 3
         }
      );
      // await getAllSubjects();
      // await getSubjectById(1);
      // await getSubjectById(99);
   } catch (err) {
      console.error(err);
```

DIY

Add function removeSubject to testDb, Run again

Create REST API CRUD with Database

Modify subjectRepository.js (1/3)

```
const pool = require("../config/pool");

async function findAll() {
   const [rows] = await pool.query(
      "SELECT id, subject_code, subject_title, credit FROM subjects");
   return rows;
}

async function findById(id) {
   const [rows] = await pool.query(
      "SELECT id, subject_code, subject_title, credit FROM subjects WHERE id = ?", [id]);
   if (rows.length === 0) return null;
   subject  = {id : rows[0].id, code:rows[0].subject_code,
      title:rows[0].subject_title, credit:rows[0].credit};
   return subject;
}
```

Modify subjectRepository.js (2/3)

```
async function save(subject) {
  const [result] = await pool.query(
    "INSERT INTO subjects (subject_code, subject_title, credit) VALUES (?, ?, ?)",
    [subject.code, subject.title, subject.credit]
  );
  if (result.affectedRows === 0) return null;
  return findById(result.insertId);
}

async function update(subject) {
  const [result] = await pool.query(
    "UPDATE subjects SET subject_code = ?, subject_title = ?, credit = ? WHERE id = ?",
    [subject.code, subject.title, subject.credit, subject.id]
  );
  if (result.affectedRows === 0) return null;
  return findById(subject.id);
}
```

Modify subjectRepository.js (3/3)

```
async function deleteById(id) {
    const [result] = await pool.query("DELETE FROM subjects WHERE id = ?", [id]);
    return result.affectedRows > 0;
}

module.exports = {
    findAll,
    findById,
    save,
    update,
    deleteById
};
```

Modify subjectService.js (1/4)

```
const repo = require('../repositories/subjectRepository');
function validateSubjectBody(body) {
  if (!body || Object.keys(body).length === 0) {
    const err = new Error('Bad Request: empty body');
    err.status = 400;
    throw err;
  }
  const { code, title, credit } = body;
  if (!code || typeof code !== 'string') {
    const err = new Error('Bad Request: subject_code is required');
    err.status = 400;
    throw err;
  }
 if (!title || typeof title !== 'string') {
    const err = new Error('Bad Request: subject title is required');
    err.status = 400;
    throw err;
  }
```

```
 if (credit === undefined || credit ===
null || isNaN(Number(credit))) {
    const err = new Error('Bad Request:
credit must be a number');
    err.status = 400;
    throw err;
  }
}
```

Modify subjectService.js (2/4)

```
module.exports = {
   getAllSubjects: async function () {
      const subjects = await repo.findAll();
      return subjects;
   },
   getSubjectById: async function (id) {
      const subject = await repo.findById(id);
      if (!subject) {
         const err = new Error(`Subject not found for ID ${id}`);
         err.code = 'NOT_FOUND';
         err.status = 404;
         throw err;
      }
      return subject;
   },
```

Modify subjectService.js (3/4)

```
addSubject: async function (newSubject) {
   validateSubjectBody(newSubject);
   // Enforce unique subject_code handled by DB unique constraint; let error bubble up to controller
   const created = await repo.save(newSubject);
   return created;
},
updateSubject: async function (id, subject) {
   validateSubjectBody(subject);
   subject.id = id;
   const updated = await repo.update(subject);
   if (!updated) {
      const err = new Error(`Subject not found for ID ${id}`);
      err.status = 404;
      throw err;
   }
   return updated;
},
```

Modify subjectService.js (4/4)

```
 removeSubject: async function (id) {
      const removed = await repo.deleteById(id);
      if (!removed) {
         const err = new Error(`Subject not found for ID ${id}`);
         err.status = 404;
         throw err;
      }
      return removed;
   }
}
```

Modify subjectController.js (1/5)

```
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: async function (req, res) {
        try {
            const subjects = await service.getAllSubjects();
            res.json(subjects);
        } catch (e) {
            const status = e.status || 500;
            res.status(status).json(error(req, e.code, e.message, status));
        }
    },
```

Modify subjectController.js (2/5)

```
get: async function (req, res) {
    const idStr = (req.params.id || '').toString().trim();
    if (idStr === "") {
        return res.status(400).json(error(req, "Bad Request",
            "Bad Request: empty id", 400));
    }
    const id = Number(idStr);
    if (isNaN(id)) {
        return res.status(400).json(error(req, "Bad Request",
            "Bad Request: id must be a number", 400));
    }
    try {
        const subject = await service.getSubjectById(id);
        res.json(subject);
    } catch (e) {
        const status = e.status || 500;
        res.status(status).json(error(req, e.code, e.message, status));
    }
},
```

Modify subjectController.js (3/5)

```
create: async function (req, res) {
    const newSubject = req.body;
    try {
        const created = await service.addSubject(newSubject);
        res.status(201).json(created);
    } catch (e) {
        const status = e.status || (e.code === 'ER_DUP_ENTRY' ? 409 : 500);
        res.status(status).json(error(req, e.code, e.message, status));
    }
},
```

Modify subjectController.js (4/5)

```
update: async function (req, res) {
    const id = Number((req.params.id || '').toString().trim());
    if (isNaN(id)) {
        return res.status(400).json(error(req, "Bad Request",
            "Bad Request: id must be a number", 400));
    }
    try {
        const updated = await service.updateSubject(id, req.body);
        res.json(updated);
    } catch (e) {
        const status = e.status || (e.code === 'ER_DUP_ENTRY' ? 409 : 500);
        res.status(status).json(error(req, e.code, e.message, status));
    }
},
```

Modify subjectController.js (5/5)

```
remove: async function (req, res) {
    const id = Number((req.params.id || '').toString().trim());
    if (isNaN(id)) {
      return res.status(400).json(error(req, "Bad Request",
        "Bad Request: id must be a number", 400));
    }
    try {
      await service.removeSubject(id);
      res.status(204).send();
    } catch (e) {
      const status = e.status || 500;
      res.status(status).json(error(req, e.code, e.message, status));
    }
  }
}
```