To-do API Development with Spark

By Anny Patino
Software Engineer Fellow

Agenda

Project Overview

API Design and Implementation

API Endpoints and Methods

Testing and Validation

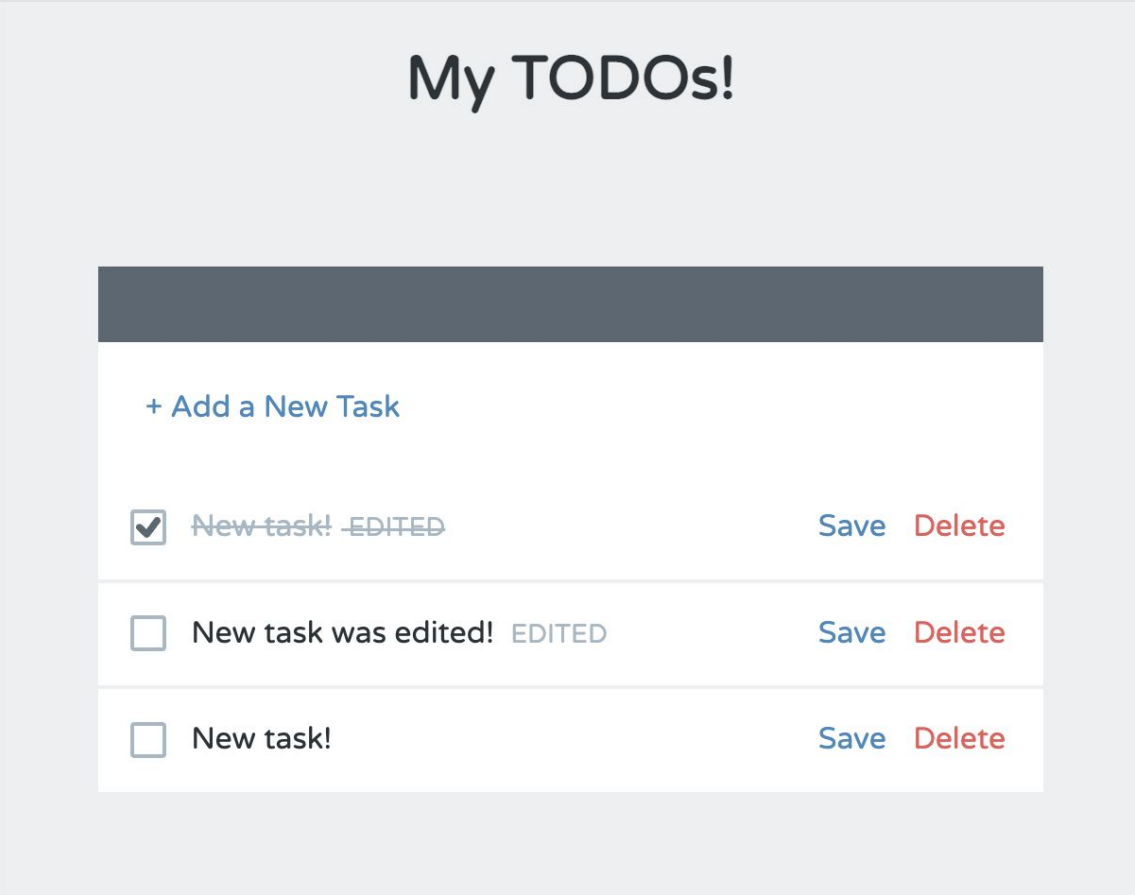
Q&A

Project Overview

Objective:

The goal was to provide a robust backend using Spark to handle RESTful requests from the front-end application.

Technology Stack:
- Spark Java
- Sql2o
- H2 database



Gradle Dependencies

- Sql2o
- H2database
- Spark Java Framework
- Gson
- JUnit

build.gradle (techdegrees)

```gradle
plugins {
    id 'java'
}

group = 'com.teamtreehouse'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.sql2o:sql2o:1.6.0'
    implementation 'com.h2database:h2:2.2.224'
    implementation 'com.sparkjava:spark-core:2.9.4'
    implementation 'com.google.code.gson:gson:2.11.0'
    implementation 'ch.qos.logback:logback-classic:1.2.3'

    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.2'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.2'

}

test {
    useJUnitPlatform()
}
```

API Design and Implementation

To-do Model

The To-do model represents the fundamental data structure within the application. It encapsulates the properties of a "to-do" item.

Todo
- id: int - name: String - isCompleted: boolean
+ getId( ): int + setId( ): void + getName( ): String + setName( ): void + isCompleted( ): boolean + setCompleted(boolean): void

Todo.java

```java
package com.teamtreehouse.techdegrees.model;

public class Todo {  59 usages  Anny Patino
    private int id;  5 usages
    private String name;  6 usages
    private boolean isCompleted;  6 usages

    // Constructor
    public Todo(String name, boolean isCompleted) {  18 usages  Anny Patino
        this.name = name;
        this.isCompleted = isCompleted;
    }

    // Getter and Setters

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public boolean isCompleted() { return isCompleted; }

    public void setCompleted(boolean completed) { isCompleted = completed; }

    @Override  Anny Patino
    public boolean equals(Object o) {
        if (o == null || getClass() != o.getClass()) return false;

        Todo todo = (Todo) o;
        return id == todo.id && isCompleted == todo.isCompleted && name.equals(todo.name);
    }

    @Override  Anny Patino
    public int hashCode() {
        int result = id;
        result = 31 * result + name.hashCode();
        result = 31 * result + Boolean.hashCode(isCompleted);
        return result;
    }
}
```

To-do DAO Interface

The interface abstracts the implementation details of data access. Defines the CRUD operations.

<<Interface>> **TodoDao**
+ add(todo: Todo): void + updateTodo(id: int, name: String, isCompleted: boolean): void + deleteTodo(id: int): void + findAll(): List<Todo> + findByTodoId(id: int): Todo

TodoDao.java

```java
package com.teamtreehouse.techdegrees.dao;

import com.teamtreehouse.techdegrees.exc.DaoException;
import com.teamtreehouse.techdegrees.model.Todo;

import java.util.List;

public interface TodoDao {  4 usages  1 implementation  Anny Patino
    void add(Todo todo) throws DaoException;  7 usages  1 implementation  Anny Patino
    void updateTodo(int id, String name, boolean isCompleted) throws Exception;
    void deleteTodo(int id);  2 usages  1 implementation  Anny Patino

    List<Todo> findAll();  3 usages  1 implementation  Anny Patino
    Todo findByTodoId(int id);  4 usages  1 implementation  Anny Patino
}
```

Sql2o To-do DAO Implementation

The Sql2oTodoDAO implements the TodoDao interface using Sql2o which allows for building SQL queries and mapping results directly to Java objects.

Sql2oTodoDao
- sql2o: Sql2o
+ add(todo: Todo): void + updateTodo(id: int, name: String, isCompleted: boolean): void + deleteTodo(id: int): void + findAll(): List<Todo> + findByTodoId(id: int): Todo

Sql2oTodoDao.java

```java
package com.teamtreehouse.techdegrees.dao;

import ...

public class Sql2oTodoDao implements TodoDao {  7 usages  Anny Patino *

    private final Sql2o sql2o;  6 usages

    public Sql2oTodoDao(Sql2o sql2o) {  3 usages  Anny Patino
        this.sql2o = sql2o;
    }

    @Override  10 usages  Anny Patino *
    public void add(Todo todo) throws DaoException {...}

    // Update a to-do
    @Override  1 usage  Anny Patino *
    public void updateTodo(int id, String name, boolean isCompleted) {...}

    // Delete a to-do
    @Override  2 usages  Anny Patino
    public void deleteTodo(int id) {...}

    // Find all todos
    @Override  3 usages  Anny Patino
    public List<Todo> findAll() {...}

    // Find a to-do by ID
    @Override  5 usages  Anny Patino
    public Todo findByTodoId(int id) {...}
}
```

Create Operation

Overview:

- The `add` method is responsible for adding new to-do items to the database.
- It constructs an SQL `INSERT` statement and executes it using the Sql2o library.

```java
@Override  10 usages  Anny Patino *
public void add(Todo todo) throws DaoException {
    // SQL query for inserting a new to-do into the database
    String sql = "INSERT INTO todos(name, isCompleted) VALUES (:name, :isCompleted)";

    // Open a new database connection and automatically closes the connection after execution
    try (Connection con = sql2o.open()) {
        Integer id = (Integer) con.createQuery(sql, returnGeneratedKeys: true) Query
                .addParameter( name: "name", todo.getName())
                .addParameter( name: "isCompleted", todo.isCompleted())
                .executeUpdate() Connection
                .getKey();
        if (id != null) {
            todo.setId(id); // syncs Java object with database
        } else {
            throw new DaoException("No ID generated for the Todo");
        }
    } catch (Sql2oException e) {
        throw new DaoException("Problem adding todo");
    }
}
```

Read Operation

Overview:

- The `Read` operation allows retrieving data from the database. It is split into two methods:
  - `findAll()`: Retrieves all to-do items.
  - `findByTodoId(int id)`: Retrieves a specific to-do item by its ID.

```java
    // Find all todos
    @Override  3 usages  Anny Patino
    public List<Todo> findAll() {
        String sql = "SELECT * FROM todos";
        try (Connection con = sql2o.open()) {
            return con.createQuery(sql).executeAndFetch(Todo.class);
        }
    }

    // Find a to-do by ID
    @Override  5 usages  Anny Patino
    public Todo findByTodoId(int id) {
        String sql = "SELECT * FROM todos WHERE id = :id";
        try (Connection con = sql2o.open()) {
            return con.createQuery(sql)
                    .addParameter( name: "id", id)
                    .executeAndFetchFirst(Todo.class);
        }
    }
}
```

Update Operation

Overview:

- The `Update` operation modifies an existing to-do item's attributes in the database.
- This is handled by the `updateTodo` method.

```
41  // Update a to-do
42  @Override  1 usage  Anny Patino *
43  public void updateTodo(int id, String name, boolean isCompleted) {
44      String sql = "UPDATE todos SET name = :name, isCompleted = :isCompleted WHERE id = :id";
45      try (Connection con = sql2o.open()) {
46          con.createQuery(sql)
47                  .addParameter( name: "id", id)
48                  .addParameter( name: "name", name)
49                  .addParameter( name: "isCompleted", isCompleted)
50                  .executeUpdate();
51      }
52  }
```

Delete Operation

Overview:

- The `Delete` operation removes a specific to-do item from the database.
- This functionality is implemented through the `deleteTodo` method.

```java
    // Delete a to-do
    @Override  2 usages  Anny Patino
    public void deleteTodo(int id) {
        String sql = "DELETE FROM todos WHERE id = :id";
        try (Connection con = sql2o.open()) {
            con.createQuery(sql)
                    .addParameter( name: "id", id)
                    .executeUpdate();
        }
    }
```

API Endpoints and Methods

Purpose of the `Api` Class:

- Acts as the **controller** for the application, managing all incoming HTTP requests and delegating tasks to the `TodoDao` interface for data persistence.
- Centralizes the setup for routing, database connections, and configuration.

Api.java

```java
package com.teamtreehouse.techdegrees;

import ...

public class Api {  Anny Patino

    public static void main(String[] args) {  Anny Patino
        String datasource = "jdbc:h2:~/todos.db";
        if (args.length > 0) {
            if (args.length != 2 ) {
                System.out.println("java API <port> <datasource>");
                System.exit( status: 0);
            }
            port(Integer.parseInt(args[0]));
            datasource = args[1];
        }

        port(4567);
        staticFileLocation( folder: "/public");
        Sql2o sql2o = new Sql2o(
                String.format("%s;INIT=RUNSCRIPT from 'classpath:db/init.sql'", datasource)
                , user: "", pass: "" );

        // Initialize DAO
        TodoDao todoDao = new Sql2oTodoDao(sql2o);

        // JSON transformation
        Gson gson = new Gson();

        // Default response type is JSON
        before( path: "/api/v1/*", ( Request req, Response res) -> res.type( contentType: "application/json"));

        // Setup Routes
        routes(todoDao, gson);

    }
```

GET /api/v1/todos

Functionality:

- This route is responsible for **fetching all "to-do" items** from the database and returning them in JSON format.
- Implements the GET method to retrieve resources.

```
private static void routes(TodoDao todoDao, Gson gson) {  1 usage  Anny Patino *
    // Route that fetches all todos from the database
    get( path: "/api/v1/todos",  acceptType: "application/json", ( Request req,  Response res) -> {
        try {
            return todoDao.findAll();
        } catch (Exception e) {
            res.status( statusCode: 500); // Internal Server Error
            res.type( contentType: "application/json");
            return gson.toJson(new ErrorMessage("Failed to fetch to-dos: " + e.getMessage()));
        }
    }, gson::toJson);
```

POST /api/v1/todos

Functionality:

- This route allows the client to **create a new "to-do" item** by sending data in the request body.
- Implements the POST method to create a resource.

```
// Route to add new to-do to the database
post( path: "/api/v1/todos",  acceptType: "application/json", ( Request req,  Response res) -> {
    Todo todo = gson.fromJson(req.body(), Todo.class);
    todoDao.add(todo);
    res.status( statusCode: 201); // Created
    return todo;
}, gson::toJson);
```

PUT /api/v1/todos/{id}

Functionality:

- This endpoint is used to **update an existing "to-do" item** based on its unique identifier (id).
- Implements the PUT method to update a resource.

```
// Updating existing to-do
put( path: "/api/v1/todos/:id", acceptType: "application/json", ( Request req, Response res) -> {
    try {
        int id = Integer.parseInt(req.params(":id"));
        Todo updatedTodo = gson.fromJson(req.body(), Todo.class);

        // fallback
        Todo existingTodo = todoDao.findByTodoId(id);

        if (existingTodo == null) {
            res.status( statusCode: 404); // Not Found
            return gson.toJson( src: "To-do not found");
        }

        // Update to-do with new values
        if (updatedTodo.getName() != null) existingTodo.setName(updatedTodo.getName());
        if (updatedTodo.isCompleted() != existingTodo.isCompleted())
            existingTodo.setCompleted(updatedTodo.isCompleted());

        res.status( statusCode: 200); // ok
        return "Todo updated successfully";
    } catch (Exception e) {
        res.status( statusCode: 500);
        res.type( contentType: "application/json");
        return gson.toJson(new ErrorMessage("An error occurred: " + e.getMessage()));
    }
}, gson::toJson);
```

DELETE /api/v1/todos/{id}

Functionality:

- This endpoint is used to **delete an existing "to-do" item** from the database using its unique identifier (id).
- Implements the DELETE method to remove a resource.

```
// Route to delete a to-do from the database
delete( path: "/api/v1/todos/:id", acceptType: "application/json", ( Request req, Response res) -> {
    int id = Integer.parseInt(req.params(":id"));
    Todo todo = todoDao.findByTodoId(id);

    if (todo != null) {
        todoDao.deleteTodo(id);
        res.status( statusCode: 204); // No Content
        return "";
    } else {
        res.status( statusCode: 404); // Not Found
        return gson.toJson( src: "To-do not found");
    }
});
```

Testing and Validation

Unit Testing

Focus: Targeted the core functionalities of the Todo model and data access object (DAO) layers.

Tools Used: JUnit for assertions and test lifecycle management.

TodoTest.java

```java
package com.teamtreehouse.techdegrees.model;

import ...

class TodoTest {  Anny Patino *

    // Testing Getters and Setters
    @Test  Anny Patino
    void getIdShouldReturnCorrectId() {...}

    @Test  Anny Patino
    void setIdShouldSetIdCorrectly() {...}

    @Test  Anny Patino
    void getNameShouldReturnCorrectName() {...}

    @Test  Anny Patino
    void setNameShouldSetNameCorrectly() {...}

    @Test  Anny Patino
    void isCompletedShouldReturnCorrectCompletionStatus() {...}

    @Test  Anny Patino
    void setCompletedShouldSetCompletionStatusCorrectly() {...}

    // Test that To-do model correctly implements equality checks
    @Test  Anny Patino
    void testEqualsSameAttributesShouldReturnTrue() {...}

    @Test  Anny Patino
    void testEqualsShouldReturnFalseForDifferentAttributes() {...}

    // Test hash code generation
    @Test  Anny Patino
    void testHashCode() {...}
}
```

Sql2oTodoDaoTest.java

```java
package com.teamtreehouse.techdegrees.dao;

import ...

public class Sql2oTodoDaoTest {  Anny Patino *

    private Sql2oTodoDao dao;  13 usages
    private Connection con;  4 usages

    @BeforeEach  Anny Patino *
    public void setUp() throws Exception {
        // Setup the database with H2 in memory and create tables
        String connectionString = "jdbc:h2:mem:testing;DB_CLOSE_DELAY=-1;";

        Sql2o sql2o = new Sql2o(connectionString, user: null, pass: null);
        dao = new Sql2oTodoDao(sql2o);

        con = sql2o.beginTransaction();

        String sqlCreateTable = "CREATE TABLE IF NOT EXISTS todos " +
                "(id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(255), isCompleted BOOLEAN);";

        con.createQuery(sqlCreateTable).executeUpdate();
    }

    @AfterEach  Anny Patino
    public void tearDown() throws Exception {...}

    @Test  Anny Patino
    public void addingTodoSetsId() throws DaoException {...}

    @Test  Anny Patino
    public void addedTodosAreReturnedFromFindAll() throws  Exception{...}

    @Test  Anny Patino
    public void updatingTodoChangesValues() throws DaoException {...}

    @Test  Anny Patino
    public void deletingTodoRemovesIt() throws DaoException {...}

    @Test  Anny Patino
    public void findTodoByIdReturnsCorrectTodo() throws Exception {...}
}
```

Functional Testing

Objective: Verify that the API endpoints respond correctly to HTTP requests and interact correctly with the database.

ApiFunctionalTest.java

```java
package com.teamtreehouse.techdegrees.api;

import ...

public class ApiFunctionalTest {  Anny Patino

    public static final String PORT = "4567";  2 usages
    public static final String TEST_DATASOURCE = "jdbc:h2:mem:testing";  2 usages
    private Connection conn;  2 usages
    private ApiClient client;  6 usages
    private Gson gson;  3 usages
    private Sql2oTodoDao todoDao;  5 usages

    @BeforeAll  Anny Patino
    public static void startServer() throws Exception {
        String[] args = {PORT, TEST_DATASOURCE};
        Api.main(args);
    }

    @AfterAll  Anny Patino
    public static void stopServer() throws Exception {
        Spark.stop();
    }

    @BeforeEach  Anny Patino
    public void setUp() throws Exception {
        Sql2o sql2o = new Sql2o( url: TEST_DATASOURCE + ";INIT=RUNSCRIPT from 'classpath:db/init.sql'",  user: "",  pass: "");
        conn = sql2o.open();
        client = new ApiClient( server: "http://localhost:" + PORT);
        gson = new Gson();
        todoDao = new Sql2oTodoDao(sql2o);
    }

    @AfterEach  Anny Patino
    public void tearDown() throws Exception {
        conn.close();
    }
```

ApiFunctionalTest.java

```java
public class ApiFunctionalTest {  Anny Patino
    void addingTodoReturnsCreatedStatus() throws Exception {
        Map<String, Object> values = new HashMap<>();
        values.put("name", "Test");
        values.put("isCompleted", false);

        ApiResponse res = client.request( method: "POST",  url: "/api/v1/todos", gson.toJson(values));

        assertEquals( expected: 201, res.getStatus());
    }

    @Test  Anny Patino
    public void todoCanBeSuccessfullyDeleted () throws Exception{
        Todo todo = new Todo( name: "test",  isCompleted: false);
        todoDao.add(todo);
        int id = todo.getId();

        ApiResponse res = client.request( method: "DELETE",  url: "/api/v1/todos/" + todo.getId());

        assertEquals( expected: 204, res.getStatus());
        assertNull(todoDao.findByTodoId(id));
    }

    @Test  Anny Patino
    public void deletingToDoReturnsNoContentStatus() throws Exception{
        Todo todo = new Todo( name: "test",  isCompleted: false);
        todoDao.add(todo);

        ApiResponse res = client.request( method: "DELETE",  url: "/api/v1/todos/" + todo.getId());

        assertEquals( expected: 204, res.getStatus());
    }

    @Test  Anny Patino
    public void updatingTodoReturnsSuccessfulStatus() throws Exception {
        Todo todo = new Todo( name: "test",  isCompleted: false);
        todoDao.add(todo);
        Map<String, Object> values = new HashMap<>();
        values.put("name", "change");
        values.put("isCompleted", true);

        ApiResponse res = client.request( method: "PUT",  url: "/api/v1/todos/" + todo.getId(), gson.toJson(values));

        assertEquals( expected: 200, res.getStatus());
```

Questions?

Thank you

https://github.com/apatino16/todoAPIWithSpark