Jimma Institute of Technology

Faculty of Computing and Informatics

Department of Software Engineering

ELECTIVE I: WEB SERVICE GROUP ASSIGNMENT

October, 2025

Jimma, Ethiopia

1. Java Web Service Development

1.1 Overview

This exercise focused on developing a web service using Java, emphasizing how Java technologies are used to build and deploy backend services

1.2 Objectives

The exercise is specifically aimed to:

- Understand Java frameworks for web service development.
- Implement RESTful endpoints that perform CRUD operations.
- Learn how Java integrates with tools like Maven for dependency management.

1.3 Implementation Steps

Step 1: Created a new Spring Boot project using Spring Initializr with dependencies: Spring Web, Spring Boot DevTools.

Step 2: Implemented REST controllers with endpoints for CRUD operations (/create, /read, /update, /delete).

Step 3: Added a service layer to handle business logic.

Step 4: Executed the Spring Boot server and verified endpoints were available on http://localhost:8080.

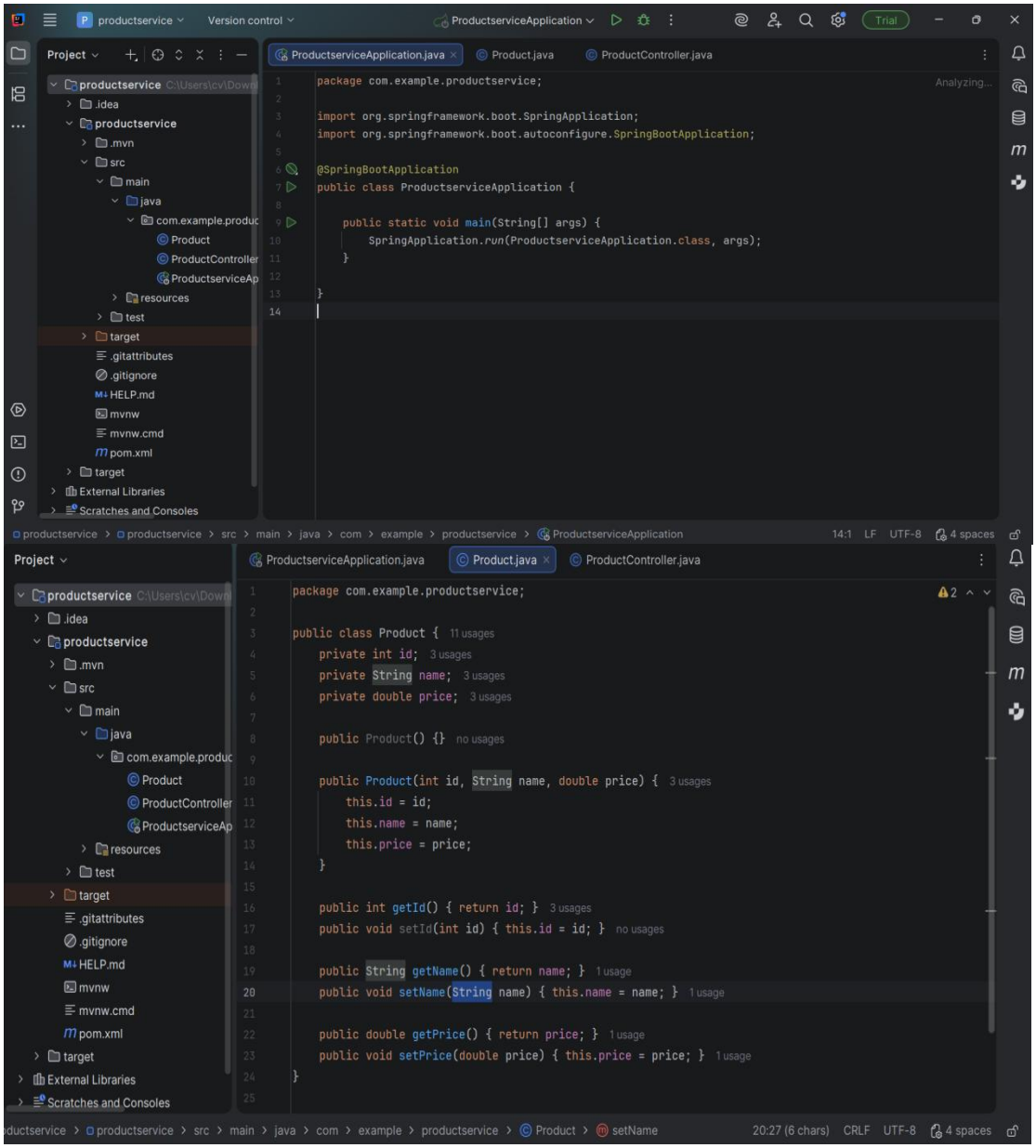
Step 5: Sent sample HTTP requests using Postman to validate API functionality.

1.4 Tools and Technologies

- ✓ Java WEB service
- ✓ Java 17
- ✓ Spring Boot 3.x
- ✓ Maven
- ✓ IntelliJ IDEA
- ✓ Java Classes
- ✓ Spring Annotations

1.5 Results and Observations

- E All endpoints successfully handled CRUD operations and returned expected JSON responses.
- The REST endpoints worked correctly using JSON data.
- CRUD operations were successfully tested.
- Spring Boot automatically handled JSON conversion and routing.
- Postman helped visualize requests and responses easily.


productservice
Version control
ProductserviceApplication
Trial
Project
ProductserviceApplication.java
Product.java
ProductController.java
Analyzing...
productservice C:\Users\cv\Down
.idea
productservice
.mvn
src
main
java
com.example.produc
Product
ProductController
ProductserviceAp
resources
test
target
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
target
External Libraries
Scratches and Consoles
package com.example.productservice;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class ProductserviceApplication {
public static void main(String[] args) {
SpringApplication.run(ProductserviceApplication.class, args);
}
}
productservice › productservice › src › main › java › com › example › productservice › ProductserviceApplication
14:1 LF UTF-8 4 spaces
Project
ProductserviceApplication.java
Product.java
ProductController.java
productservice C:\Users\cv\Down
.idea
productservice
.mvn
src
main
java
com.example.produc
Product
ProductController
ProductserviceAp
resources
test
target
.gitattributes
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
target
External Libraries
Scratches and Consoles
package com.example.productservice;
public class Product { 11 usages
private int id; 3 usages
private String name; 3 usages
private double price; 3 usages
public Product() {} no usages
public Product(int id, String name, double price) { 3 usages
this.id = id;
this.name = name;
this.price = price;
}
public int getId() { return id; } 3 usages
public void setId(int id) { this.id = id; } no usages
public String getName() { return name; } 1 usage
public void setName(String name) { this.name = name; } 1 usage
public double getPrice() { return price; } 1 usage
public void setPrice(double price) { this.price = price; } 1 usage
}
productservice › productservice › src › main › java › com › example › productservice › Product › setName
20:27 (6 chars) CRLF UTF-8 4 spaces

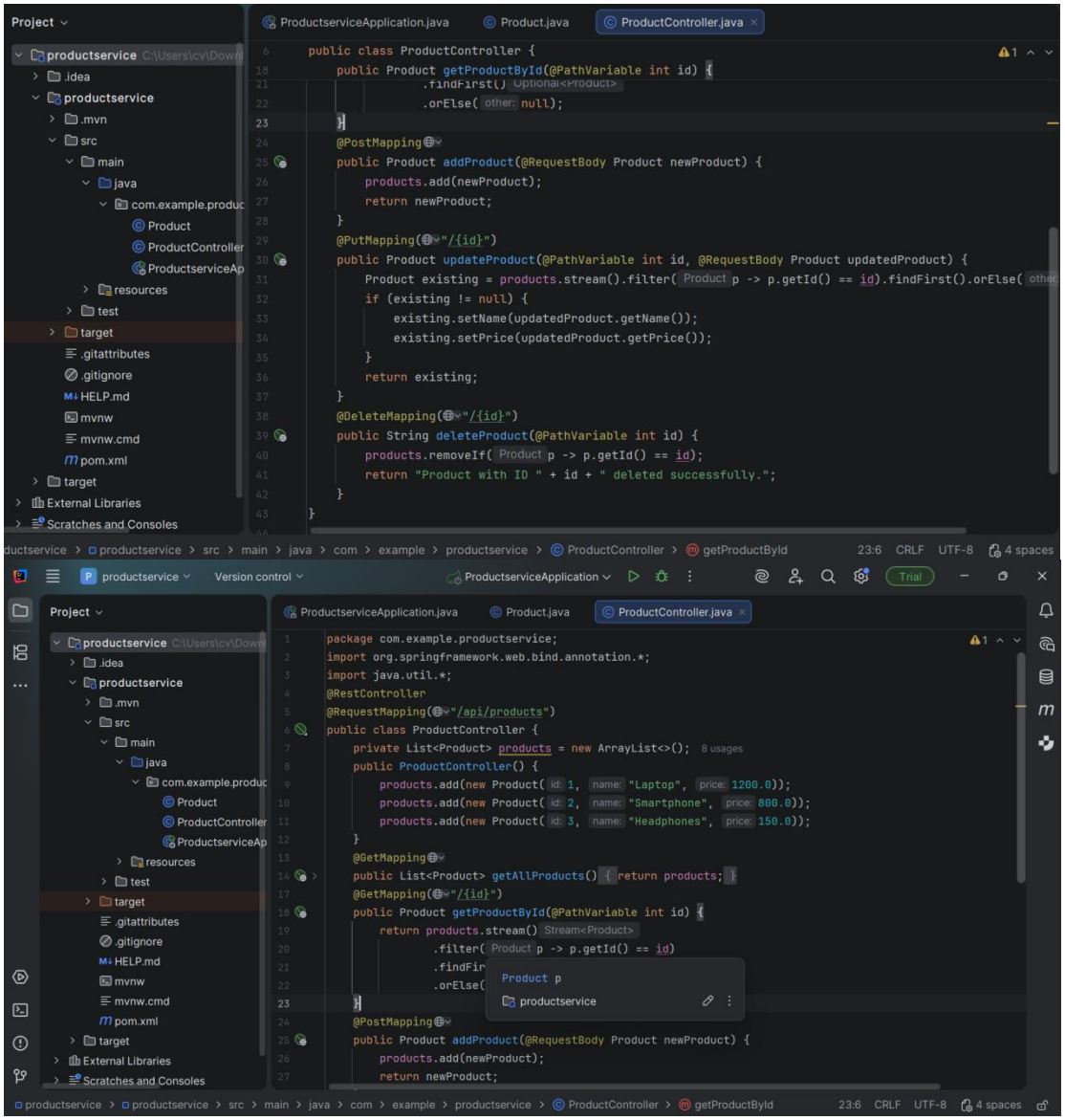


Figure 1Figures showing the setup of the Spring Boot project, REST controller implementation, and API testing using Postman.

2. Web API Design and Testing Tools (Postman)

2.1 Overview

This exercise centered on testing and validating web services using Postman, a widely used API client tool.

2.2 Objectives

The objective was to:

- Learn how to send HTTP requests to RESTful endpoints.
- Validate request parameters, headers, and response data.
- Understand API testing workflows, including GET, POST, PUT, and DELETE requests.

2.3 Implementation Steps

Step 1: Installed and configured Postman

Step 2: Created requests matching REST endpoints (GET, POST, PUT, DELETE).

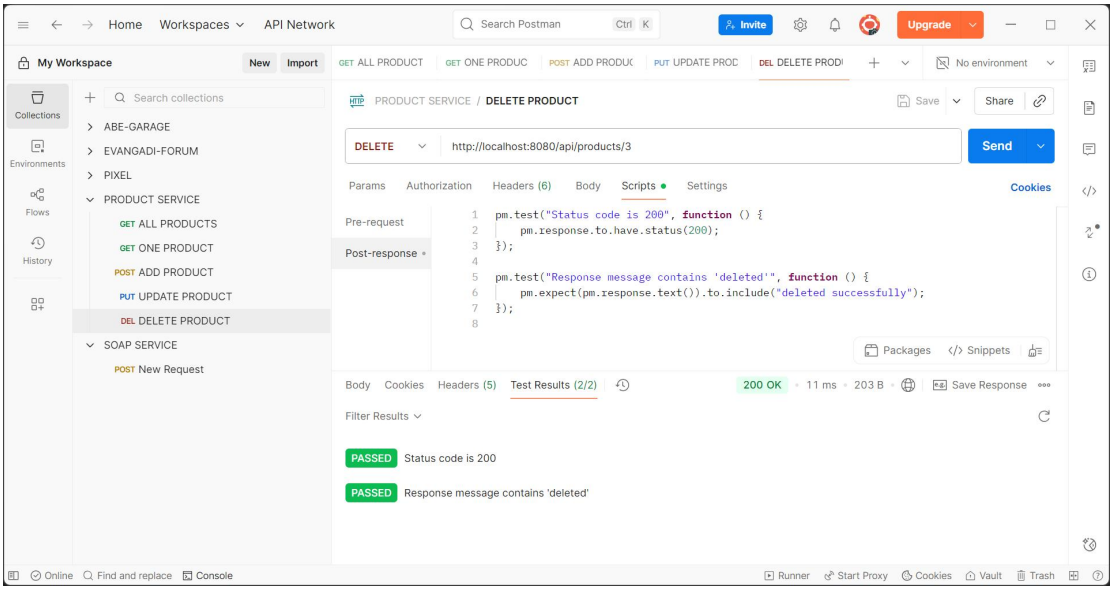
Step 3: Added path parameters, query parameters, and JSON payloads where necessary.

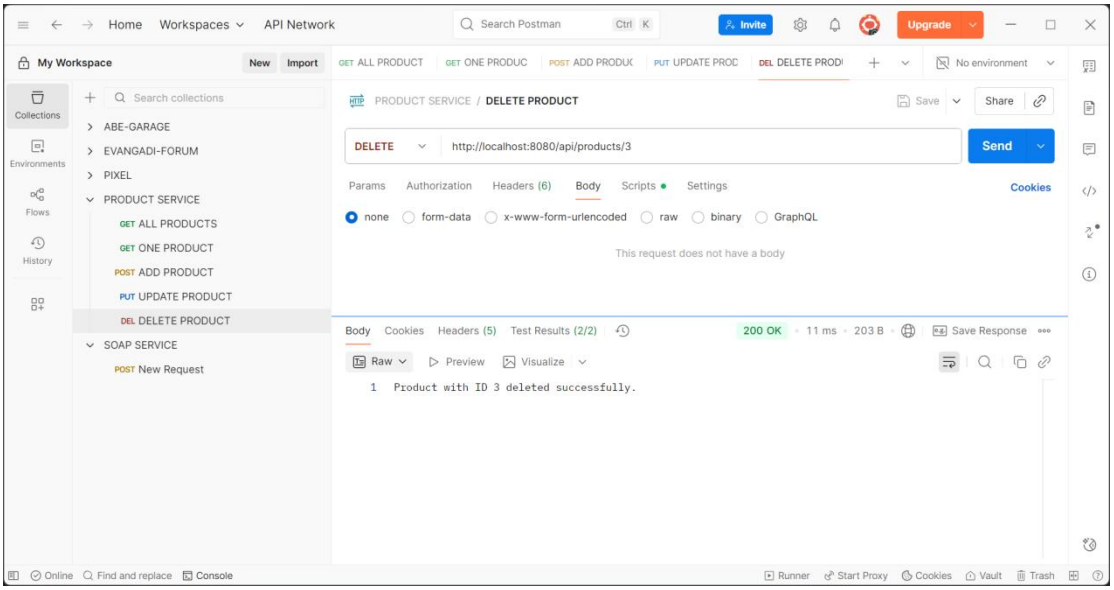
Step 4: Observed status codes, headers, and JSON responses.

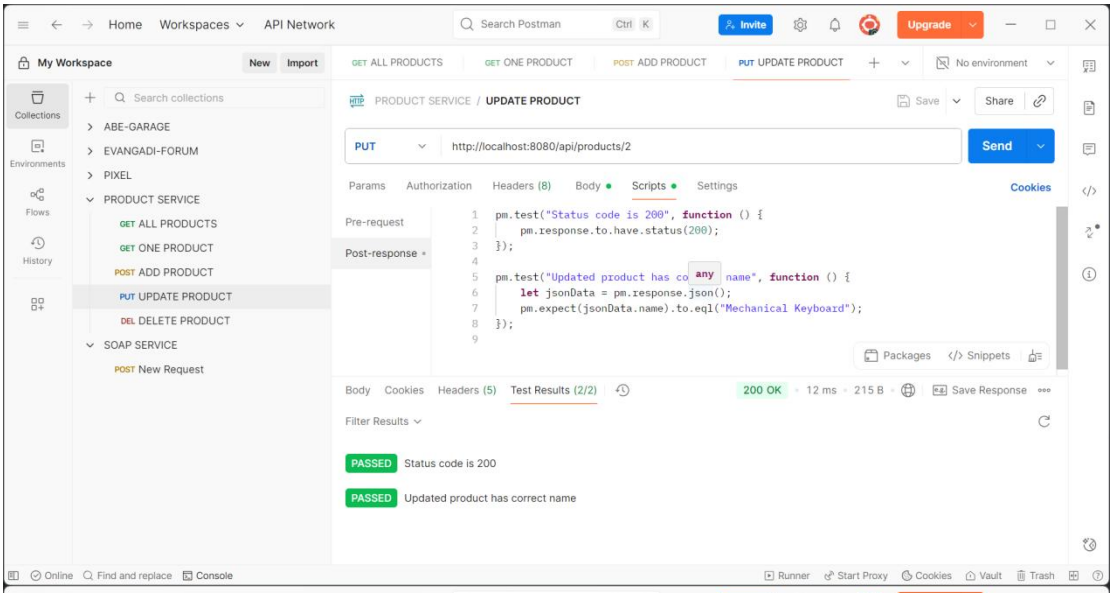
Step 5: Saved requests in Postman collection for reproducibility.

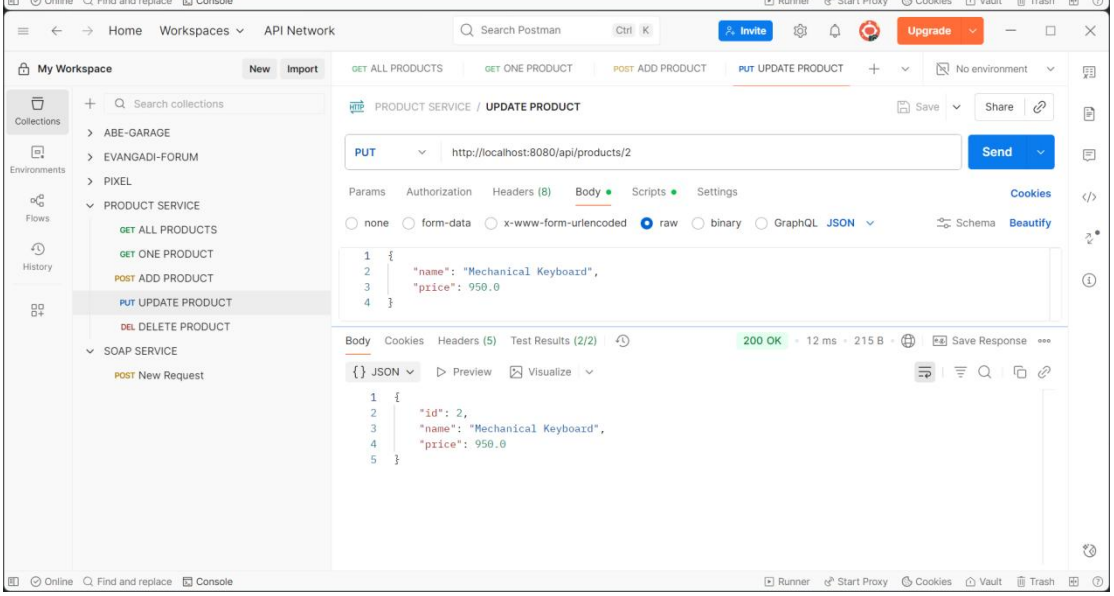
2.4 Tools and Technologies

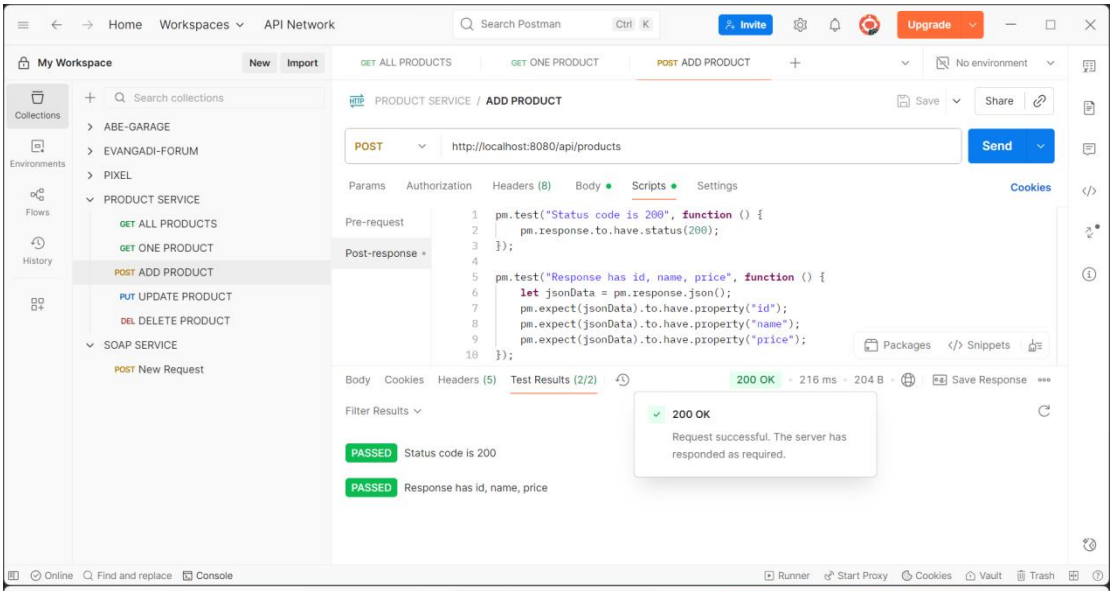
- ✓ Postman
- ✓ Spring Boot REST API
- ✓ Java 17
- ✓ Maven

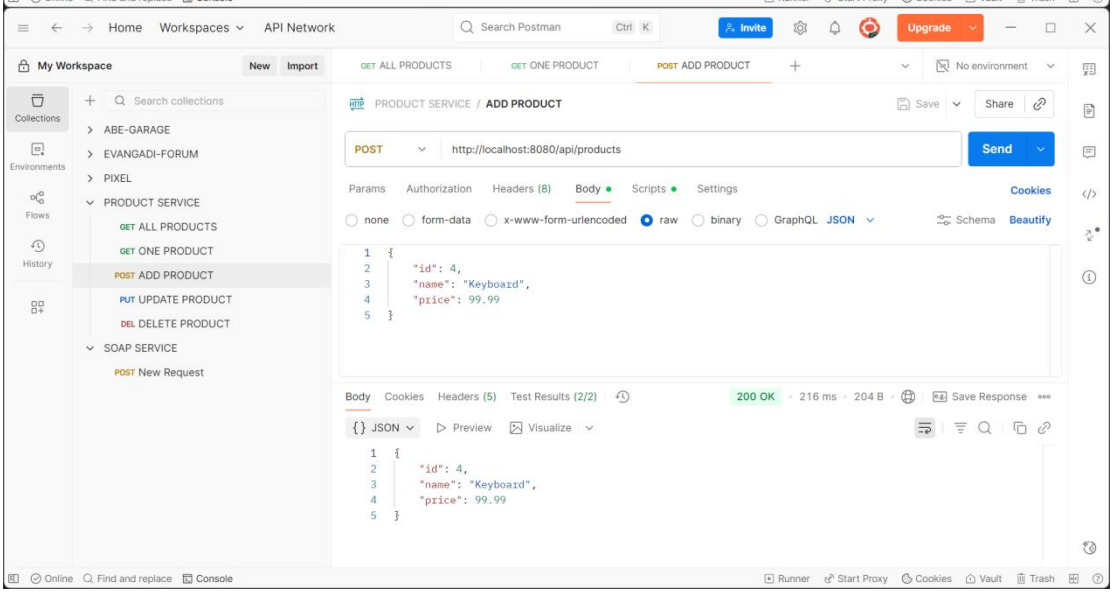



Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
My Workspace
New
Import
GET ALL PRODUCT
GET ONE PRODUC
POST ADD PRODUC
PUT UPDATE PROC
DEL DELETE PROD
No environment
Collections
Environments
Flows
History
Search collections
ABE-GARAGE
EVANGADI-FORUM
PIXEL
PRODUCT SERVICE
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
DEL DELETE PRODUCT
SOAP SERVICE
POST New Request
PRODUCT SERVICE / DELETE PRODUCT
Save
Share
DELETE
http://localhost:8080/api/products/3
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body
Body
Cookies
Headers (5)
Test Results (2/2)
200 OK
11 ms
203 B
Save Response
Raw
Preview
Visualize
1 Product with ID 3 deleted successfully.
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
My Workspace
New
Import
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
No environment
Collections
Environments
Flows
History
Search collections
ABE-GARAGE
EVANGADI-FORUM
PIXEL
PRODUCT SERVICE
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
DEL DELETE PRODUCT
SOAP SERVICE
POST New Request
PRODUCT SERVICE / UPDATE PRODUCT
Save
Share
PUT
http://localhost:8080/api/products/2
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status code is 200", function () {
pm.response.to.have.status(200);
});
pm.test("Updated product has co any name", function () {
let jsonData = pm.response.json();
pm.expect(jsonData.name).to.eql("Mechanical Keyboard");
});
Packages
Snippets
Body
Cookies
Headers (5)
Test Results (2/2)
200 OK
12 ms
215 B
Save Response
Filter Results
PASSED
Status code is 200
PASSED
Updated product has correct name
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
My Workspace
New
Import
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
No environment
Collections
Environments
Flows
History
Search collections
ABE-GARAGE
EVANGADI-FORUM
PIXEL
PRODUCT SERVICE
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
DEL DELETE PRODUCT
SOAP SERVICE
POST New Request
PRODUCT SERVICE / UPDATE PRODUCT
Save
Share
PUT
http://localhost:8080/api/products/2
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"name": "Mechanical Keyboard",
"price": 950.0
}
Body
Cookies
Headers (5)
Test Results (2/2)
200 OK
12 ms
215 B
Save Response
JSON
Preview
Visualize
{
"id": 2,
"name": "Mechanical Keyboard",
"price": 950.0
}
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
My Workspace
New
Import
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
No environment
Collections
Environments
Flows
History
Search collections
ABE-GARAGE
EVANGADI-FORUM
PIXEL
PRODUCT SERVICE
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
DEL DELETE PRODUCT
SOAP SERVICE
POST New Request
PRODUCT SERVICE / ADD PRODUCT
Save
Share
POST
http://localhost:8080/api/products
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Pre-request
Post-response
pm.test("Status code is 200", function () {
pm.response.to.have.status(200);
});
pm.test("Response has id, name, price", function () {
let jsonData = pm.response.json();
pm.expect(jsonData).to.have.property("id");
pm.expect(jsonData).to.have.property("name");
pm.expect(jsonData).to.have.property("price");
});
Packages
Snippets
Body
Cookies
Headers (5)
Test Results (2/2)
200 OK
216 ms
204 B
Save Response
Filter Results
200 OK
Request successful. The server has responded as required.
PASSED
Status code is 200
PASSED
Response has id, name, price
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
My Workspace
New
Import
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
No environment
Collections
Environments
Flows
History
Search collections
ABE-GARAGE
EVANGADI-FORUM
PIXEL
PRODUCT SERVICE
GET ALL PRODUCTS
GET ONE PRODUCT
POST ADD PRODUCT
PUT UPDATE PRODUCT
DEL DELETE PRODUCT
SOAP SERVICE
POST New Request
PRODUCT SERVICE / ADD PRODUCT
Save
Share
POST
http://localhost:8080/api/products
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Schema
Beautify
{
"id": 4,
"name": "Keyboard",
"price": 99.99
}
Body
Cookies
Headers (5)
Test Results (2/2)
200 OK
216 ms
204 B
Save Response
{} JSON
Preview
Visualize
{
"id": 4,
"name": "Keyboard",
"price": 99.99
}
Online
Find and replace
Console
Runner
Start Proxy
Cookies
Vault
Trash

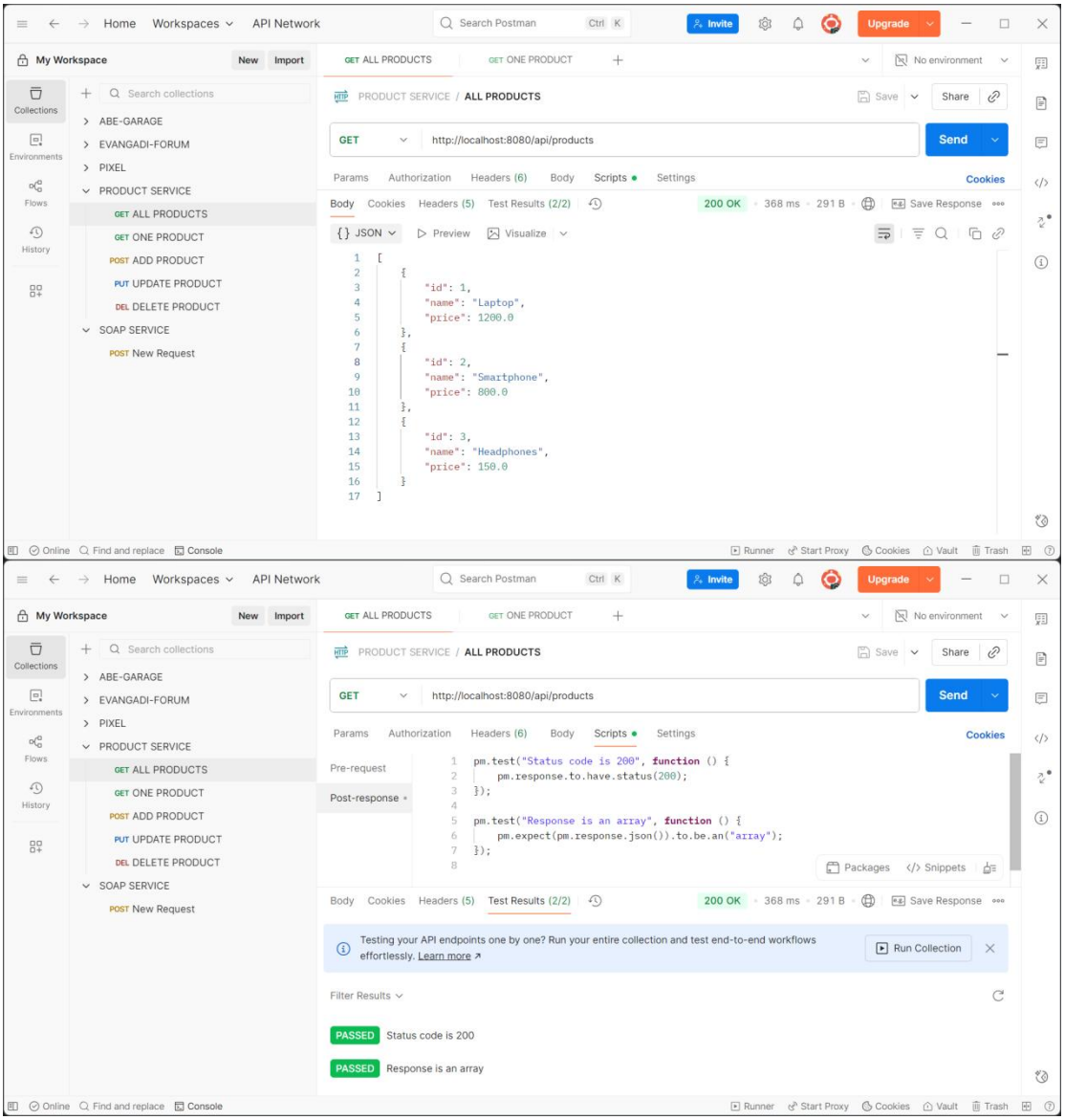


Figure 2Figures demonstrating API testing in Postman, including request creation, response validation, and collection management.

3. SOAP-based Web Services

3.1 Overview

The third exercise focused on creating a SOAP-based web service using Node.js and the soap library

3.2 Objectives

- Understand the structure of SOAP messages and WSDL (Web Services Description Language).
- Implement a SOAP server with multiple operations (e.g., sayHello, addNumbers, and getStudentName).
- Learned to consume SOAP services using clients and analyze XML-based responses.

3.3 Implementation Steps

Step 1: Created a Node.js project with npm init, installed dependencies soap and express.

Step 2: Designed service.wsdl with operations: sayHello, addNumbers, and getStudentName.

Step 3: Created server.js with SOAP service methods and exposed endpoint /wsdl.

Step 4: Created client.js to consume the SOAP service using the WSDL URL.

Step 5: Verified responses in Node console and Postman (SOAP request with XML body).

3.4 Tools and Technologies

- ✓ SOAP-based Web Services
- ✓ Node.js (v22.19.0)
- ✓ npm
- ✓ WSDL (XML files)
- ✓ VS Code
- ✓ express, soap library

3.5 Web API Testing

Postman, Browser (for quick endpoint checks)

GET /students/1: Returned student details.

POST /students: Added a new student with JSON payload.

PUT /students/2: Updated existing student information.

DELETE /students/3: Removed a student record.

3.6 Observation

- ➢ The SOAP service correctly returned XML responses for all operations (sayHello, addNumbers, and getStudentName).
- ➢ Node.js implementation allowed multiple operations and WSDL-based client consumption.

3.7 SOAP-based Web Services

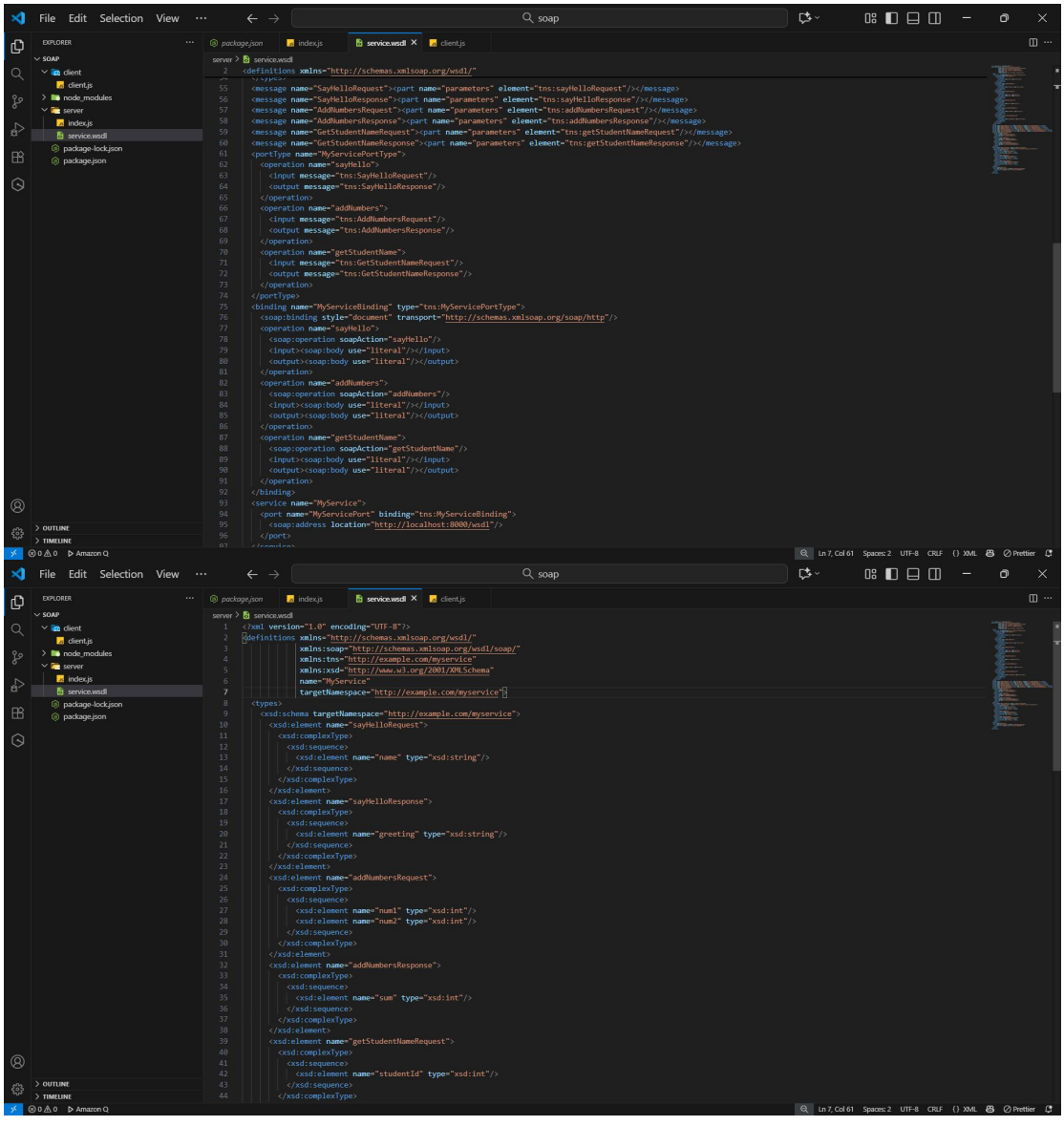
Sample Request (sayHello):

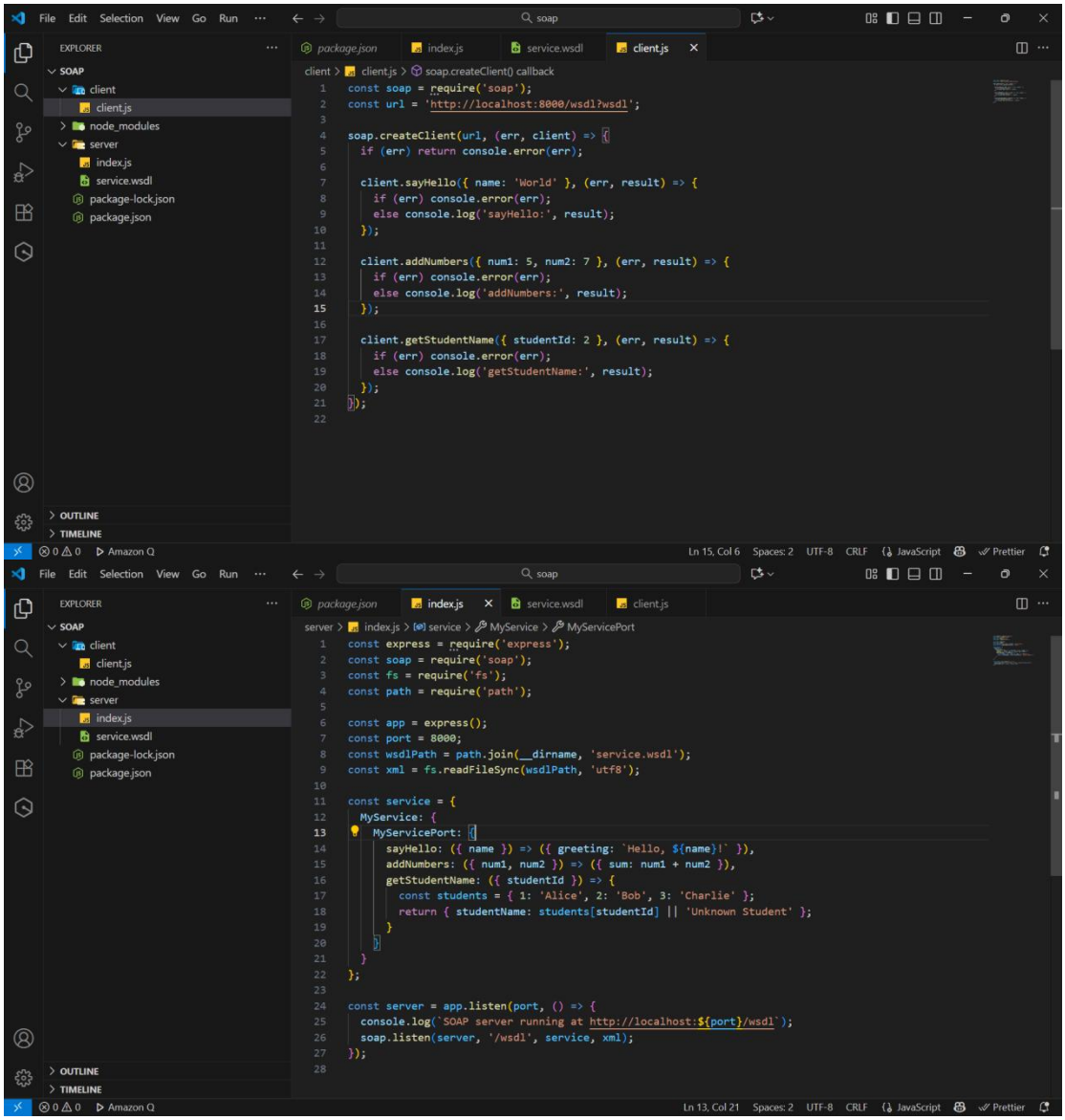
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://example.com/myservice">
  <soapenv:Header/>
  <soapenv:Body>
    <tns:sayHelloRequest>
      <name>World</name>
```

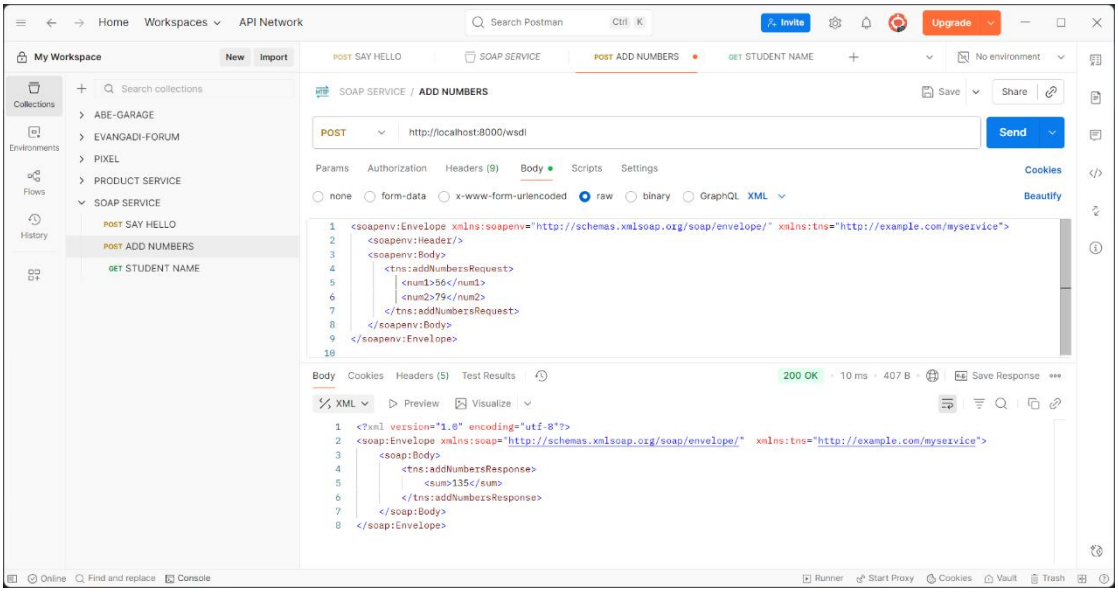
```
    </tns:sayHelloRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Sample Response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:tns="http://example.com/myservice">
  <soap:Body>
    <tns:sayHelloResponse>
      <greeting>Hello, World!</greeting>
    </tns:sayHelloResponse>
  </soap:Body>
</soap:Envelope>
```




client.js
const soap = require('soap');
const url = 'http://localhost:8000/wsdl?wsdl';
soap.createClient(url, (err, client) => {
if (err) return console.error(err);
client.sayHello({ name: 'World' }, (err, result) => {
if (err) console.error(err);
else console.log('sayHello:', result);
});
client.addNumbers({ num1: 5, num2: 7 }, (err, result) => {
if (err) console.error(err);
else console.log('addNumbers:', result);
});
client.getStudentName({ studentId: 2 }, (err, result) => {
if (err) console.error(err);
else console.log('getStudentName:', result);
});
});
index.js
const express = require('express');
const soap = require('soap');
const fs = require('fs');
const path = require('path');
const app = express();
const port = 8000;
const wsdlPath = path.join(__dirname, 'service.wsdl');
const xml = fs.readFileSync(wsdlPath, 'utf8');
const service = {
MyService: {
MyServicePort: {
sayHello: ({ name }) => ({ greeting: `Hello, ${name}!` }),
addNumbers: ({ num1, num2 }) => ({ sum: num1 + num2 }),
getStudentName: ({ studentId }) => {
const students = { 1: 'Alice', 2: 'Bob', 3: 'Charlie' };
return { studentName: students[studentId] || 'Unknown Student' };
}
}
}
};
const server = app.listen(port, () => {
console.log(`SOAP server running at http://localhost:${port}/wsdl`);
soap.listen(server, '/wsdl', service, xml);
});


POST http://localhost:8000/wsdl
Send
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://example.com/myservice">
<soapenv:Header/>
<soapenv:Body>
<tns:addNumbersRequest>
<num1>56</num1>
<num2>79</num2>
</tns:addNumbersRequest>
</soapenv:Body>
</soapenv:Envelope>
200 OK
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:tns="http://example.com/myservice">
<soap:Body>
<tns:addNumbersResponse>
<sum>135</sum>
</tns:addNumbersResponse>
</soap:Body>
</soap:Envelope>

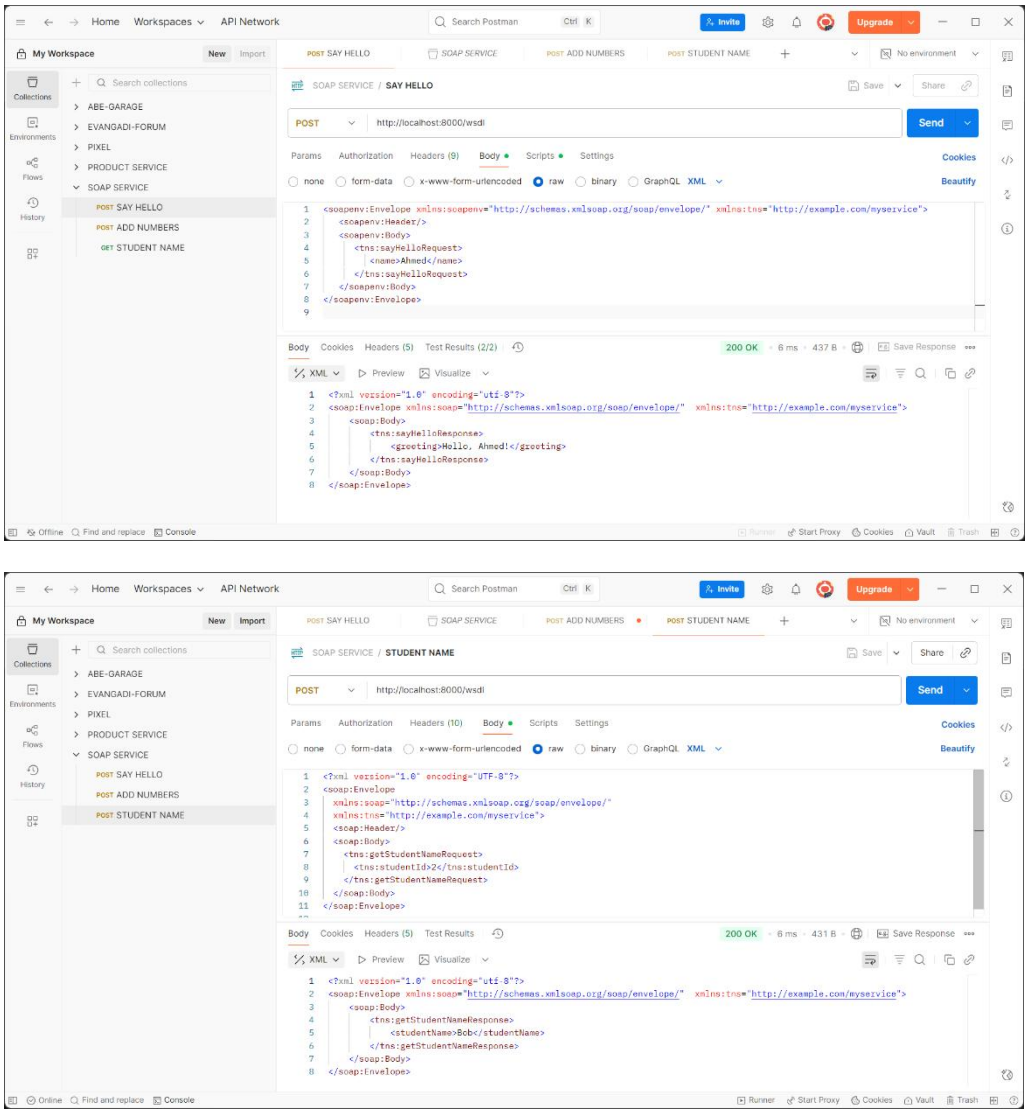


Figure 3Figures displaying the Node.js SOAP service setup, WSDL file design, and sample XML request–response exchanges.