Create your First REST API with Spring Boot

Estimated time needed: 40 minutes

Overview

In this lab, you will use `Spring Initializr` to generate a project to create a Library Spring Boot web application with REST API endpoints. You will use Java Collections to maintain a database of the books.

Learning objectives

After completing this lab, you will be able to:

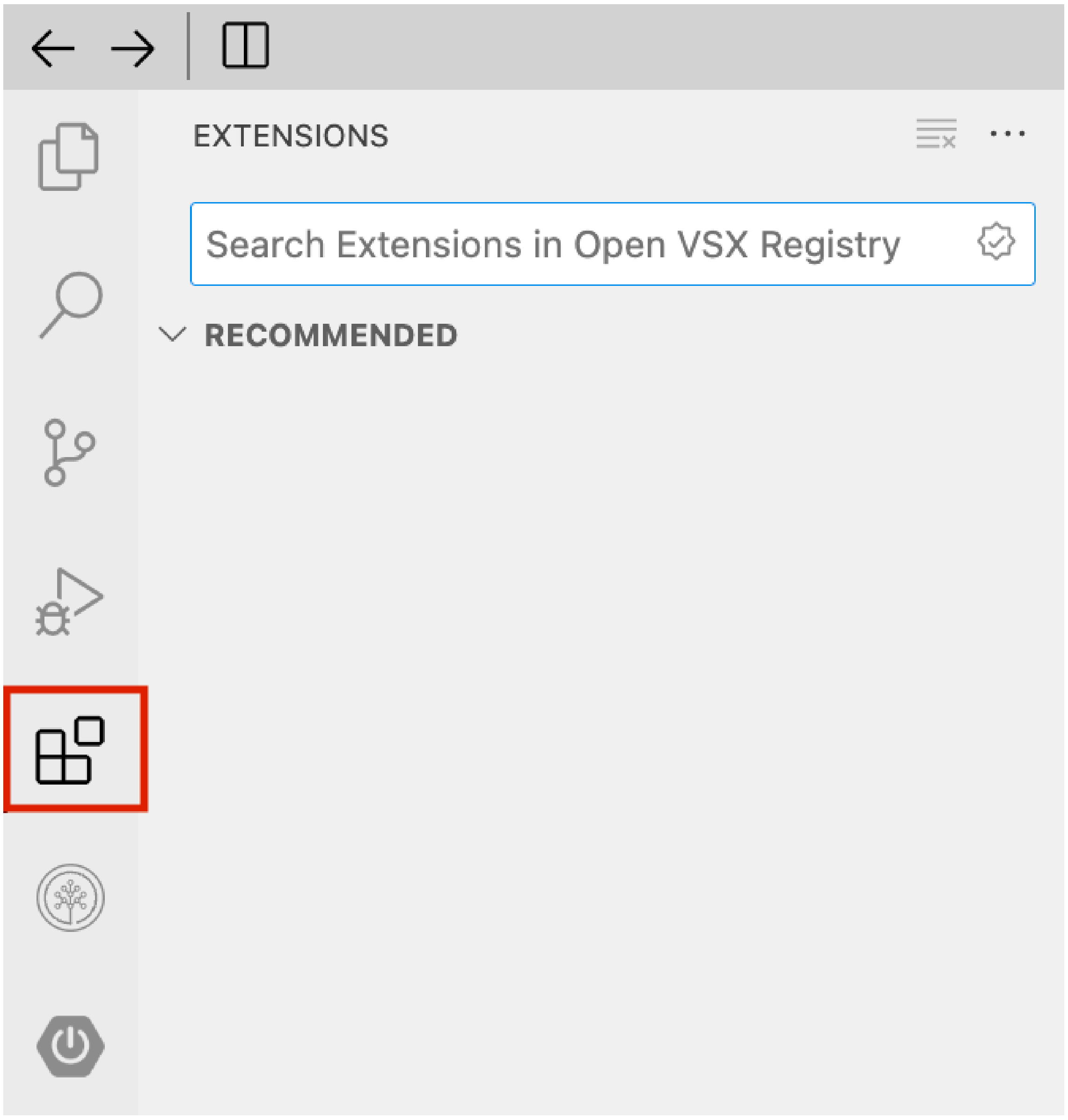
- Use Spring Initializr to generate the project
- Set up a Spring Boot project structure
- Edit the core configuration contained in `pom.xml`
- Create the required models with getters and setters
- Create a controller that uses the models and handles HTTP requests for CRUD operations
- Build and run the Spring Boot application
- Test the endpoints with POSTman

Prerequisites (optional)

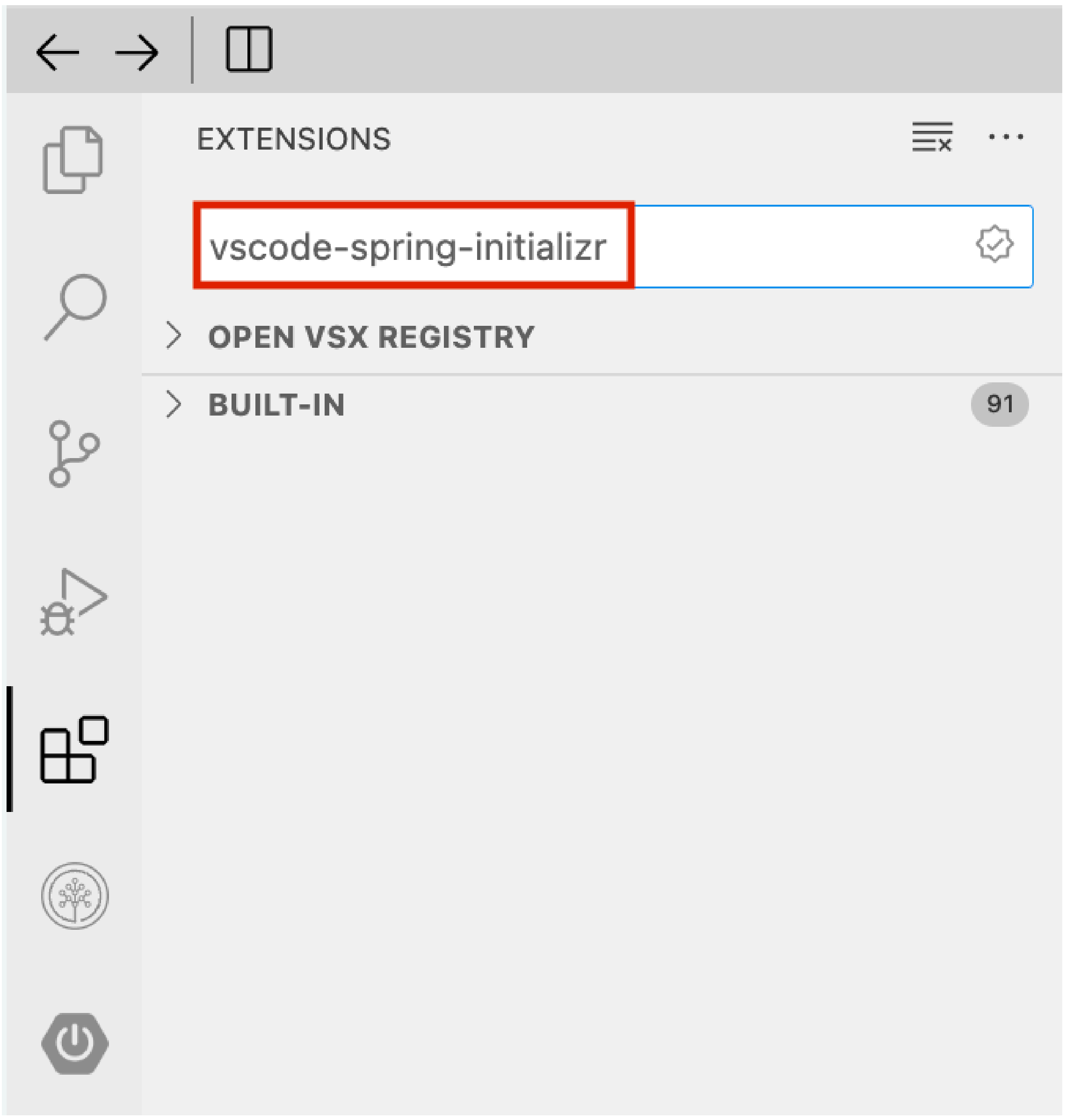
You should know basic Java programming before you get started with this lab. Please ensure that you complete the labs sequentially as they are constructed progressively.

Set up Cloud IDE for Maven Project

1. Click the extensions icon to start installing the extensions.



2. In the search bar for the extensions, type `vscode-spring-initializr`.

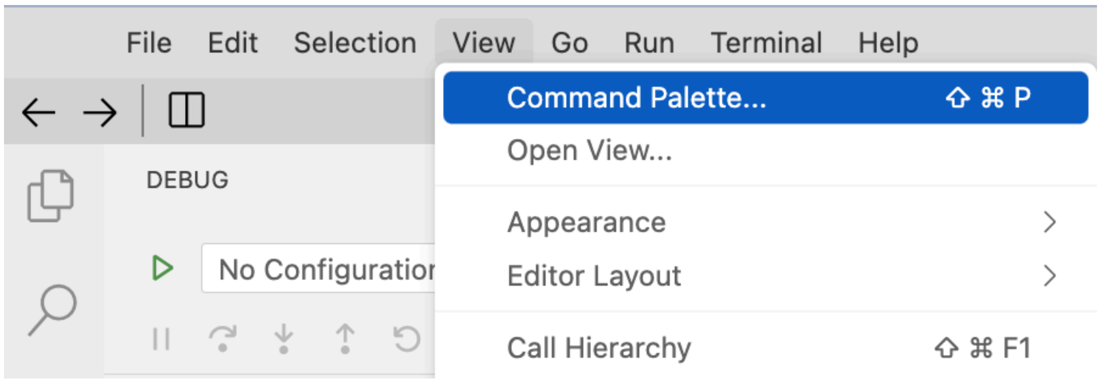


This will list all the extensions you will need for Spring Initializr.

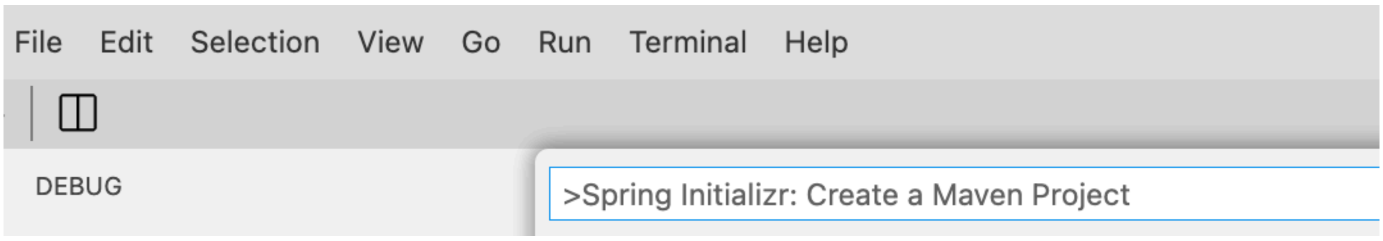
3. Install the `Spring Initializr Java Support` extension.

Create Maven Project

1. Open command palette.

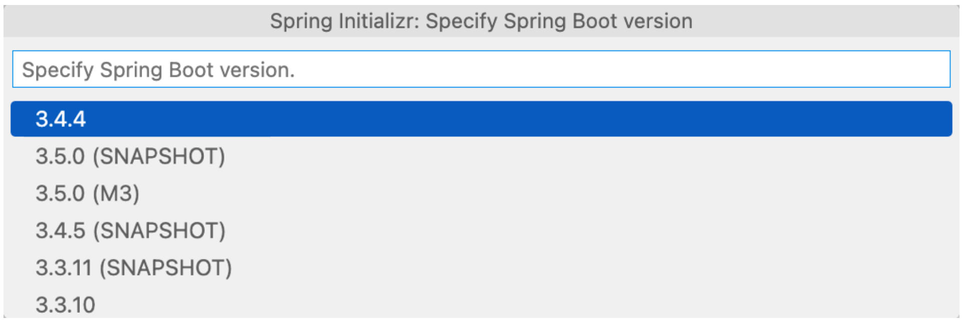


2. In the command palette text entry box, type `Spring Initializr: Create a Maven Project`.



>Spring Initializr: Create a Maven Project

3. Specify the Spring Boot version `3.4.4`.

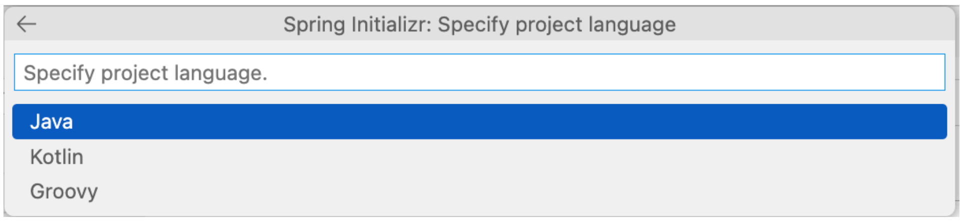


Spring Initializr: Specify Spring Boot version

Specify Spring Boot version.

- 3.4.4
- 3.5.0 (SNAPSHOT)
- 3.5.0 (M3)
- 3.4.5 (SNAPSHOT)
- 3.3.11 (SNAPSHOT)
- 3.3.10

4. Select the language you want to use, `Java`, in your case.

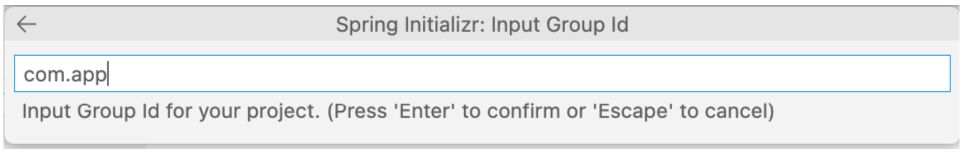


Spring Initializr: Specify project language

Specify project language.

- Java
- Kotlin
- Groovy

5. Specify the group id, such as `com.app`.

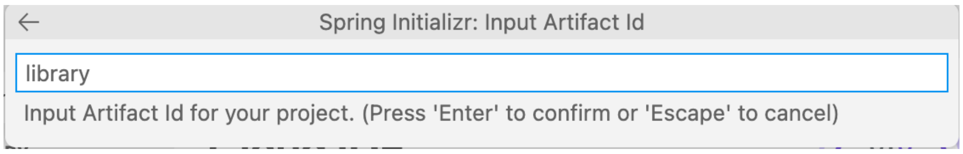


Spring Initializr: Input Group Id

com.app

Input Group Id for your project. (Press 'Enter' to confirm or 'Escape' to cancel)

6. Specify the artifact id, such as `library`.

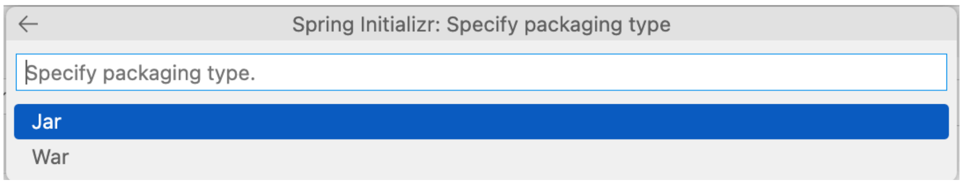


Spring Initializr: Input Artifact Id

library

Input Artifact Id for your project. (Press 'Enter' to confirm or 'Escape' to cancel)

7. Specify the packaging you want to use as `Jar` (for Java archives).

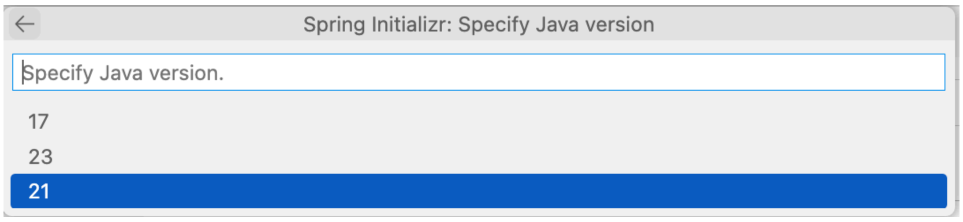


Spring Initializr: Specify packaging type

Specify packaging type.

- Jar
- War

8. Specify the Java version you want to use, `21` in the case of the IDE.

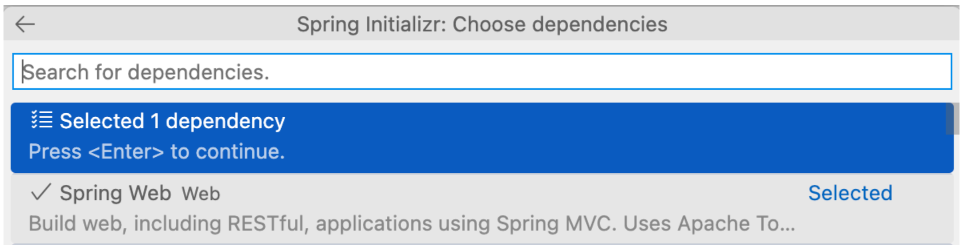


Spring Initializr: Specify Java version

Specify Java version.

- 17
- 23
- 21

9. You will be using `Spring Web` dependency for the project. Press enter to continue.

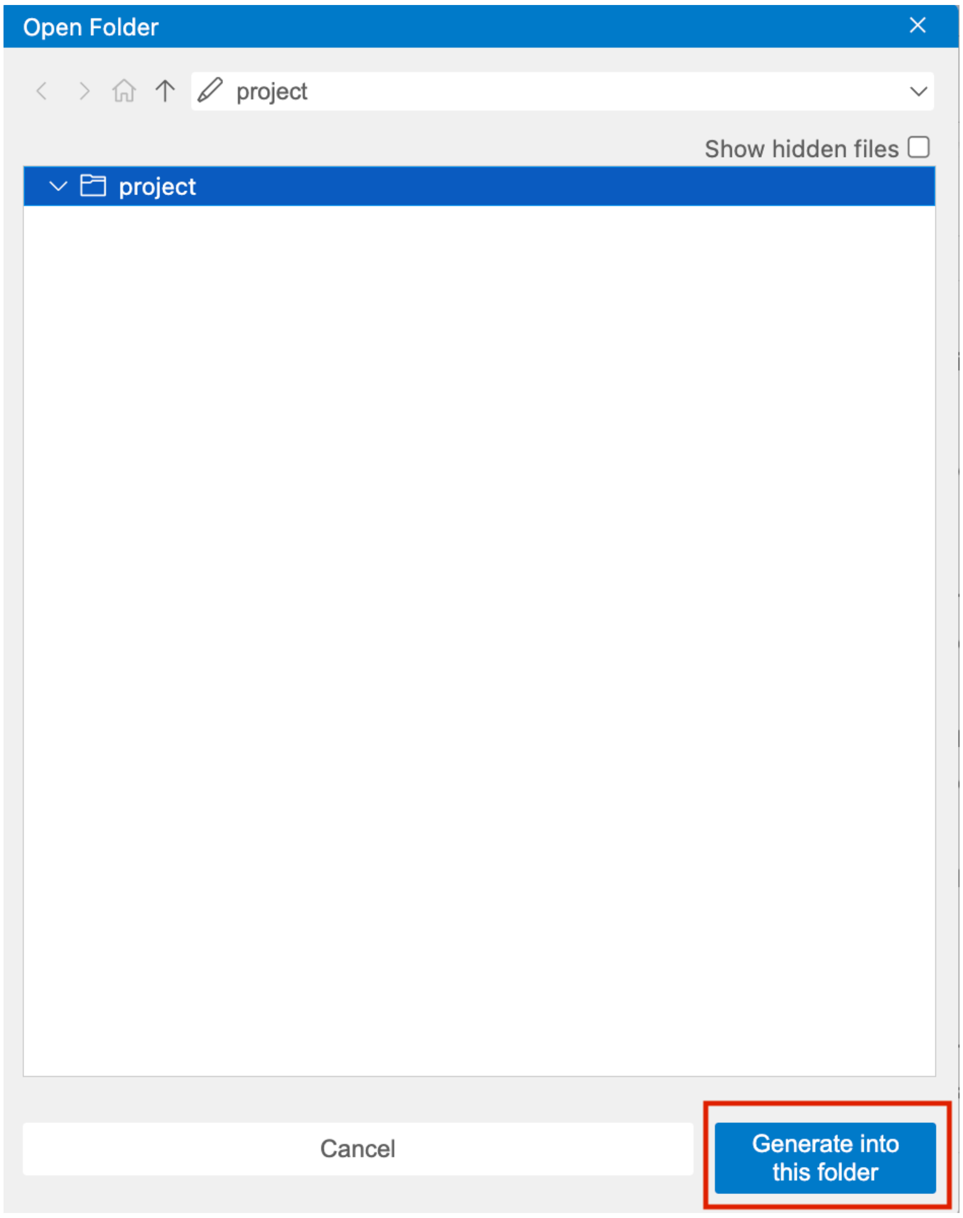

Spring Initializr: Choose dependencies

Search for dependencies.

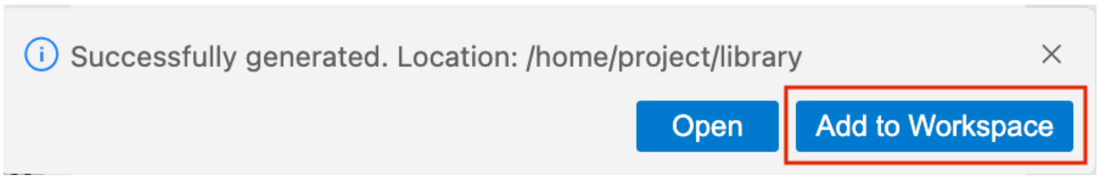
Selected 1 dependency
Press <Enter> to continue.

✓ Spring Web Web Selected
Build web, including RESTful, applications using Spring MVC. Uses Apache To...

10. The project is generated by default in the `/home/project` folder. Click `Generate` to generate the project in the desired folder.



11. Add the project to the workspace.


Successfully generated. Location: /home/project/library

Configure project and add code

1. Click the button below to open `pom.xml` and edit the project settings.

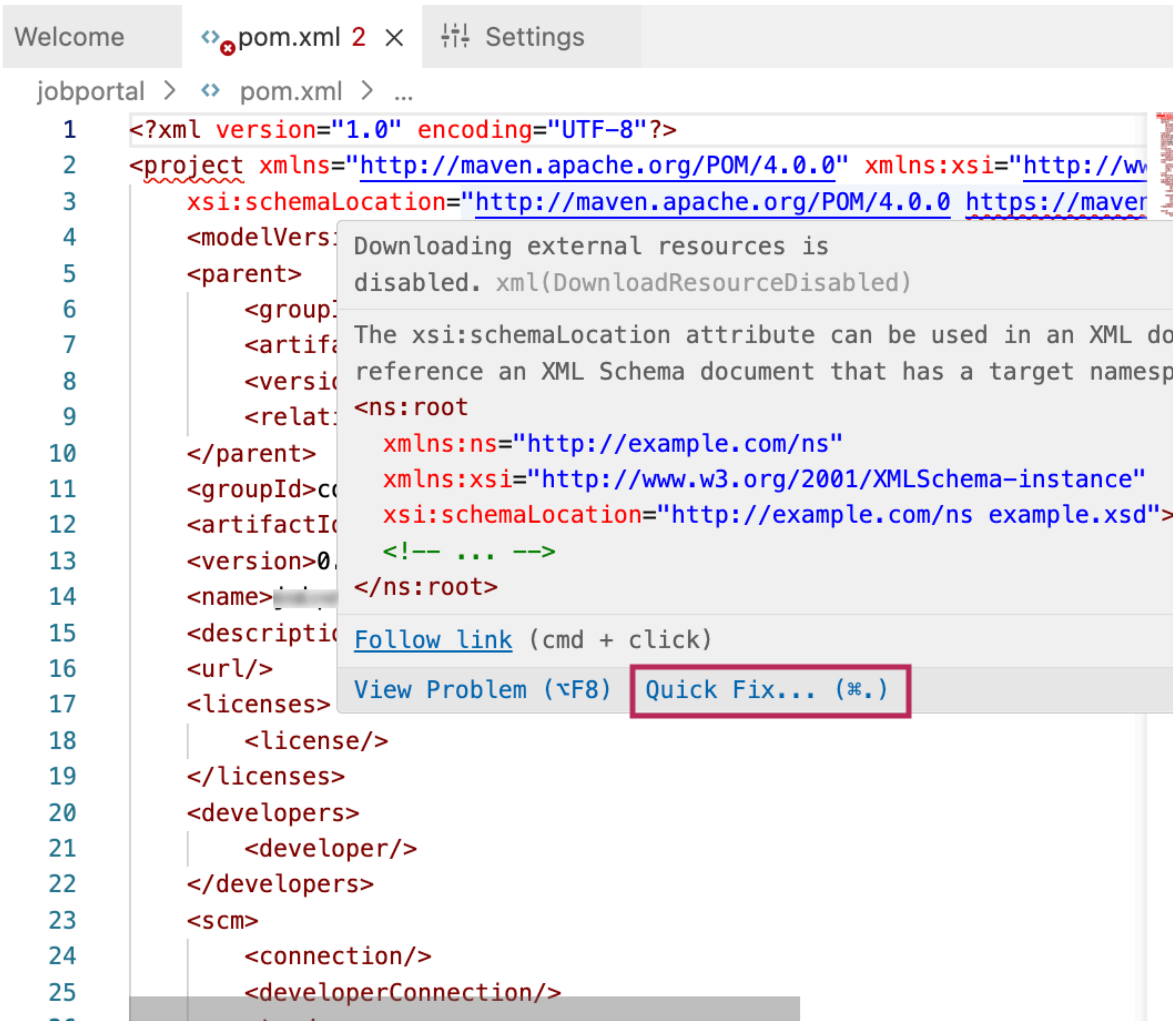
Open **pom.xml** in IDE

Ensure that the required dependencies are included.

- spring-boot-starter-web
- spring-boot-starter-test
- spring-boot-maven-plugin

With these, you get a framework for building web applications using the Model-View-Controller (MVC) pattern with an embedded Tomcat server, which allows you to run your web application without the need for a separate application server.

2. Hover over the highlighted links on `pom.xml` and click. A suggestion comes up. Click on `Quick Fix`.

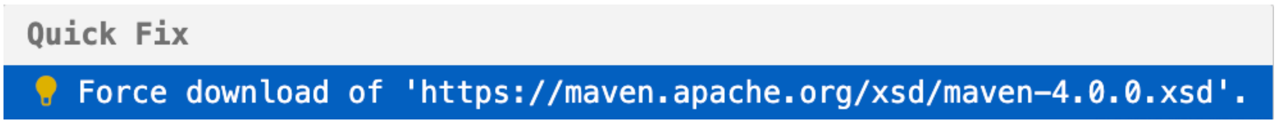


```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://ww
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maver
    <modelVers
    <parent>
        <group
        <artifa
        <versic
        <relat
    </parent>
    <groupId>c
    <artifactI
    <version>0.
    <name>[redacted]
    <descripti
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
```

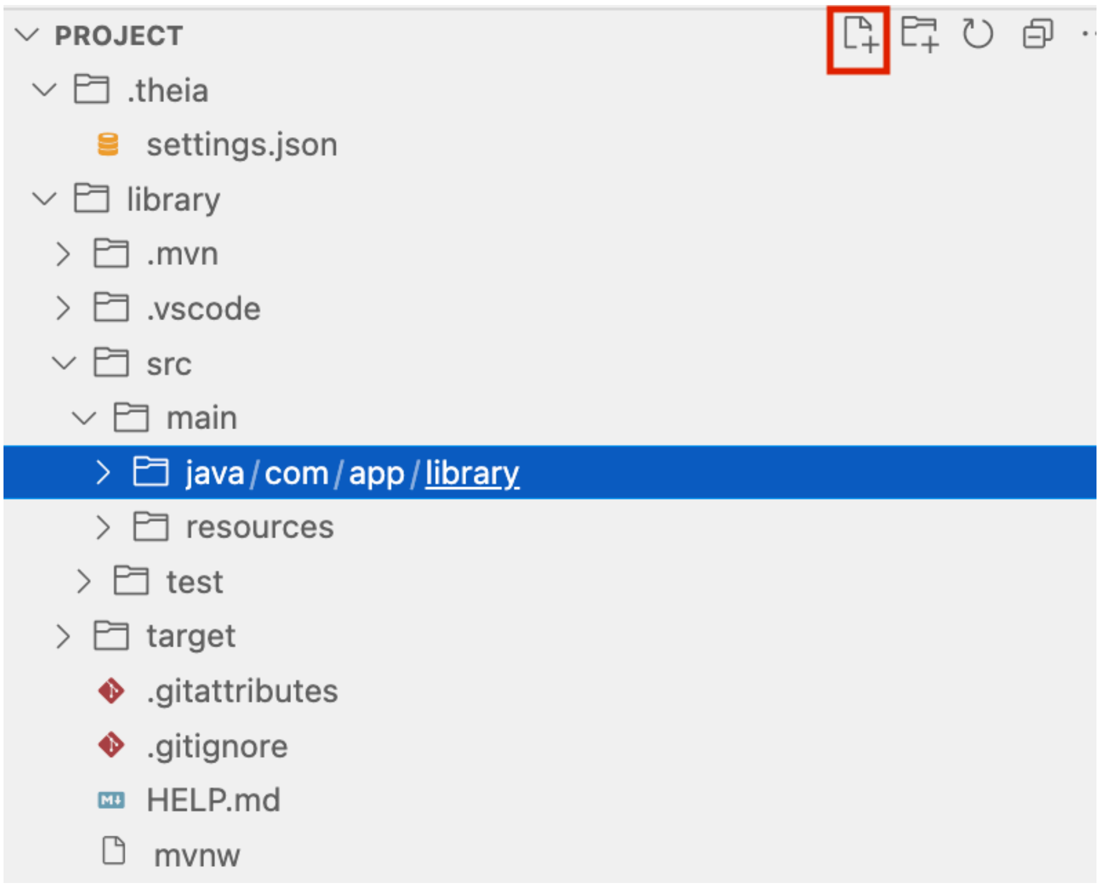
```
Downloading external resources is
disabled. xml(DownloadResourceDisabled)

The xsi:schemaLocation attribute can be used in an XML do
reference an XML Schema document that has a target namesp
<ns:root
  xmlns:ns="http://example.com/ns"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://example.com/ns example.xsd">
  <!-- ... -->
</ns:root>

Follow link (cmd + click)

View Problem (⌥F8)  Quick Fix... (⌘.)
```

3. Click to download the `xsd` file.

The xsd (XML Schema Definition) file in a pom.xml serves as a schema validator that defines the structure, syntax, and data types allowed in the Maven Project Object Model (POM) file.


Quick Fix
Force download of 'https://maven.apache.org/xsd/maven-4.0.0.xsd'.

4. Create a new file named `LibraryController.java` in the `com/app/library` folder.



5. Add the following code in `LibraryController.java` to create the `LibraryController` class.

```
package com.app.library;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class LibraryController {
    @GetMapping("/")
    public String listBooks() {
        return "<h1>Here are the list of books</h1>";
    }
}
```

6. Change to the project directory.

```
cd /home/project/library
```

7. Use `mvn clean` (maven clean) to clean any existing class files and and `mvn install` (maven install) to compile the files in the project directory and generate the runnable jar file.

```
mvn clean install
```

8. Run the following command to start the application. It will start in port `8080`.

```
mvn exec:java -Dexec.mainClass="com.app.library.LibraryApplication"
```

The server will start in port 8080.

9. Click the button below to open the browser page and access the end point.

Library Application

You should see a page with the message "Here are the list of books."

10. Press `Ctrl+C` to stop the server.

Add all the models

1. Create file `model/Book.java` (Book.java under model directory) in `src/java/com/app/library`, which contains the Book Model in `com.app.library.model` package. This is to maintain the book details and will have attributes such as:

- book id
- title
- author
- publication year
- genre
- availableCopies

Copy the code below into `Book.java`.

```
package com.app.library.model;
public class Book {
    private Long id;
    private String title;
    private String author;
    private int publicationYear;
    private String genre;
    private int availableCopies;
    // Default constructor
    public Book() {}
    // Parameterized constructor
    public Book(String title, String author, int publicationYear, String genre, int availableCopies) {
        this.title = title;
        this.author = author;
        this.publicationYear = publicationYear;
        this.genre = genre;
        this.availableCopies = availableCopies;
    }
    // Getters and Setters
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
    public int getPublicationYear() {
        return publicationYear;
    }
    public void setPublicationYear(int publicationYear) {
        this.publicationYear = publicationYear;
    }
    public String getGenre() {
        return genre;
    }
    public void setGenre(String genre) {
        this.genre = genre;
    }
    public int getAvailableCopies() {
        return availableCopies;
    }
    public void setAvailableCopies(int availableCopies) {
        this.availableCopies = availableCopies;
    }
    // toString
    @Override
    public String toString() {
        return "Book{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", author='" + author + '\'' +
                ", publicationYear=" + publicationYear +
                ", genre='" + genre + '\'' +
                ", availableCopies=" + availableCopies +
                '}';
    }
}
```

2. Create a file named `Member.java` in `src/java/com/app/library/model`, which contains the Member Model in `com.app.library.model` package. This is to maintain the member details and will have attributes such as:

- member id
- name
- email
- phone number
- membershipStartDate
- membershipEdndate

Copy the code below into `Member.java`.

```
package com.app.library.model;
import java.time.LocalDate;
public class Member {
    private Long id;
    private String name;
    private String email;
    private String phoneNumber;
    private LocalDate startDate;
    private LocalDate endDate;
    // Default constructor
    public Member() {}
    // Parameterized constructor
    public Member(String name, String email, String phoneNumber, LocalDate startDate, LocalDate endDate) {
        this.name = name;
        this.email = email;
        this.phoneNumber = phoneNumber;
        this.startDate = startDate;
        this.endDate = endDate;
    }
    // Getters and Setters
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPhoneNumber() {
        return phoneNumber;
    }
    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
    public LocalDate getStartDate() {
        return startDate;
    }
    public void setStartDate(LocalDate startDate) {
        this.startDate = startDate;
    }
    public LocalDate getEndDate() {
        return endDate;
    }
    public void setEndDate(LocalDate endDate) {
        this.endDate = endDate;
    }
    // toString
    @Override
    public String toString() {
        return "Member{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", phoneNumber='" + phoneNumber + '\'' +
                ", membershipStartDate=" + startDate +
                ", membershipEndDate=" + endDate +
                '}';
    }
}
```

3. Create a file named `BorrowingRecord.java` in `src/java/com/app/library/model`, which contains the BorrowingRecord Model in `com.app.library.model` package. This is to maintain the borrowing record details and will have attributes such as:

- id
- book
- member
- borrowDate
- returnDate
- dueDate

Copy the code below into `BorrowingRecord.java`.

```
package com.app.library.model;
import java.time.LocalDate;
public class BorrowingRecord {
    private Long id;
    private Book book;
    private Member member;
    private LocalDate borrowDate;
    private LocalDate returnDate;
    private LocalDate dueDate;
    // Default constructor
    public BorrowingRecord() {}
    // Parameterized constructor
    public BorrowingRecord(Book book, Member member, LocalDate borrowDate, LocalDate dueDate) {
        this.book = book;
        this.member = member;
        this.borrowDate = borrowDate;
        this.dueDate = dueDate;
    }
    // Getters and Setters
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public Book getBook() {
        return book;
    }
    public void setBook(Book book) {
        this.book = book;
    }
    public Member getMember() {
        return member;
    }
    public void setMember(Member member) {
        this.member = member;
    }
    public LocalDate getBorrowDate() {
        return borrowDate;
    }
    public void setBorrowDate(LocalDate borrowDate) {
        this.borrowDate = borrowDate;
    }
    public LocalDate getReturnDate() {
        return returnDate;
    }
    public void setReturnDate(LocalDate returnDate) {
        this.returnDate = returnDate;
    }
    public LocalDate getDueDate() {
        return dueDate;
    }
    public void setDueDate(LocalDate dueDate) {
        this.dueDate = dueDate;
    }
    // toString
    @Override
    public String toString() {
        return "BorrowingRecord{" +
                "id=" + id +
                ", book=" + book +
                ", member=" + member +
                ", borrowDate=" + borrowDate +
                ", returnDate=" + returnDate +
                ", dueDate=" + dueDate +
                '}';
    }
}
```

Create the class with all the services

1. Create file `service/LibraryService.java` (LibraryService.java in `service` directory) in `src/java/com/app/library`, which contains the LibraryService class in `com.app.library.service` package. This is to provide service for all the REST APIs using the models as listed below:

Books

- Get all books
- Get a book by ID
- Add a new book
- Update a book
- Delete a book by ID

Members

- Get all members
- Get a member by ID
- Add a new member
- Update a member
- Delete a member by ID

BorrowingRecords

- Get all borrowing records
- Borrow a book (create a new borrowing record)
- Return a book (update the borrowing record with the return date)

Copy the code below into `LibraryService.java`.

```
package com.app.library.service;
import com.app.library.model.Book;
import com.app.library.model.Member;
import com.app.library.model.BorrowingRecord;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import org.springframework.stereotype.Service;
@Service
public class LibraryService {
    // In-memory storage using ArrayList
    private List<Book> books = new ArrayList<>();
    private List<Member> members = new ArrayList<>();
    private List<BorrowingRecord> borrowingRecords = new ArrayList<>();
    // ==================== Book Methods ====================
    // Get all books
    public List<Book> getAllBooks() {
        return books;
    }
    // Get a book by ID
    public Optional<Book> getBookById(Long id) {
        return books.stream()
                .filter(book -> book.getId().equals(id))
                .findFirst();
    }
    // Add a new book
    public void addBook(Book book) {
        books.add(book);
    }
    // Update a book
    public void updateBook(Book updatedBook) {
        for (int i = 0; i < books.size(); i++) {
            Book book = books.get(i);
            if (book.getId().equals(updatedBook.getId())) {
                books.set(i, updatedBook);
                break;
            }
        }
    }
    // Delete a book by ID
    public void deleteBook(Long id) {
        books.removeIf(book -> book.getId().equals(id));
    }
    // ==================== Member Methods ====================
    // Get all members
    public List<Member> getAllMembers() {
        return members;
    }
    // Get a member by ID
    public Optional<Member> getMemberById(Long id) {
        return members.stream()
                .filter(member -> member.getId().equals(id))
                .findFirst();
    }
    // Add a new member
    public void addMember(Member member) {
        members.add(member);
    }
    // Update a member
    public void updateMember(Member updatedMember) {
        for (int i = 0; i < members.size(); i++) {
            Member member = members.get(i);
            if (member.getId().equals(updatedMember.getId())) {
                members.set(i, updatedMember);
                break;
            }
        }
    }
    // Delete a member by ID
    public void deleteMember(Long id) {
        members.removeIf(member -> member.getId().equals(id));
    }
    // ==================== BorrowingRecord Methods ====================
    // Get all borrowing records
    public List<BorrowingRecord> getAllBorrowingRecords() {
        return borrowingRecords;
    }
    // Borrow a book (create a new borrowing record)
    public void borrowBook(BorrowingRecord record) {
        // Set borrow date and due date (e.g., due date = borrow date + 14 days)
        record.setBorrowDate(LocalDate.now());
        record.setDueDate(LocalDate.now().plusDays(14));
        borrowingRecords.add(record);
        // Decrease the available copies of the book
        Book book = record.getBook();
        book.setAvailableCopies(book.getAvailableCopies() - 1);
    }
    // Return a book (update the borrowing record with the return date)
    public void returnBook(Long recordId, LocalDate returnDate) {
        for (BorrowingRecord record : borrowingRecords) {
            if (record.getId().equals(recordId)) {
                record.setReturnDate(returnDate);
                // Increase the available copies of the book
                Book book = record.getBook();
                book.setAvailableCopies(book.getAvailableCopies() + 1);
                break;
            }
        }
    }
}
```

Modify the code in the controller and run the app

1. Open `LibraryController.java` and replace the content with the following code. This will provide the endpoint for retrieving the jobs list. This code has an endpoint for CRUD (Creation, Retrieval, Updation, and Deletion) for all the three models Books, Members, and BorrowingRecord required for library management and uses the services defined in `LibraryService.java`. The `LibraryController` uses sl4j, which is imported as part of the Spring Boot app for logging.

```
package com.app.library.controller;
import com.app.library.model.Book;
import com.app.library.model.Member;
import com.app.library.model.BorrowingRecord;
import com.app.library.service.LibraryService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDate;
import java.util.List;
import java.util.Optional;
@RestController
@RequestMapping("/api")
public class LibraryController {
    // Create a logger instance
    private static final Logger logger = LoggerFactory.getLogger(LibraryController.class);
    @Autowired
    private LibraryService libraryService;
    // ==================== Book Endpoints ====================
    // Get all books
    @GetMapping("/books")
    public ResponseEntity<List<Book>> getAllBooks() {
        List<Book> books = libraryService.getAllBooks();
        logger.info("The list of books returned"+books);
        return new ResponseEntity<>(books, HttpStatus.OK);
    }
    // Get a book by ID
    @GetMapping("/books/{id}")
    public ResponseEntity<Book> getBookById(@PathVariable Long id) {
        Optional<Book> book = libraryService.getBookById(id);
        logger.info("The book returned"+book);
        return book.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
    // Add a new book
    @PostMapping("/books")
    public ResponseEntity<Book> addBook(@RequestBody Book book) {
        libraryService.addBook(book);
        logger.info("The book was added");
        return new ResponseEntity<>(book, HttpStatus.CREATED);
    }
    // Update a book
    @PutMapping("/books/{id}")
    public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {
        if (!libraryService.getBookById(id).isPresent()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        updatedBook.setId(id);
        libraryService.updateBook(updatedBook);
        logger.info("The book has been updated "+updatedBook);
        return new ResponseEntity<>(updatedBook, HttpStatus.OK);
    }
    // Delete a book
    @DeleteMapping("/books/{id}")
    public ResponseEntity<Void> deleteBook(@PathVariable Long id) {
        if (!libraryService.getBookById(id).isPresent()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        libraryService.deleteBook(id);
        logger.info("The book has been deleted ");
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    // ==================== Member Endpoints ====================
    // Get all members
    @GetMapping("/members")
    public ResponseEntity<List<Member>> getAllMembers() {
        List<Member> members = libraryService.getAllMembers();
        logger.info("The members in the system " + members);
        return new ResponseEntity<>(members, HttpStatus.OK);
    }
    // Get a member by ID
    @GetMapping("/members/{id}")
    public ResponseEntity<Member> getMemberById(@PathVariable Long id) {
        Optional<Member> member = libraryService.getMemberById(id);
        logger.info("The member you retrieved "+member);
        return member.map(value -> new ResponseEntity<>(value, HttpStatus.OK))
                .orElseGet(() -> new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
    // Add a new member
    @PostMapping("/members")
    public ResponseEntity<Member> addMember(@RequestBody Member member) {
        libraryService.addMember(member);
        logger.info("The member has been added ");
        return new ResponseEntity<>(member, HttpStatus.CREATED);
    }
    // Update a member
    @PutMapping("/members/{id}")
    public ResponseEntity<Member> updateMember(@PathVariable Long id, @RequestBody Member updatedMember) {
        if (!libraryService.getMemberById(id).isPresent()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        updatedMember.setId(id);
        libraryService.updateMember(updatedMember);
        logger.info("The member has been updated "+updatedMember);
        return new ResponseEntity<>(updatedMember, HttpStatus.OK);
    }
    // Delete a member
    @DeleteMapping("/members/{id}")
    public ResponseEntity<Void> deleteMember(@PathVariable Long id) {
        if (!libraryService.getMemberById(id).isPresent()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        libraryService.deleteMember(id);
        logger.info("The member has been deleted "+id);
        return new ResponseEntity<>(HttpStatus.NO_CONTENT);
    }
    // ==================== BorrowingRecord Endpoints ====================
    // Get all borrowing records
    @GetMapping("/borrowing-records")
    public ResponseEntity<List<BorrowingRecord>> getAllBorrowingRecords() {
        List<BorrowingRecord> records = libraryService.getAllBorrowingRecords();
        logger.info("The records has been retrieved "+records);
        return new ResponseEntity<>(records, HttpStatus.OK);
    }
    // Borrow a book
    @PostMapping("/borrow")
    public ResponseEntity<BorrowingRecord> borrowBook(@RequestBody BorrowingRecord record) {
        // Set borrow date and due date (e.g., due date = borrow date + 14 days)
        record.setBorrowDate(LocalDate.now());
        record.setDueDate(LocalDate.now().plusDays(14));
        libraryService.borrowBook(record);
        logger.info("The book has been borrowed "+record);
        return new ResponseEntity<>(record, HttpStatus.CREATED);
    }
    // Return a book
    @PutMapping("/return/{recordId}")
    public ResponseEntity<Void> returnBook(@PathVariable Long recordId) {
        libraryService.returnBook(recordId, LocalDate.now());
        logger.info("The book has been retrieved "+recordId);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

3. Use `mvn clean` (maven clean) to clean any existing class files and and `mvn install` (maven install) to compile the files in the project directory and generate the runnable jar file.

```
mvn clean install
```

4. Run the following command to start the application. It will start in port `8080`.

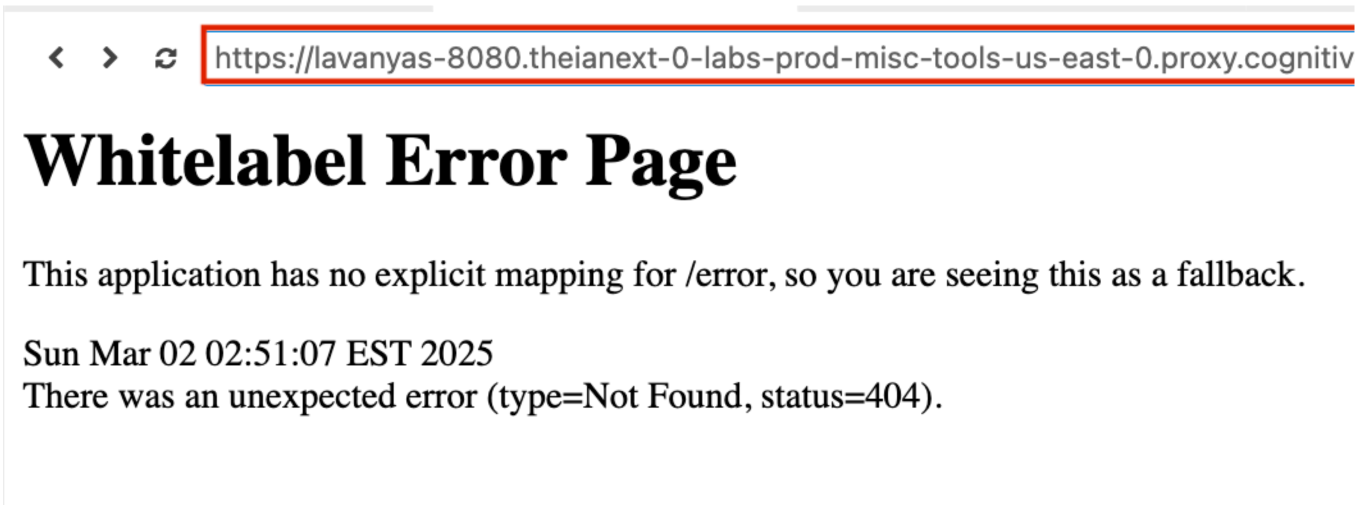
```
mvn exec:java -Dexec.mainClass="com.app.library.LibraryApplication"
```

The server will start in port 8080.

5. Click the button below to open the app.

Library Application

6. Copy the URL and note it down in a notepad or other editor. You will use this URL in POSTman.



https://lavanyas-8080.theianext-0-labs-prod-misc-tools-us-east-0.proxy.cognitiv

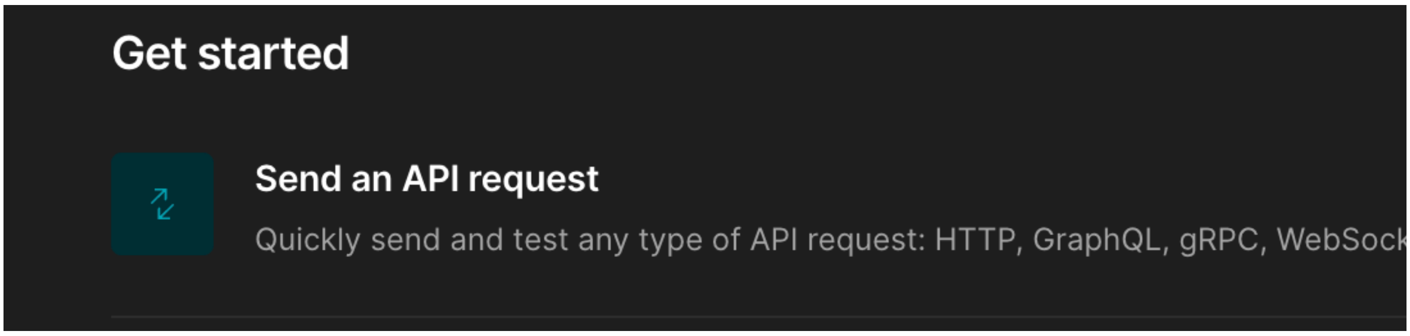
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Mar 02 02:51:07 EST 2025
There was an unexpected error (type=Not Found, status=404).

Use POSTman to explore the REST APIs for CRUD

1. Go to www.postman.com and select `Sign Up for Free` to create a new login, or sign in with your Google Mail credentials to log in.
2. Once you complete signing up, on the POSTman homepage, select `New Request` to start sending request to your request to the Library app's REST APIs.



Get started

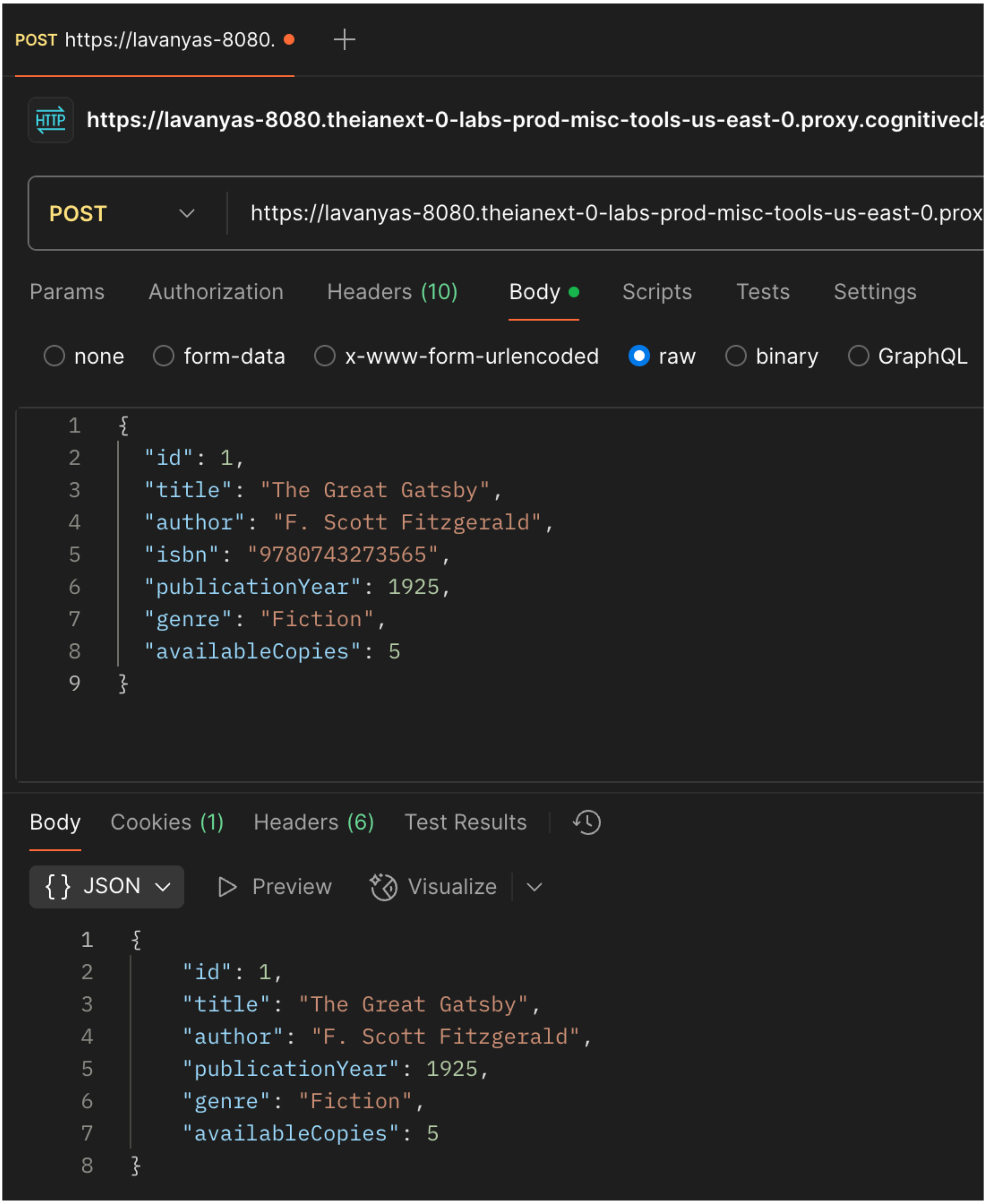
Send an API request

Quickly send and test any type of API request: HTTP, GraphQL, gRPC, WebSock

3. In the request page, set the request option to `POST`. Type the URL that you had copied and suffix it with `/books`. Choose the `Body` option for input and configure the input to be `raw,json`. Paste a JSON, like the option provided below. Click `Send` to post the book.

```
{
    "id": 1,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "publicationYear": 1925,
    "genre": "Fiction",
    "availableCopies": 5
}
```

If the request goes through, you will see a response, as shown in the image.


POST https://lavanyas-8080.

https://lavanyas-8080.theianext-0-labs-prod-misc-tools-us-east-0.proxy.cognitivecl

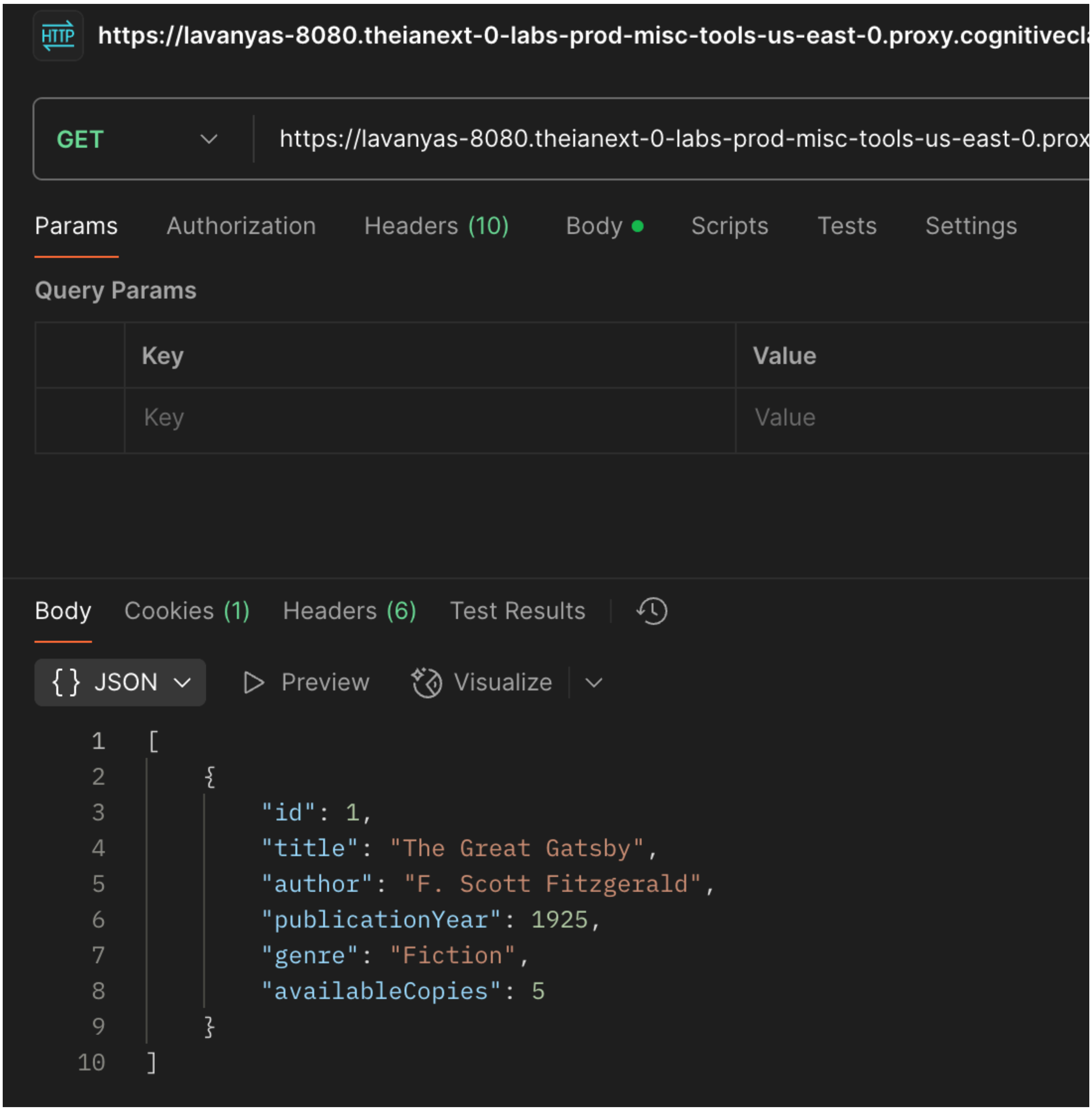
POST https://lavanyas-8080.theianext-0-labs-prod-misc-tools-us-east-0.prox

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL

```
{
  "id": 1,
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "isbn": "9780743273565",
  "publicationYear": 1925,
  "genre": "Fiction",
  "availableCopies": 5
}
```

```
{
    "id": 1,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "publicationYear": 1925,
    "genre": "Fiction",
    "availableCopies": 5
}
```

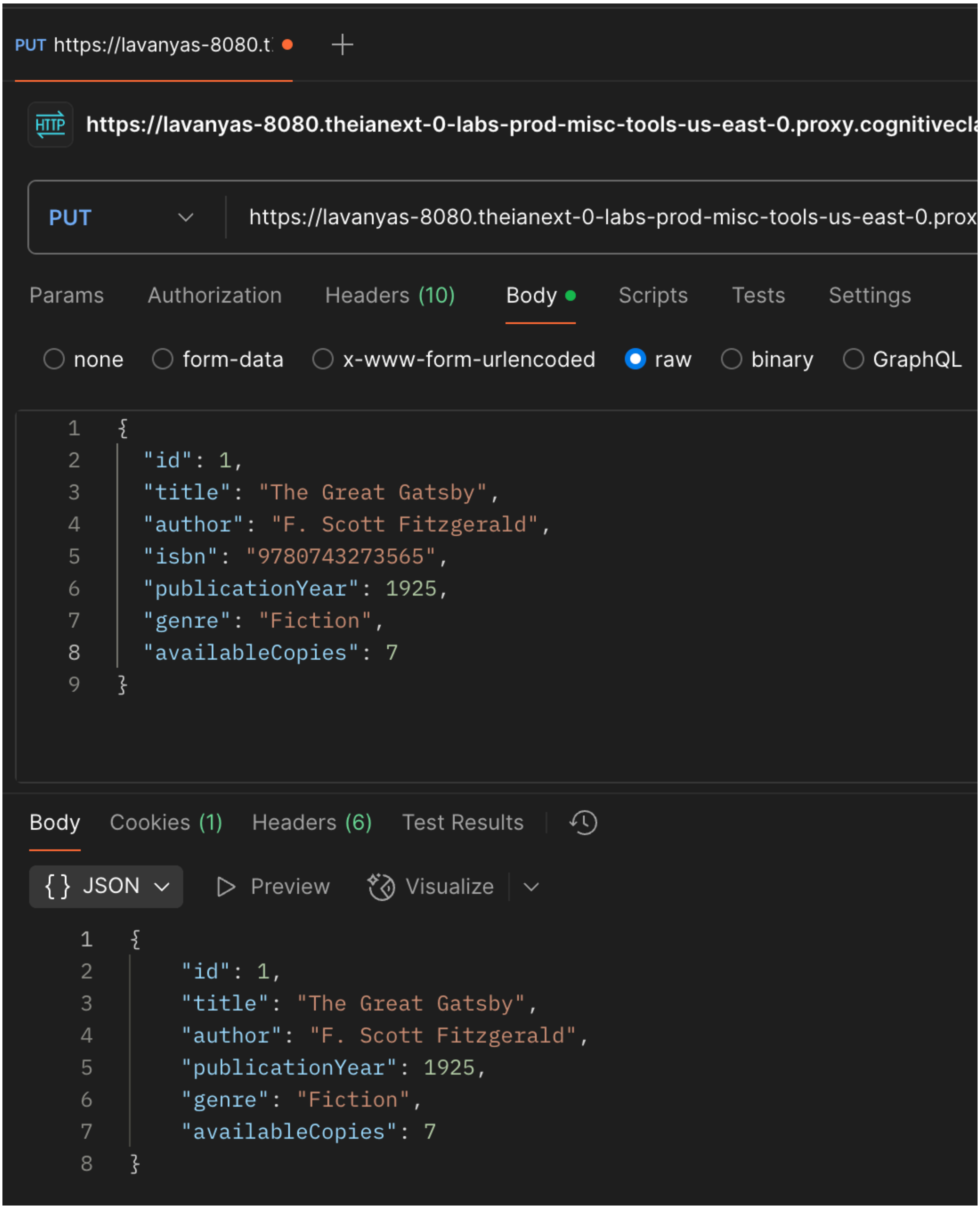
4. Change the request type to `GET` and see if the book that you just posted is retrieved.



```
[
    {
        "id": 1,
        "title": "The Great Gatsby",
        "author": "F. Scott Fitzgerald",
        "publicationYear": 1925,
        "genre": "Fiction",
        "availableCopies": 5
    }
]
```

5. Change the request type to `PUT`, append `/1` to the end of the URL, and change the `availableCopies` to 7. This will update the available copies of the book with id `1` to `7`.

```
"id": 1,
"title": "The Great Gatsby",
"author": "F. Scott Fitzgerald",
"publicationYear": 1925,
"genre": "Fiction",
"availableCopies": 7
```



PUT https://lavanyas-8080.theianext-0-labs-prod-misc-tools-us-east-0.prox

Body: [✓ raw]

```
{
  "id": 1,
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "isbn": "9780743273565",
  "publicationYear": 1925,
  "genre": "Fiction",
  "availableCopies": 7
}
```

```
{
    "id": 1,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "publicationYear": 1925,
    "genre": "Fiction",
    "availableCopies": 7
}
```

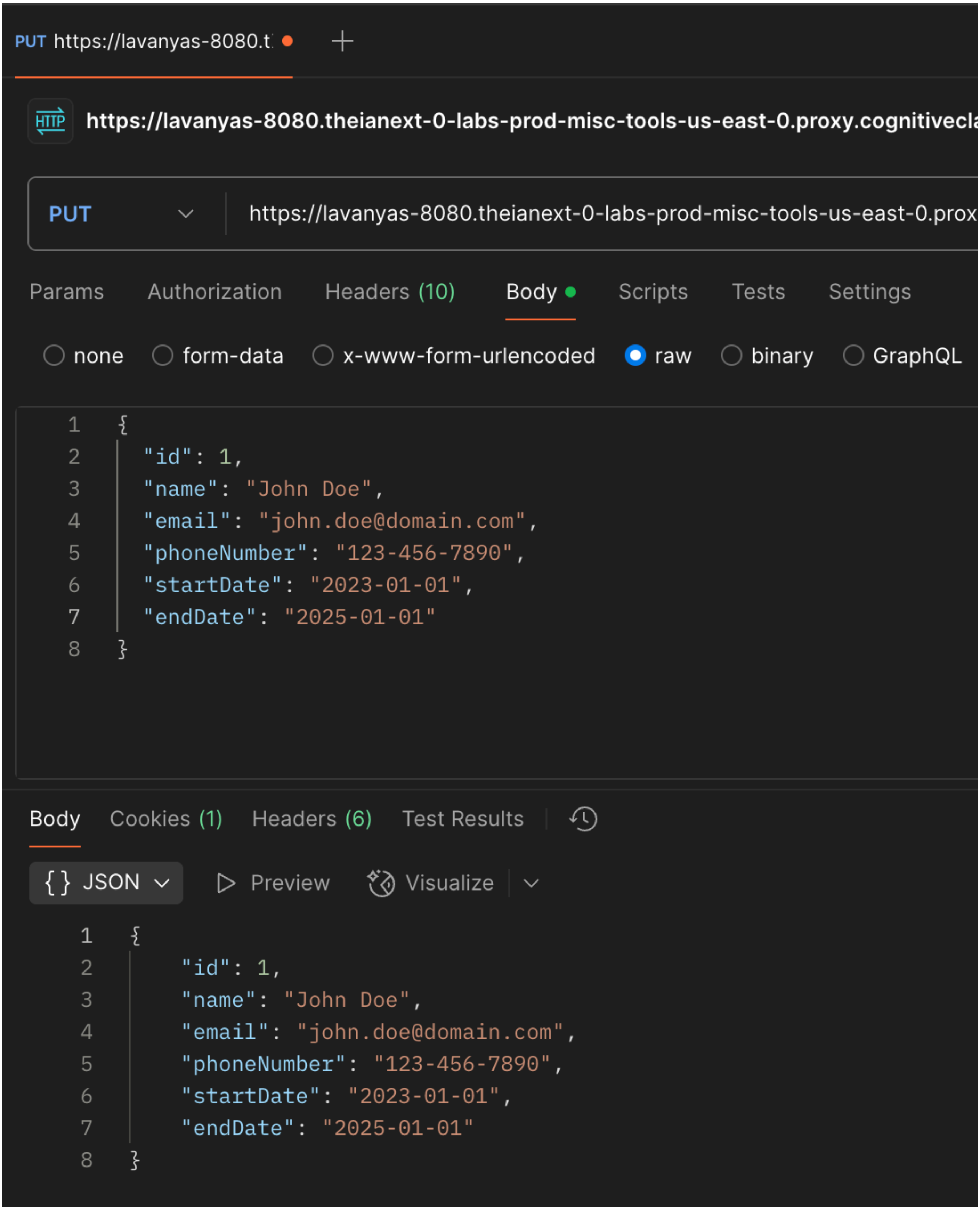
You can send a `GET` request to check if the changes are made.

6. In the request page, set the request option to `POST`. Type the URL that you had copied and suffix it with `/members`. Choose the `Body` option for input and configure the input to be `raw,json`. Paste a JSON, like the option provided below. Click `Send` to post the member.

```
{
    "id": 1,
    "name": "John Doe",
    "email": "john.doe@example.com",
    "phoneNumber": "123-456-7890",
    "startDate": "2023-01-01",
    "endDate": "2025-01-01"
  }
```

This will add the member to the Library management system.

7. You can update the member by changing the request to 'PUT', suffix the request URL with the member id that you want to change, and then add the JSON.



PUT https://lavanyas-8080.t

https://lavanyas-8080.theianext-0-labs-prod-misc-tools-us-east-0.proxy.cognitivecl

PUT https://lavanyas-8080.theianext-0-labs-prod-misc-tools-us-east-0.prox

Body: raw

```
{
  "id": 1,
  "name": "John Doe",
  "email": "john.doe@domain.com",
  "phoneNumber": "123-456-7890",
  "startDate": "2023-01-01",
  "endDate": "2025-01-01"
}
```

```
{
    "id": 1,
    "name": "John Doe",
    "email": "john.doe@domain.com",
    "phoneNumber": "123-456-7890",
    "startDate": "2023-01-01",
    "endDate": "2025-01-01"
}
```

8. You can delete a book by suffixing the default URL with the book followed by the id, such as `/books/1`, and delete a member by suffixing the default URL with the book followed by the id, such as `/books/1`.

9. You can add a `BorrowingRecord` by changing the request to `POST` and the suffix to the default URL, `/borrow`. For example, your endpoint will be as follows:

`https://user-8080.theianext-0-labs-prod-misc-tools-us-east-0.proxy.cognitiveclass.ai/api/borrow.`

A borrowing record will have the record id, the member details, and the book details. The date of borrowing will be the current date, and the due date will be calculated intrinsically. A sample borrowing record has been provided.

```
{
  "id": 1,
  "book": {
    "id": 1,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "publicationYear": 1925,
    "genre": "Fiction",
    "availableCopies": 7
  },
  "member": {
    "id": 1,
    "name": "John Doe",
    "email": "john.doe@example.com",
    "phoneNumber": "123-456-7890",
    "startDate": "2023-01-01",
    "endDate": "2025-01-01"
  }
}
```

Practice exercise

1. Test all the endpoints that you created.

2. Add an endpoint where you can search books by title.

3. Add an endpoint where you can search books by the author.

Conclusion

In this lab, you have:

- Used Spring Initializr to generate the Spring Boot application with REST API
- Set up a Spring Boot project structure
- Created the required models with getters and setters
- Created a service class that is autowired (created automatically when the controller is accessed) to handle all the functionalities
- Created a controller that uses the models and handles HTTP requests for CRUD operations
- Ran the Spring Boot application and tested the endpoints with POSTman

Author(s)

Lavanya