can you summerize steps of video <https://www.youtube.com/watch?v=J0aEfUUervE&t=1314s>

Here are the steps to deploy a Spring Boot Serverless CRUD API to AWS Lambda using API Gateway, based on the video:

**Prerequisites**

* Basic understanding of Spring Boot and AWS
* Java Development Kit (JDK) 17 or later
* An AWS account

**Steps**

1. **Create a Spring Boot project** using the Maven archetype designed for AWS Lambda deployment. This will generate a basic project structure with necessary dependencies and configurations.
2. **Develop your CRUD API** using Spring Boot controllers, services, and data models. The video uses an in-memory list for simplicity, but you can replace it with a database.
3. **Package your application** into a JAR file using Maven.
4. **Create an AWS Lambda function**
   * Go to the AWS Management Console and navigate to the Lambda service.
   * Click on "Create function".
   * Choose "Author from scratch".

[1. medium.com](https://medium.com/@tananya0310/automated-birthday-wishes-sending-joyful-surprises-with-amazon-eventbridge-amazon-sns-amazon-s3-303e81140be8" \t "_blank)

[medium.com](https://medium.com/@tananya0310/automated-birthday-wishes-sending-joyful-surprises-with-amazon-eventbridge-amazon-sns-amazon-s3-303e81140be8" \t "_blank)

* + Give your function a name (e.g., course-service).
  + Select the Java runtime (Java 17 in this case).
  + Click on "Create function".

1. **Upload your JAR file** to the Lambda function.
   * In the function's configuration, go to the "Code" section.
   * Click on "Upload a .zip or JAR file".
   * Select your JAR file and upload it.
2. **Configure the function handler**
   * In the function's configuration, go to the "Environment variables" section.
   * Add a variable named HANDLER with the value com.example.streamLambdaHandler::handleRequest (replace with the actual package and class name of your handler).
3. **Test your function** (Optional)
   * You can test your function by providing a test event and clicking on the "Test" button.
4. **Create an API Gateway**
   * Go to the AWS Management Console and navigate to the API Gateway service.
   * Click on "Create API".
   * Choose the "REST API" option.
   * Give your API a name (e.g., CourseAPI).
   * Click on "Create API".
5. **Create a resource** for your API
   * In the API Gateway console, select your API.
   * Click on "Actions" -> "Create Resource".
   * Give your resource a name (e.g., /courses).
   * Click on "Create Resource".
6. **Create a method** for your API
   * Select the resource you created.
   * Click on "Actions" -> "Create Method".
   * Choose the HTTP method (e.g., GET, POST, PUT, DELETE).
   * Select "Lambda proxy integration".
   * Choose your Lambda function from the dropdown.
   * Click on "Save".
7. **Deploy your API**
   * In the API Gateway console, select your API.
   * Click on "Actions" -> "Deploy API".
   * Choose a deployment stage (e.g., dev).
   * Click on "Deploy".
8. **Test your API**
   * You can use tools like Postman or curl to test your API by sending requests to the API endpoint generated by API Gateway.

By following these steps, you should be able to successfully deploy your Spring Boot CRUD API to AWS Lambda using API Gateway.

can you display complete java code inside above step 2 Develop your CRUD API

**Understanding the Limitations**

**Before we dive into the code, it's essential to clarify a few points:**

* **Video Content:** The specific code implementation in the video might differ from what I'll provide.
* **Data Storage:** The video likely uses an in-memory list for simplicity. For a production-ready application, you'd typically use a database.
* **Error Handling:** For brevity, error handling is omitted. In production, robust error handling is crucial.
* **Dependency Injection:** Spring Boot's dependency injection is used implicitly.

**Complete Java Code for a Basic CRUD API**

**Model (Course.java)**

Java

import java.math.BigDecimal;

public class Course {

private Long id;

private String name;

private BigDecimal price;

// Constructors, getters, and setters

}

Use code [with caution.](/faq#coding)

**Service (CourseService.java)**

Java

import java.util.ArrayList;

import java.util.List;

public class CourseService {

private static List<Course> courses = new ArrayList<>();

private static long currentId = 0;

public Course createCourse(Course course) {

course.setId(++currentId);

courses.add(course);

return course;

}

public List<Course> getAllCourses() {

return courses;

}

public Course getCourseById(Long id) {

return courses.stream().filter(c -> c.getId().equals(id)).findFirst().orElse(null);

}

public Course updateCourse(Course course) {

Course existingCourse = getCourseById(course.getId());

if (existingCourse != null) {

existingCourse.setName(course.getName());

existingCourse.setPrice(course.getPrice());

}

return existingCourse;

}

public void deleteCourse(Long id) {

courses.removeIf(c -> c.getId().equals(id));

}

}

Use code [with caution.](/faq#coding)

**Controller (CourseController.java)**

Java

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

[1. github.com](https://github.com/bigcode-project/octopack" \t "_blank)

[MIT](https://github.com/bigcode-project/octopack" \t "_blank)

[github.com](https://github.com/bigcode-project/octopack" \t "_blank)

import java.util.List;

@RestController

[1. masteringbackend.com](https://masteringbackend.com/posts/spring-boot" \t "_blank)

[masteringbackend.com](https://masteringbackend.com/posts/spring-boot" \t "_blank)

@RequestMapping("/courses")

public class CourseController {

private final CourseService courseService;

public CourseController(CourseService courseService) {

this.courseService = courseService;

[1. baeldung3.rssing.com](https://baeldung3.rssing.com/chan-58647577/all_p99.html" \t "_blank)

[baeldung3.rssing.com](https://baeldung3.rssing.com/chan-58647577/all_p99.html" \t "_blank)

}

@PostMapping

public ResponseEntity<Course> createCourse(@RequestBody Course

[1. github.com](https://github.com/dedol1/FoE-Eplatform-Backend" \t "_blank)

[github.com](https://github.com/dedol1/FoE-Eplatform-Backend" \t "_blank)

course) {

Course createdCourse = courseService.createCourse(course);

return new ResponseEntity<>(createdCourse, HttpStatus.CREATED);

[1. github.com](https://github.com/Paragmgit/ecompass" \t "_blank)

[github.com](https://github.com/Paragmgit/ecompass" \t "_blank)

}

@GetMapping

public ResponseEntity<List<Course>> getAllCourses() {

List<Course> courses = courseService.getAllCourses();

return new ResponseEntity<>(courses, HttpStatus.OK);

}

@GetMapping("/{id}")

[1. github.com](https://github.com/DiViNiZe/CloudnativMidtermExam" \t "_blank)

[github.com](https://github.com/DiViNiZe/CloudnativMidtermExam" \t "_blank)

public ResponseEntity<Course> getCourseById(@PathVariable Long id) {

Course course = courseService.getCourseById(id);

if (course != null) {

return

[1. github.com](https://github.com/Giuseppe-Andinolfi/YourSystem" \t "_blank)

[github.com](https://github.com/Giuseppe-Andinolfi/YourSystem" \t "_blank)

new ResponseEntity<>(course, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

@PutMapping

[1. github.com](https://github.com/Aline-Tumukunde/CRUD_LIST_Assignment" \t "_blank)

[github.com](https://github.com/Aline-Tumukunde/CRUD_LIST_Assignment" \t "_blank)

public ResponseEntity<Course> updateCourse(@RequestBody Course course) {

Course updatedCourse = courseService.updateCourse(course);

[1. github.com](https://github.com/LorenaMitelut/course-scheduling-app" \t "_blank)

[github.com](https://github.com/LorenaMitelut/course-scheduling-app" \t "_blank)

if (updatedCourse != null) {

return new ResponseEntity<>(updatedCourse, HttpStatus.OK);

} else {

return new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

@DeleteMapping("/{id}")

[1. github.com](https://github.com/AbhinavRayini/HiSkill" \t "_blank)

[github.com](https://github.com/AbhinavRayini/HiSkill" \t "_blank)

public ResponseEntity<Void> deleteCourse(@PathVariable Long id) {

courseService.deleteCourse(id);

return new ResponseEntity<>(HttpStatus.NO\_CONTENT);

[1. github.com](https://github.com/LorenaMitelut/course-scheduling-app" \t "_blank)

[github.com](https://github.com/LorenaMitelut/course-scheduling-app" \t "_blank)

}

}

Use code [with caution.](/faq#coding)

**Note:** This is a basic implementation and doesn't include features like data validation, exception handling, or security. For a production environment, you'd need to incorporate these aspects.

**Additional Considerations:**

* **Data Persistence:** Replace the in-memory list with a database like MySQL, PostgreSQL, or MongoDB using Spring Data JPA or Spring Data MongoDB.
* **Error Handling:** Implement proper error handling mechanisms to provide informative responses.
* **Testing:** Write unit and integration tests to ensure code quality and reliability.
* **Security:** Protect your API with authentication and authorization mechanisms.

By building upon this foundation and addressing the mentioned points, you can create a robust and scalable CRUD API.

**Would you like to explore any of these additional points in more detail?**