AI-Powered University Course Management System

Problem Statement

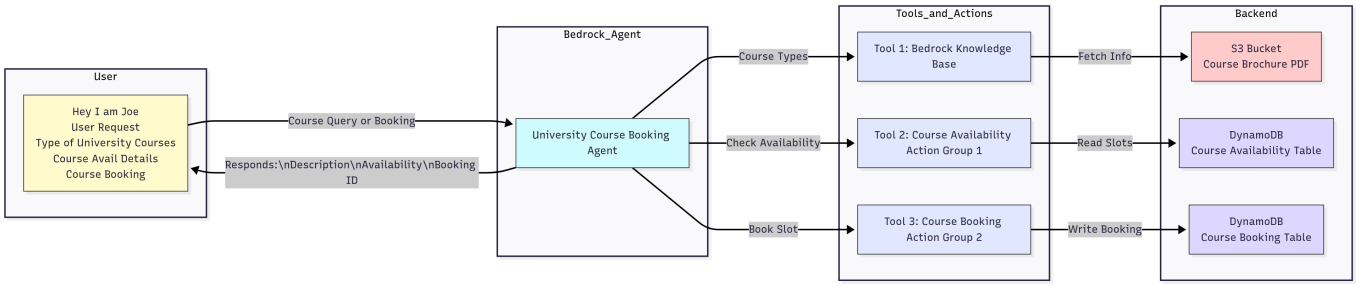
Universities often struggle with managing real-time course information and student enrollment efficiently across departments. Students face difficulties in discovering available courses, checking slot availability, and booking seats — especially during peak enrollment periods.

Traditional portals require multiple steps, manual interventions, and offer limited interactivity. There's a need for a smart, conversational system that allows students to interact naturally, get accurate information, and perform bookings seamlessly.

Use Case: AI-Powered University Course Management

Goal

To build a fully automated **University Course Management System** using **AWS Bedrock Agents**, enabling students to explore course offerings, check slot availability, and book course slots through a natural language interface.



Step 1: Building the Bedrock Agent

1. Create a Bedrock Agent

- **Agent Name**: `university-course-booking-agent`
- **Description**: An agent to explore courses, check availability, and book course slots

Steps:

1. Go to the **AWS Console** and navigate to **Amazon Bedrock**
2. On the left menu, click on **Agents**
3. Click on **Create Agent**
4. Enter:
    - **Name**: `university-course-booking-agent`
    - **Description**: "*An agent to explore and book university courses via natural language*"
5. **Uncheck** multi-agent collaboration (single agent use case)
6. Click **Create**

---

2. Set Up Agent Permissions

- Let Bedrock **create a new IAM service role** with required permissions
- This allows the agent to interact with services like:
  - Lambda (Action Groups)
  - DynamoDB
  - S3 (Knowledge Base)

---

3. Select Foundation Model

- Click **Select Model**
- Choose:
  - **Provider**: `Anthropic`
  - **Model**: `Amazon Nova Pro` (preferred for reasoning tasks)
- If throttling occurs or access is limited:
  - Switch to **Amazon Titan Text Premier** (note: no long-term memory support)

---

4. Define Agent Instructions

Add the following instructions to guide the agent's behavior:

```
University Course Booking Assistant for Example University

You are a course management assistant for Example University. Your role is to help
prospective students and learners explore available courses, check slot
availability, and book course slots for the academic year.

Your primary responsibilities are:
Greet the user with: "Welcome to Example University – Your gateway to world-class
education!"
Help users:
    Explore course offerings (degree programs and certifications)
    Search for available slots in specific courses
    Understand course details and highlights
Collect and validate booking details from users
Book course slots and confirm the registration with a Booking ID

Follow these guidelines when interacting with customers:
    Be courteous, professional, and informative throughout the conversation
    Provide accurate responses based on real-time data and the university brochure
(knowledge base)
    Maintain the flow of conversation and ask clarifying questions if required

Course Booking Process

Step 1: Collect Course Booking Details
```

```
    Ask the user to confirm the following information:
    1. Course ID or Course Name
    2. Preferred Date of Enrollment
    3. Student Name
    4. Duration (in days or number of sessions, if applicable)
Step 2: Validate Availability
    Before proceeding to booking:
    Confirm course details from the brochure/knowledge base
    Check available slots for the selected course and date from the real-time
DynamoDB table
    Explain course highlights (e.g., for “MBA201” → “MBA with focus on Business
Analytics, Global Strategy, etc.”)

Step 3: Confirm Booking
    Once availability is verified:
    Repeat back the collected details
    Ask user:“Shall I proceed with booking your slot in [Course Name] on [Date]
for [Student Name]?”

Step 4: Finalize Booking
    Call the slot booking action via Lambda

    On success, return: Unique Booking ID (e.g., UCMS-XYZ123)
    Confirmation message: “Your booking is confirmed! Your Booking ID is UCMS-
XYZ123.”
    Mention location and mode (e.g., On-campus or Online, if available)
    Inform about check-in dates (if residential programs), orientation schedule,
etc.
```

Paste this into the agent's **Instruction** section.

---

5. Save and Prepare Agent

- Click **Save and Exit**
- Then click **Prepare** (required to activate recent changes)

---

6. Test the Agent (Basic)

Without tools connected yet, the agent can greet users but cannot perform course queries.

Example:

```
> User: "Hi"
> Agent: "Welcome to the University Course Booking System. How may I assist you
today?"

> ❗ Note: Booking/course details will not work until Action Groups and Knowledge
Base are added.
```

Tool 1: Explore Courses using Knowledge Base

The first tool we add to the agent is the **Amazon Bedrock Knowledge Base**, which allows the agent to perform **RAG (Retrieval-Augmented Generation)** using unstructured data — in our case, a PDF brochure containing course descriptions.

Goal

Enable the agent to answer natural language questions related to university course offerings and descriptions by indexing a university brochure stored in S3.

---

1. Create and Upload the Course Brochure PDF to S3

1. Go to **S3**
2. Create a new bucket: `university-course-brochure`
3. Upload the university brochure PDF (e.g., `coursebrochure.pdf`) containing course types, descriptions, and departments

---

2. Create a Bedrock Knowledge Base

1. Go to **Amazon Bedrock** → **Build Tools** → **Knowledge Bases**
2. Click **Create** → **knowledge base with vector store**
3. Fill the form of **Provide Knowledge Base details** and click **Next**:

Field	Value
Name	`course-brochure-kb`
Description	Tool 1 for University Agent
IAM Role	Create and use a new service role
Data Source Type	Amazon S3

4. Configure the **Data Source**:
    - Data Source Name: `university-course-brochure`
    - S3 Bucket: select the pdf file uploaded earlier using the **Browse** button
    - Parsing Strategy: `Amazon Bedrock default Parser`
    - and other settings as default
    - click **Next**
5. Configure **data storage and processing**:
    - Select the Embedding Model: `Titan Text Embeddings V2`
    - Vector Store creation: **Create a new vector store**
    - Vector source type: **Amazon Aurora PostgreSQL Serverless**
    - click **Next**

6. Review the configuration and click **Create Knowledge Base**
7. Wait for the Knowledge Base to be created (this may take a more than 15 minutes depending on the PDF size)

---

4. Sync the Knowledge Base

- Once created and ready, go to the **Knowledge Base** page
- Select the newly created Knowledge Base: `course-brochure-kb`
- Look for the **Data Sources** and select the data source: `university-course-brochure`
- Click on the **Sync** button.
- Wait for status: `Sync completed`

---

5. Test the Knowledge Base

1. Go to **Amazon Bedrock > Knowledge Bases**
2. Select the Knowledge Base: `course-brochure-kb`
3. Click on **Test Knowledge Base** button
4. Select the model: `Amazon Nova Pro`

```
- What courses are available in the School of Business?
- Tell me about the Data Science program.
```

You should see responses sourced from the uploaded PDF.

---

6. Connect the Knowledge Base to the Agent

1. Go back to **Amazon Bedrock > Agents**
2. Select the agent: `university-course-booking-agent`
3. Click on **Edit in Agent Builder**
4. Scroll to **Knowledge Bases** section
5. Click **Add**
6. Select: `course-brochure-kb`
7. Provide routing instructions, e.g.:

```
Route user queries related to course types, department offerings, or course
descriptions to the knowledge base.
```

8. Click **Add**

---

7. Save and Prepare Agent

- Click **Save and Exit**
- Then click **Prepare** to apply changes

---

8. Test Agent with Knowledge Base Integrated

Use the agent test chat panel:

- Ask:

```
I would like to explore the courses offered by Example University. Please
provide a list of available programs along with their course IDs, durations,
and highlights.
```

- The routing trace should show: `orchestration > knowledgeBase`
- Response should include course titles and descriptions retrieved from the brochure PDF

Tool 2: Course Slot Availability Check

To support real-time course slot availability, we add **Tool 2**, which integrates a **DynamoDB table** and a **Lambda function** through an **Amazon Bedrock Action Group**.

This allows students to ask questions like:

> "Is there a seat available for MBA201 on December 25, 2025?"

Goal

Create a tool that enables the Bedrock agent to query course slot availability using a Lambda-backed Action Group that reads from a DynamoDB table.

Step 1: Create a DynamoDB Table

Table Name: `courseAvailabilityTable`
Partition Key: `course_id` (String)

Sample Schema:

course_id	course_name	date	slot
CERT103	UI/UX Design and Prototyping	2025-12-25	7

course_id	course_name	date	slot
PHY201	M.Sc in Physics	2025-12-25	3
CERT101	Data Analytics with Python	2025-12-25	12
ENG101	BA in English Literature	2025-12-25	2
CSE101	B.Tech in Computer Science & Engineering	2025-12-25	5
MBA201	MBA (Master of Business Administration)	2025-12-25	10

Steps:

1. Go to **DynamoDB** → Click **Create Table**
2. Enter:
    - **Table Name**: `courseAvailabilityTable`
    - **Partition Key**: `date` (String)
3. Click **Create Table**

---

2. Insert Sample Data

1. Go to **Explore Table Items**
2. Click **Create Item**
3. Add key-value pairs:

```
{
  "course_id": {
    "S": "CERT103"
  },
  "course_name": {
    "S": "UI/UX Design and Prototyping"
  },
  "date": {
    "S": "2025-12-26"
  },
  "slot": {
    "N": "7"
  }
}
```

4. Repeat for other dates using **JSON View** to copy and paste structure:

```
{
  "course_id": {
    "S": "PHY201"
  },
  "course_name": {
    "S": "M.Sc in Physics"
```

```
  },
  "date": {
    "S": "2025-12-26"
  },
  "slot": {
    "N": "3"
  }
}
```

```
{
  "course_id": {
    "S": "CERT101"
  },
  "course_name": {
    "S": "Data Analytics with Python"
  },
  "date": {
    "S": "2025-12-26"
  },
  "slot": {
    "N": "12"
  }
}
```

```
{
  "course_id": {
    "S": "ENG101"
  },
  "course_name": {
    "S": "BA in English Literature"
  },
  "date": {
    "S": "2025-12-26"
  },
  "slot": {
    "N": "2"
  }
}
```

```
{
  "course_id": {
    "S": "CSE101"
  },
  "course_name": {
    "S": "B.Tech in Computer Science & Engineering"
  },
  "date": {
```

```
      "S": "2025-12-26"
    },
    "slot": {
      "N": "5"
    }
}
```

```
{
  "course_id": {
    "S": "MBA201"
  },
  "course_name": {
    "S": "MBA (Master of Business Administration)"
  },
  "date": {
    "S": "2025-12-26"
  },
  "slot": {
    "N": "10"
  }
}
```

5. Repeat for all courses and dates as needed

Step 2: Add Lambda Function for Course Slot Availability

We now create a **Lambda function** that will serve as the backend for our second tool—**slot availability check**. This tool allows the agent to fetch the number of available seats for each course on a specified date.

Goal

Build a Lambda function that:

- Accepts a `course_id` input via event
- Queries a DynamoDB table (`courseAvailabilityTable`)
- Returns available seats for each course on that date
- Responds in the format expected by a **Bedrock Agent Action Group**

---

1. Create the Lambda Function

1. Go to **AWS Lambda** → Click **Create function**
2. Select **Author from scratch**
3. Fill in the details:
    - **Function name**: `CourseAvailabilityFunction`

- **Runtime**: Python 3.13

4. **Permissions**: Create a new role with basic Lambda permissions

5. Click **Create function**

---

2. Configure Lambda Function

1. In the function code editor, replace the default code with the following:

```
#1 imports
import json
import boto3
#2 Create a client connection -
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamod
b.html
client = boto3.client('dynamodb')

def lambda_handler(event, context):

#3 Store the user input - date to check room availability
    print(f"The user input is {event}")
    user_input_date = event['parameters'][0]['value']

#4 Reference the dynamodb table and retrieve data -
https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/dynamod
b/client/get_item.html
    response = client.get_item (TableName='courseAvailabilityTable', Key=
{'course_id': {'S': user_input_date}})
    #print(response)
    room_inventory_data = response['Item']
    print(room_inventory_data)

#5 Format the response as per the requirement of Bedrock Agent Action Group -
https://docs.aws.amazon.com/bedrock/latest/userguide/agents-lambda.html
    agent = event['agent']
    actionGroup = event['actionGroup']
    api_path = event['apiPath']
    # get parameters
    get_parameters = event.get('parameters', [])


    response_body = {
        'application/json': {
            'body': json.dumps(room_inventory_data)
        }
    }

    print(f"The response to agent is {response_body}")

    action_response = {
        'actionGroup': event['actionGroup'],
```

```
        'apiPath': event['apiPath'],
        'httpMethod': event['httpMethod'],
        'httpStatusCode': 200,
        'responseBody': response_body
    }

    session_attributes = event['sessionAttributes']
    prompt_session_attributes = event['promptSessionAttributes']

    api_response = {
        'messageVersion': '1.0',
        'response': action_response,
        'sessionAttributes': session_attributes,
        'promptSessionAttributes': prompt_session_attributes
    }

    return api_response
    print(f"The final response is {api_response}")
```

6. Click **Deploy** to save the changes

---

3. Configure Lambda Function Settings

1. In the **Configuration** tab, set the following:
2. In General Configuration click **Edit**:
    - **Timeout**: Increase to **1 minute 30 seconds** (default is 3 seconds, which is too short for DynamoDB queries and processing)
3. Click **Save**
4. In the **Permissions** tab, click on the role name to open the IAM console
5. Attach the following policies to the Lambda execution role:
    - **AdministratorAccess** (for development purposes) optionally
    - **AmazonDynamoDBReadOnlyAccess** (for reading from DynamoDB)
    - **AWSLambdaBasicExecutionRole** (for logging)
    - **AmazonBedrockFullAccess** (for invoking Bedrock agents)
6. Click **Add permissions** → **Attach policies**
7. Go back to the Lambda function configuration
8. In the **Configuration** tab , click on **Permissions** tab.
    - look for the **Resource-based policy statement** section
    - click on the **Add permissions** button
    - select **AWS Service** and choose Service: `Other`
    - enter the statement id: `AllowBedrockInvoke`

- enter the principal: `bedrock.amazonaws.com`
- enter the source ARN of the agent created earlier: `arn:aws:bedrock:us-east-1:<account-id>:agent/university-course-booking-agent`
- Action : `lambda:InvokeFunction`
- click **Save**

Step 3: Create and Integrate OpenAPI Schema by Action Group

With the Lambda function ready to fetch course availability, we now integrate it into the Bedrock Agent using an **OpenAPI schema** via **Action Group**.

1. Create the OpenAPI Schema

Use the **inline OpenAPI definition** in the Agent Builder UI. Below is the sample schema tailored for our slot availability use case:

```
openapi: 3.0.0
info:
  title: Course Slot Availability API
  version: 1.0.0
  description: API for checking slot availability for a particular course

paths:
  /getCourse/{course_id}:
    get:
      summary: Get course slot availability
      description: Returns the slot availability and course name for the given
course ID
      operationId: getCourse
      parameters:
        - name: course_id
          in: path
          description: Course ID
          required: true
          schema:
            type: string
      responses:
        "200":
          description: Course slot details retrieved successfully
          content:
            application/json:
              schema:
                type: object
                properties:
                  course_name:
                    type: string
                    description: Name of the course
                  slot:
                    type: integer
                    description: Number of available slots
```

```
        "404":
          description: Course not found
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
                    example: Course not found
```

---

2. Register Action Group in Bedrock Agent

- Open **Bedrock → Agents**
- Select your Agent (e.g., `university-course-booking-agent`)
- Click **Edit in Agent Builder**
- Under **Action Groups**, click **Add Action Group**
- Fill details:
  - **Action Group Name**: `CourseAvailabilityChecker`
  - **Description**: "Checks seat availability for courses on a specific date"
  - **Action Group Type**: `Define with API Schemas`
  - **Action Group Invocation**: `Select an existing Lambda function`
  - Select the Lambda function: `CourseAvailabilityFunction`
  - **Action Group schema**: select `Define via in-line schema editor`
  - Paste the OpenAPI schema defined above
  - Click on **Create**

Click **Save and Exit**, then **Prepare** the agent again.

---

3. Test the Agent Course Slot Availability Check

Now that the Action Group is set up, we can test the agent's ability to check course slot availability.

```
Can you check if there are available slots for course ID MBA201?
```

Tool 3: Book Course Slot

In the previous tools, the user was able to:

- Explore courses using a Knowledge Base
- Check course slot availability using a Lambda function and DynamoDB

Now, we implement **Tool 3**, where the user can **book a course slot** based on chosen date, course ID, and number of participants.

Goal

Allow the user to:

- Submit a booking request with:
    - Course ID
    - course Date
    - student name
    - Booking ID
- Check if slots are available for that date
- Book the course slot if available
- Return a unique Booking ID for confirmation

Step 1: Create a DynamoDB Table for Course Bookings

1. DynamoDB Table: `courseBookingTable`

We'll use a new DynamoDB table to store booking data.

- **Table Name**: `courseBookingTable`
- **Partition Key**: `bookingID` (Type: String)

Example Booking Record

```
{
  "bookingID": {
    "S": "dd5da4d2-dbdb-44f5-b303-b2961e0019b5"
  },
  "courseDate": {
    "S": "2025-12-25"
  },
  "course_id": {
    "S": "CERT101"
  },
  "studentName": {
    "S": "Priya Sharma"
  }
}
```

---

2. Create Table from AWS Console

Follow these steps to create the table:

1. Go to **AWS Console** → **DynamoDB**
2. Click **Create Table**
3. Enter:
    - **Table name**: `courseBookingTable`
    - **Partition key**: `bookingID` (Type: String)
4. Leave other options as default
5. Click **Create Table**

Once created, your table should appear in the DynamoDB dashboard.

Step 2: Create Lambda Function for Course Booking

1. Create Lambda Function: `CourseBookingFunction`

1. Go to **AWS Lambda** → **Create function**
2. Select **Author from scratch**
3. Fill in the details:
    - **Function name**: `CourseBookingFunction`
    - **Runtime**: Python 3.13
    - **Permissions**: Create a new role with basic Lambda permissions
4. Click **Create function**

2. Configure Lambda Function Code

1. In the function code editor, replace the default code with the following:

```
import json
import boto3
import uuid

# Create DynamoDB client
client = boto3.client('dynamodb')

def lambda_handler(event, context):
    print(f"The user input from Agent is {event}")

    # Extract user inputs from request body
    input_data = event['requestBody']['content']['application/json']['properties']

    # Initialize variables
    studentName = course_id = courseDate = None
```

```
    # Extract values from input
    for item in input_data:
        if item['name'] == 'studentName':
            studentName = item['value']
        elif item['name'] == 'course_id':
            course_id = item['value']
        elif item['name'] == 'courseDate':
            courseDate = item['value']

    # Query courseAvailabilityTable for available slots
    try:
        response = client.get_item(
            TableName='courseAvailabilityTable',
            Key={
                'course_id': {'S': course_id}
                }
        )
        if 'Item' not in response:
            raise Exception("No availability found for the selected course and
date.")

        course_inventory = response['Item']
        current_slot = int(course_inventory['slot']['N'])

        print(f"Available slots for {course_id} on {courseDate}: {current_slot}")

        if current_slot == 0:
            return {
                'statusCode': 404,
                'body': json.dumps({'error': 'No slots available for the selected
course and date'})
            }

        # Proceed with booking
        bookingID = str(uuid.uuid4())

        # Insert booking into courseBookingTable
        client.put_item(
            TableName='courseBookingTable',
            Item={
                'bookingID': {'S': bookingID},
                'studentName': {'S': studentName},
                'course_id': {'S': course_id},
                'courseDate': {'S': courseDate}
            }
        )

        print(f"Booking confirmed. ID: {bookingID}")

        # Prepare Bedrock agent-compatible response
        response_body = {
            'application/json': {
                'body': json.dumps({'bookingID': bookingID})
```

```
            }
        }

        action_response = {
            'actionGroup': event['actionGroup'],
            'apiPath': event['apiPath'],
            'httpMethod': event['httpMethod'],
            'httpStatusCode': 200,
            'responseBody': response_body
        }

        return {
            'messageVersion': '1.0',
            'response': action_response,
            'sessionAttributes': event.get('sessionAttributes', {}),
            'promptSessionAttributes': event.get('promptSessionAttributes', {})
        }

    except Exception as e:
        print(f"Error occurred: {str(e)}")
        return {
            'statusCode': 500,
            'body': json.dumps({'error': str(e)})
        }
```

2. Click **Deploy** to save the changes

3. Configure Lambda Function Settings

1. In the **Configuration** tab, set the following:
2. In General Configuration click **Edit**:
    - **Timeout**: Increase to **1 minute 30 seconds** (default is 3 seconds, which is too short for DynamoDB queries and processing)
3. Click **Save**
4. In the **Permissions** tab, click on the role name to open the IAM console
5. Attach the following policies to the Lambda execution role:
    - **AdministratorAccess** (for development purposes) optionally
    - **AmazonDynamoDBReadOnlyAccess** (for reading from DynamoDB)
    - **AWSLambdaBasicExecutionRole** (for logging)
    - **AmazonBedrockFullAccess** (for invoking Bedrock agents)
6. Click **Add permissions → Attach policies**
7. Go back to the Lambda function configuration
8. In the **Configuration** tab , click on **Permissions** tab.

- look for the **Resource-based policy statement** section
- click on the **Add permissions** button
- select **AWS Service** and choose Service: `Other`
- enter the statement id: `AllowBedrockInvoke`
- enter the principal: `bedrock.amazonaws.com`
- enter the source ARN of the agent created earlier: `arn:aws:bedrock:us-east-1:<account-id>:agent/university-course-booking-agent`
- Action : `lambda:InvokeFunction`
- click **Save**

Step 3: Create and Integrate OpenAPI Schema by Action Group

With the Lambda function ready to handle course bookings, we now integrate it into the Bedrock Agent using an **OpenAPI schema** via **Action Group**.

1. Create the OpenAPI Schema

Use the **inline OpenAPI definition** in the Agent Builder UI. Below is the sample schema tailored for our course booking use case:

```
openapi: 3.0.0
info:
  title: University Course Booking API
  version: 1.0.0
  description: API to book a course slot at Example University

paths:
  /bookCourseSlot:
    post:
      summary: Book a slot for a university course
      description: Allows a student to book a course slot by providing course ID,
student name, and date
      operationId: bookCourseSlot
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                course_id:
                  type: string
                  description: Unique ID of the course (e.g., CERT101, MBA201)
                courseDate:
                  type: string
                  description: Date of course enrollment (YYYY-MM-DD)
                studentName:
                  type: string
                  description: Full name of the student booking the course
              required:
```

```
                - course_id
                - courseDate
                - studentName
      responses:
        "200":
          description: Booking confirmed. Returns unique Booking ID.
          content:
            application/json:
              schema:
                type: object
                properties:
                  bookingID:
                    type: string
                    description: Your booking is confirmed. Booking Confirmation
ID.
        "400":
          description: Bad request. One or more required fields are missing or
invalid.
        "404":
          description: No slots available for the given course ID and date.
        "500":
          description: Internal server error. Something went wrong on the server.
```

---

2. Register Action Group in Bedrock Agent

- Open **Bedrock → Agents**
- Select your Agent (e.g., `university-course-booking-agent`)
- Click **Edit in Agent Builder**
- Under **Action Groups**, click **Add Action Group**
- Fill details:
  - **Action Group Name**: `CourseBooking`
  - **Description**: "Book a course slot for a student"
  - **Action Group Type**: `Define with API Schemas`
  - **Action Group Invocation**: `Select an existing Lambda function`
  - Select the Lambda function: `CourseBookingFunction`
  - **Action Group schema**: select `Define via in-line schema editor`
  - Paste the OpenAPI schema defined above
  - Click on **Create**

Click **Save and Exit**, then **Prepare** the agent again.

---

3. Test the Agent Course Booking

Now that the Action Group is set up, we can test the agent's ability to book a course slot.

```
Can you book a slot for course ID MBA201 on 2025-12-25 for student Priya Sharma?
```

The agent should respond with a confirmation message and a unique Booking ID.

Conclusion

With the three tools integrated into the Bedrock Agent, we have built a comprehensive **University Course Management System** that allows students to:

- Explore courses using a Knowledge Base
- Check course slot availability using a Lambda function and DynamoDB
- Book course slots with real-time availability checks and confirmations

---