

# Chapter 1: The Contract — OpenAPI Specification

## Introduction: What You Will Learn

Before you write actual code, you need a plan. In web development, this plan is called an **API Specification**. It's a contract that says:

- "These are the URLs you can call"
- "This is the data you must send"
- "This is the data you will receive"

By the end of this chapter, you will understand:

1. What OpenAPI/Swagger is
  2. How to read and write YAML
  3. Every section of our `openapi.yaml` file
  4. Why we made specific design decisions
- 

## Part 1: What is OpenAPI?

### The Problem: Miscommunication

Imagine this scenario:

- **Frontend Developer:** "I need the list of messages"
- **Backend Developer:** "OK, call `/getMessages`"
- **Frontend Developer:** "It doesn't work"
- **Backend Developer:** "Oh, you need to send the conversation ID in the body"
- **Frontend Developer:** "Which field name? `conversationId`? `conv_id`? `id`?"

This is a disaster. People waste time on miscommunication.

### The Solution: Write It Down

OpenAPI (formerly called Swagger) is a **formal document** that defines:

- Every URL your API supports
- What data each URL expects
- What data each URL returns

- What errors can happen

It's written in YAML or JSON format, and it can:

- Generate documentation automatically
- Generate code automatically
- Be validated by tools

**Our File:** openapi.yaml This file is the "Law" of our project. It defines EVERYTHING our backend does.

---

## Part 2: YAML for Beginners

### What is YAML?

YAML = "YAML Ain't Markup Language". It's a human-readable data format. It uses **indentation (spaces)** to show structure.

### Basic Rules

1. **Use spaces, not tabs** (2 spaces per level is standard).
  2. **Colons** separate keys from values: name: Alice
  3. **Dashes** create lists: - item1
  4. **Nested items** are indented.
- 

## Part 3: The Structure of openapi.yaml

Our file has these main sections:

```
openapi: 3.0.3          # Version of OpenAPI we're using
info:                  # Metadata about our API
  title: WASAText API
  version: "1.0.0"

components:             # Reusable definitions (Security schemes, Data models)
  securitySchemes: ...
  schemas: ...

paths:                  # THE MAIN PART: All URLs
  /session: ...
  /conversations: ...
```

---

# Part 4: The info and components Sections

## The info Section

```
info:  
  title: WASAText API  
  description: |  
    API specification for WASAText messaging application.  
    Built according to the PDF specification.  
  version: "1.0.0"
```

This is metadata used by tools like Swagger UI to display a nice header.

## The securitySchemes Section

```
components:  
  securitySchemes:  
    bearerAuth:  
      type: http  
      scheme: bearer  
      description: Use the user identifier returned from doLogin
```

**Translation:** "We use HTTP Bearer authentication. The client must send Authorization: Bearer <token>."

# Part 5: The paths Section — The Heart of the API

This is where we define EVERY endpoint. Let's explain the key ones.

## Example 1: Login (POST /session)

```

paths:
  /session:
    post:
      tags: ["login"]
      operationId: doLogin
      summary: Logs in the user
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                name:
                  type: string
                  example: Maria
                  minLength: 3
                  maxLength: 16
                required:
                  - name
      responses:
        '201':
          description: User log-in action successful
          content:
            application/json:
              schema:
                type: object
                properties:
                  identifier:
                    type: string
                    example: "abcdef012345"

```

#### **Breakdown:**

- **URL:** /session
- **Method:** POST (We are creating a session)
- **operationId:** doLogin (Crucial for code generation!)
- **requestBody:** Expects JSON with a `name` field (3-16 chars).
- **responses:** Returns 201 Created with an `identifier`.

## Example 2: Send Message (POST `/conversations/`)

```

/conversations/{conversationId}/messages:
  post:
    tags: ["message"]
    operationId: sendMessage
    security:
      - bearerAuth: []
    parameters:
      - name: conversationId
        in: path
        required: true
        schema:
          type: string

```

#### Key Concepts:

- {conversationId}: This is a **path parameter**. The actual URL might be `/conversations/group123/messages`.
- in: path: Tells the server to look for `conversationId` in the URL, not the body.
- security: [bearerAuth: []]: You **MUST** be logged in to use this.

## Part 6: Design Decisions — Why We Did It This Way

### Decision 1: Why `/conversations/`

**The Question:** Should messages have their own top-level resource? **Our Choice:** Messages are nested under conversations. **Reasoning:**

- A message CANNOT exist without a conversation.
- When you fetch messages, you always want them for a specific conversation.
- This makes authorization natural: if you can access `/conversations/5`, you can access its messages.

### Decision 2: Why POST /session for login?

**The Question:** What URL should handle login? **Our Choice:** POST /session **Reasoning:**

- Login creates a "session" (conceptually).
- REST is about resources. A session IS a resource.
- In a real app, you might have `DELETE /session` for logout.

## Decision 3: Why separate `operationId` for every endpoint?

**The Question:** Do we need `operationId`? **Our Choice:** Yes, always. **Reasoning:**

- Code generators use `operationId` to name functions.
  - Without it, you get auto-generated names like `postConversationsConversationIdMessages`.
  - With it, you get clean names like `sendMessage`.
- 

## Summary Checklist

Before moving to Chapter 2, make sure you understand:

- What OpenAPI is and why we use it
- How to read YAML (indentation, lists, objects)
- What `operationId` does
- The difference between `path` parameters and `requestBody`
- What `security: [bearerAuth: []]` means
- Why we specific status codes like 201 (Created) vs 200 (OK)

## Oral Exam Questions

**Q: Why did you use OpenAPI?** "I used OpenAPI 3.0 because it's the industry standard for REST API documentation. It provides a single source of truth that both frontend and backend developers can reference. It also serves as a contract that prevents miscommunication."

**Q: Explain your URL structure.** "I followed RESTful conventions. Resources are nouns (`users`, `conversations`, `messages`). Nested resources represent hierarchy (`messages` belong to `conversations`). This makes the API intuitive and simplifies access control."

**Q: Why is `operationId` important?** "`OperationId` is crucial for code generation and maintenance. It allows us to map a specific API endpoint (like `POST /session`) to a specific function name in our code (like `doLogin`), acting as the bridge between the spec and the implementation."