Enterprise Programming (Course Slides)

Module3

Refer for detailed notes :-

https://github.com/sagaruppuluri/EP/blob/main/Module3/Readme.md

HTTP

Refer:
https://github.com/sagaruppuluri/EP/blob/main/Module3/HTTPBasics.pdf

REST (Representation State Transfer)

Architectural style for building web services that use HTTP methods to perform operations on resources.

REST (Representation State Transfer)

Key Principles:
- **Resources**: Everything is a resource (User, Product, Order)
- **URIs**: Each resource has a unique identifier (URL)
- **HTTP Methods**: Use standard methods (GET, POST, PUT, DELETE)
- **Stateless**: Each request contains all necessary information
- **JSON/XML**: Data exchange format

REST (Representation State Transfer)

Traditional:

```
POST /getUserById
POST /createUser
POST /updateUser
POST /deleteUser
```

RESTful:

```
GET /users/{id} - Get user
POST /users - Create user
PUT /users/{id} - Update user
DELETE /users/{id} - Delete user
```

Open API

- **Language agnostic interface description for HTTP APIs.**
- **Described using YAML or JSON.**
- **Key Benefits**
  - **Standardization**: Industry-standard format for API documentation
  - **Auto-generation**: Generate client SDKs, server stubs, and documentation
  - **Validation**: Validate requests and responses automatically
  - **Testing**: Enable automated API testing
  - **Discoverability**: Make APIs easier to understand and consume

JSON vs YAML

JSON (JavaScript Object Notation)

- Strict syntax with braces, brackets, quotes, and commas
- Primarily for data exchange between systems
- No comments allowed

```
{
 "user": {
 "name": "John Doe",
 "roles": ["admin", "developer"],
 "address": {
   "street": "123 Main St",
   "city": "Boston"
  }
 }
}
```

YAML (YAML Ain't Markup Language)

- Human-readable, uses indentation
- Popular for configuration files
- Supports comments with #

```
user:
  name: John Doe
  roles:
   - admin
   - developer
  address:
    street: 123 Main St
    city: Boston
```

Core Components

Meta Information

Paths:
API Endpoints

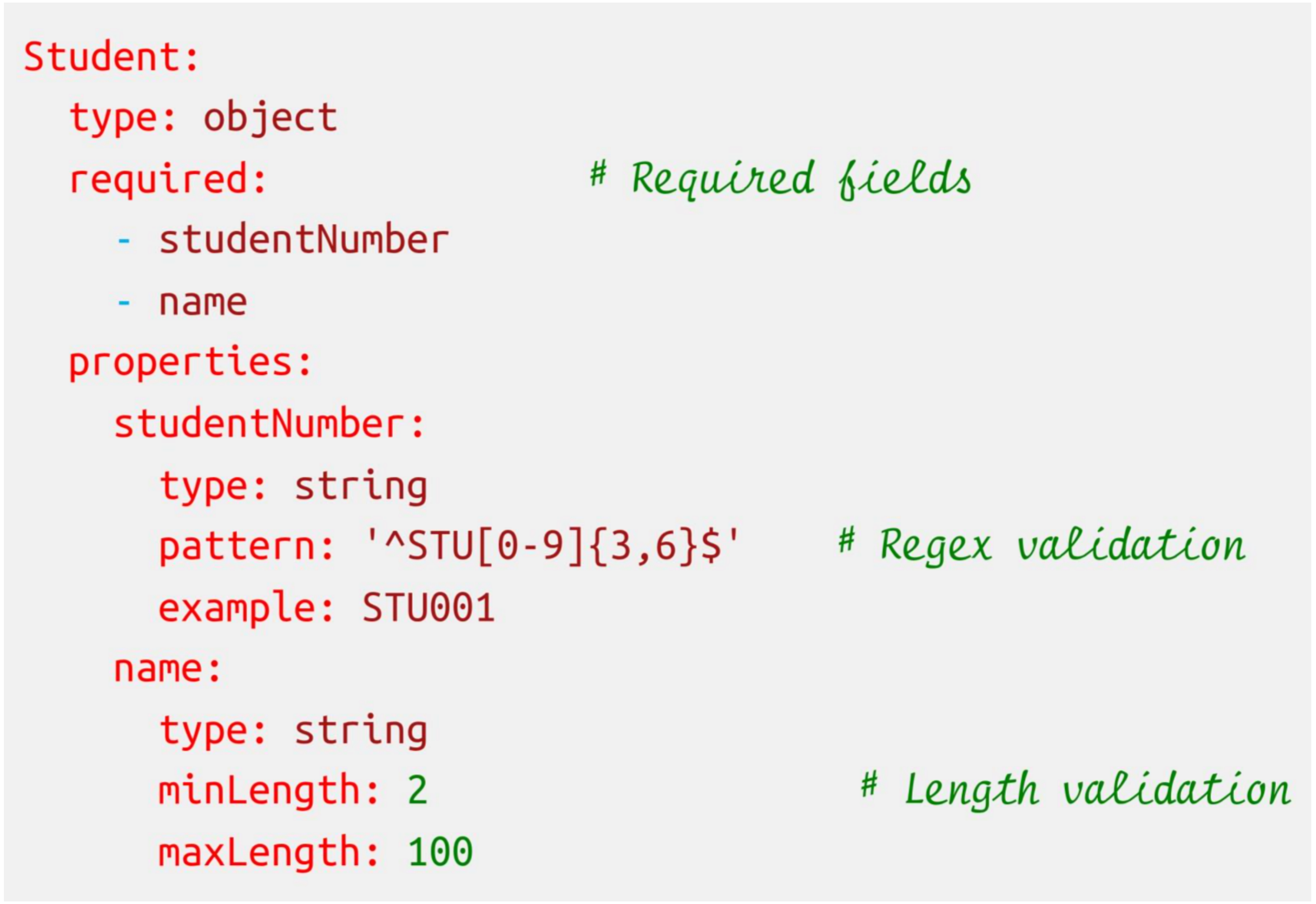
Components:
Reusable definitions (schema etc)

Security:
Authentication requirements

```yaml
openapi: 3.0.4
info:
  title: Sample API
  version: 1.0.0
  description: A sample API to demonstrate OpenAPI specification
servers:
  - url: https://api.example.com/v1
paths:
  /users:
    get:
      summary: Get a list of users
      responses:
        '200':
          description: A list of users
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/User'
components:
  schemas:
  User:
    type: object
    properties:
      id:
        type: integer
      name:
        type: string
      email:
        type: string
security:                          # Global security requirements
  - bearerAuth: []
```

1. Schemas (Data Models)

Schemas define the structure of request and response bodies.



```
Student:
  type: object
  required:                    # Required fields
    - studentNumber
    - name
  properties:
    studentNumber:
      type: string
      pattern: '^STU[0-9]{3,6}$'    # Regex validation
      example: STU001
    name:
      type: string
      minLength: 2                   # Length validation
      maxLength: 100
```

Schema Reusability

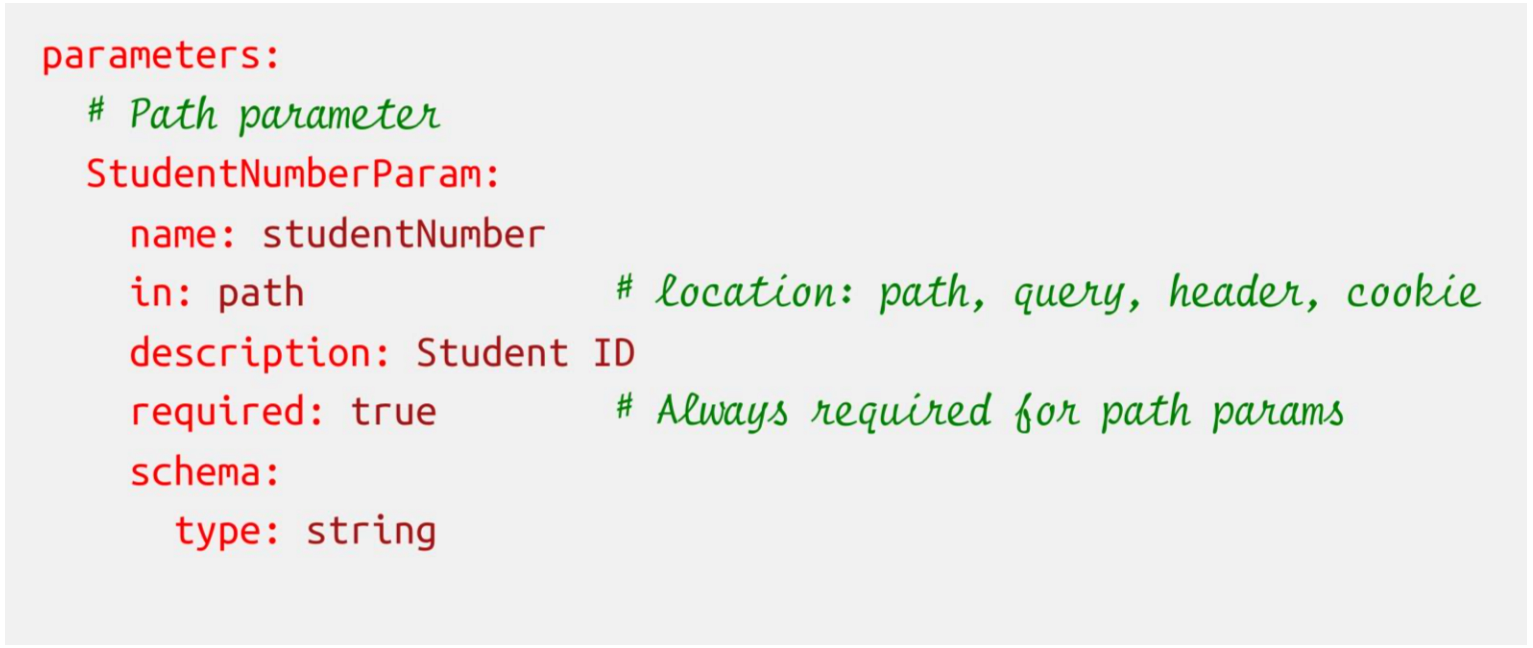
```
# Define once
components:
  schemas:
    Address:
      type: object
      properties:
        city:
          type: string

# Reuse multiple times
Student:
  properties:
    address:
      $ref: '#/components/schemas/Address'

Teacher:
  properties:
    address:
      $ref: '#/components/schemas/Address'
```

2. Parameters

Path, Query, Header, or Cookie.



```yaml
parameters:
  # Path parameter
  StudentNumberParam:
    name: studentNumber
    in: path              # location: path, query, header, cookie
    description: Student ID
    required: true        # Always required for path params
    schema:
      type: string
```

3. Request Bodies

Structure of request payloads.

```yaml
requestBody:
  required: true
  description: Student to create
  content:
    application/json:         # Content type
      schema:
        $ref: '#/components/schemas/StudentCreateRequest'
      examples:               # Multiple examples
        example1:
          summary: Basic student
          value:
            studentNumber: STU001
            name: John Doe
```

4. Responses

Define possible API responses.

```yaml
responses:
  '200':                        # HTTP status code
    description: Success
    content:
      application/json:    # Content type
        schema:
          $ref: '#/components/schemas/Student'
  '404':
    description: Not found
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
```

5. Security Schemes

Define authentication methods.

Bearer Authentication (JWT)

```
securitySchemes:
  bearerAuth:
    type: http
    scheme: bearer
    bearerFormat: JWT
    description: JWT token authentication
```

6. Tags

Organize endpoints into logical groups.

```yaml
tags:
  - name: Students
    description: Student management operations
    externalDocs:
      description: Find out more
      url: https://docs.example.com/students
  - name: Admin
    description: Administrative operations
```

```yaml
paths:
  # ==================== Health Check ====================
  /health:
    get:
      tags:
        - Health
```

REST API with Spring Boot 3

@RestController - simplifies REST API development by:
- Automatically converts return values to JSON/XML
- Eliminates need for `@ResponseBody` on every method
- Designed for REST API development
- Returns data instead of views

RestController vs Controller

`@RestController = @Controller + @ResponseBody`

```
@RestController
@RequestMapping("/api")
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello, World!";
    }

}
```

```
@Controller
@RequestMapping("/api")
public class HelloController {

    @GetMapping("/hello")
    @ResponseBody
    public String hello() {
        return "Hello, World!";
    }
}
```

Key Annotations

@RequestMapping : Maps HTTP requests to handler methods.

```
@RestController
@RequestMapping("/api/users") // Base path for all methods
public class UserController {

    @RequestMapping("/all") // /api/users/all
    public List<User> getAllUsers() {
        return userList;
    }
}
```

Key Annotations

HTTP Method Annotation : Maps HTTP methods to handler methods.

```
@RestController
@RequestMapping("/api/products")
public class ProductController {

    @GetMapping // GET - Read
    @PostMapping // POST - Create
    @PutMapping // PUT - Update
    @PatchMapping // PATCH - Partial Update
    @DeleteMapping // DELETE – Delete

}
```

Key Annotations

@PathVariable : Extracts values from URI paths

```
@RestController
@RequestMapping("/api/products")
public class ProductController {

   @GetMapping("/{id}")  // GET /api/products/123
    public Product getProduct(@PathVariable Long id) {
            return findProductById(id);
     }

}
```

Key Annotations

@RequestParam : Extract query Parameters

```
// GET /api/search?keyword=java&page=1
@GetMapping("/search")
public List<Item> search(
        @RequestParam String keyword,
        @RequestParam(defaultValue = "0") int page) {
    return searchItems(keyword, page);
}
```

Key Annotations

@RequestBody : Binds request body to object (POST, PUT)

```
@PostMapping("/users")
public User createUser(@RequestBody User user) {
    return saveUser(user);
}

// Request JSON:
{
"name": "John Doe",
"email": "john@example.com"
}
```

Key Annotations

@RequestHeader : Access HTTP Headers

```
@GetMapping("/info")
public String getInfo(
        @RequestHeader("User-Agent") String userAgent) {

        return "Browser: " + userAgent;

}
```

Response Handling

Return Simple Types

```java
@GetMapping("/message")
public String getMessage() {
    return "Hello"; // Returns: "Hello"
}
```

Response Handling

Return Objects

```
@GetMapping("/user")
public User getUser() {
    return new User("John", "john@example.com");
}



// Response:
{
"name": "John",
"email": "john@example.com"
}
```

Response Handling

ResponseEntity for status control

```
@GetMapping("/user/{id}")
public ResponseEntity<User> getUser(@PathVariable Long id) {

    User user = findUser(id);

    if (user != null) {
        return ResponseEntity.ok(user); // 200 OK
    } else {// 404 Not Found
        return ResponseEntity.notFound().build();
    }
}
```

Exception Handling: Method Level Exception Handler

```
@RestController
public class ProductController {

    @GetMapping("/products/{id}")
    public Product getProduct(@PathVariable Long id) {
        if (id < 1) {
            throw new IllegalArgumentException("Invalid ID");
        }

        return findProduct(id);
    }

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<String> handleBadRequest(IllegalArgumentException ex) {
            return ResponseEntity
                    .status(HttpStatus.BAD_REQUEST)
                    .body(ex.getMessage());
    }
}
```

Exception Handling: Global Exception Handler

```
@RestControllerAdvice
public class GlobalExceptionHandler {

  @ExceptionHandler(ResourceNotFoundException.class)
  public ResponseEntity<ErrorResponse> handleNotFound(
              ResourceNotFoundException ex) {

        ErrorResponse error = new ErrorResponse(
                        404,
                        ex.getMessage(),
                        System.currentTimeMillis()
                 );

      return ResponseEntity.status(HttpStatus.NOT_FOUND)
                 .body(error);
  }
}
```

API Versioning

```
# URL versioning (recommended)
https://api.example.com/v1

  @RestController
  @RequestMapping("/api/v1/products") // Version in URL
  public class ProductController {
      // ...
  }



# Header versioning, HTTP Header
API-Version: v1|v2
```

Best Practices

Use Proper HTTP Methods

```
// GOOD
@GetMapping("/users") // Read
@PostMapping("/users") // Create
@PutMapping("/users/{id}") // Update
@DeleteMapping("/users/{id}") // Delete

// BAD
@PostMapping("/getUsers")
@PostMapping("/createUser")
```

Use Meaningful URIs

```
// GOOD - Resource-based
/api/customers
/api/customers/{id}
/api/customers/{id}/orders

// BAD - Action-based
/api/getCustomers
/api/createCustomer
```

Return Appropriate Status Codes

```java
@PostMapping("/users")
public ResponseEntity<User> create(
                                  @RequestBody User user) {

  return ResponseEntity
          .status(HttpStatus.CREATED) // 201 instead of 200
          .body(user);
}
```

Version Your API

```
@RestController
@RequestMapping("/api/v1/products") // Version in URL
public class ProductController {
	// ...
}
```

Key Points:

- Use `@RestController` for REST APIs
- Use `@Controller` for traditional MVC (returning views)
- Leverage HTTP method annotations
  (`@GetMapping`, `@PostMapping`, etc.)

- Return `ResponseEntity` for fine-grained control
- Implement proper exception handling
- Validate input with `@Valid`
- Follow REST best practices