Purwadhika
Digital Technology School

AI Fullstack Software Development

Data Validation, File Handling, and Email Integration

Outline

Purwadhika
Digital Technology School

- Data Validation with Zod
- File Upload Handling with Cloudinary
- Email Integration

Introduction to Zod

What is Zod?

- Zod is a TypeScript-first schema validation library.
- It helps ensure that the data your application receives is valid and safe.
- It’s often used for validating inputs from forms, APIs, or database operations.

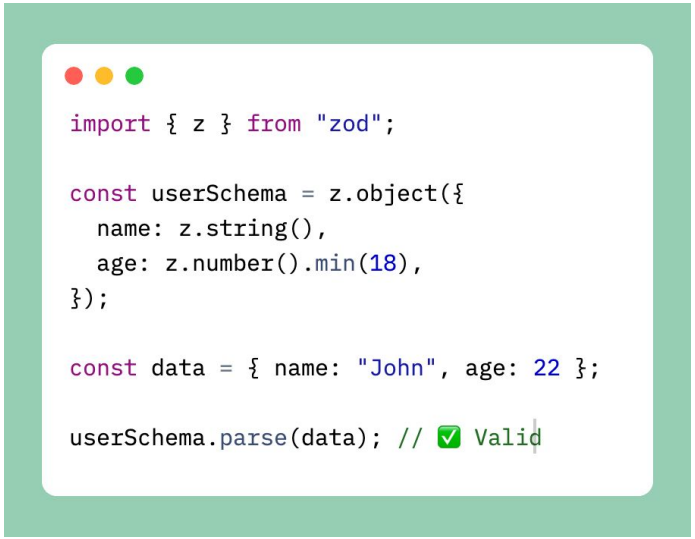
Why use Zod?

✅ Type-safe
✅ Easy to integrate
✅ No runtime dependencies

https://zod.dev/basics

Basic Schema Validation

You can define a schema to validate any type of data.



```
import { z } from "zod";

const userSchema = z.object({
  name: z.string(),
  age: z.number().min(18),
});

const data = { name: "John", age: 22 };

userSchema.parse(data); // ✅ Valid
```

`z.object()` defines an object schema.
`.parse()` validates the input; it throws an error if validation fails.

Handling Validation Errors

When validation fails, Zod throws a detailed error.

```
try {
  userSchema.parse({ name: "John", age: 15 });
} catch (error) {
  console.error(error.errors);
}
```

```
[
  {
    "code": "too_small",
    "minimum": 18,
    "type": "number",
    "message": "Number must be greater than or
        equal to 18"
  }
]
```

Nested and Optional Fields

Zod supports nested schemas and optional fields.

```
const addressSchema = z.object({
  city: z.string(),
  zip: z.string().optional(),
});

const userSchema = z.object({
  name: z.string(),
  address: addressSchema,
});

userSchema.parse({
  name: "Alice",
  address: { city: "New York" },
});
```

`.optional()` means a field can be `undefined`.
Nested validation ensures all inner fields are correct too.

Integrating Zod with Forms or APIs

Zod is perfect for validating request data before processing it.

```
import express from "express";
import { z } from "zod";

const app = express();
app.use(express.json());

const loginSchema = z.object({
  email: z.string().email(),
  password: z.string().min(6),
});

app.post("/login", (req, res) => {
  const result = loginSchema.safeParse(req.body);
  if (!result.success) return res.status(400).json(result.error.errors);

  res.json({ message: "Login successful!" });
});

app.listen(3000);
```

Best Practice: Always validate user input before saving to database or performing logic.

File Upload Handling

File upload sends data as **multipart/form-data**, where the backend receives a binary file stream instead of plain **JSON**.

In Express.js, a middleware like **Multer** intercepts the file before the controller, stores it temporarily (**memory** or **disk**), and provides file **metadata**.

Before uploading to **Cloudinary**, the backend should:

- **Validate** file type and size
- **Secure** the system by rejecting unsafe mime types
- **Transform** the file if needed (rename, resize, compress)
- **Handle** temporarily using memory or a temp folder

Only **valid files** are then uploaded to cloud storage, keeping the upload process safe and controlled.

Integrate with Multer

Use **Multer** to handle file uploads from client forms.

```
npm install multer
```

```
import express from "express";
import multer from "multer";
import { uploadImage } from "./cloudinaryUpload";

const upload = multer({ dest: "uploads/" });
const app = express();

app.post("/upload", upload.single("file"), async (req, res) => {
  const imageUrl = await uploadImage(req.file!.path);
  res.json({ imageUrl });
});

app.listen(4000);
```

What is Cloudinary?

Cloudinary is a service for uploading, optimizing, and managing media files.

Why use it?

- 🧠 Handles image/video optimization automatically
- ☁️ Stores files securely in the cloud
- 🌍 Returns public URLs for easy access

https://cloudinary.com/

Cloudinary Setup

- Create an account at cloudinary.com.
- Get your credentials (Cloud name, API key, and API secret).
- Install Cloudinary SDK.

```
npm install cloudinary
```

```
import { v2 as cloudinary } from "cloudinary";

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_NAME,
  api_key: process.env.CLOUDINARY_KEY,
  api_secret: process.env.CLOUDINARY_SECRET,
});
```

Upload Image

Upload a local image file to Cloudinary.

```
import { v2 as cloudinary } from "cloudinary";
import fs from "fs";

export async function uploadImage(filePath: string) {
  try {
    const result = await cloudinary.uploader.upload(filePath, {
      folder: "profile_pictures",
    });
    fs.unlinkSync(filePath); // delete temporary file
    return result.secure_url;
  } catch (error) {
    console.error("Upload failed:", error);
  }
}
```

What is Nodemailer?

Nodemailer is a Node.js library for sending emails easily.

Common Use Cases:

- Welcome or verification emails
- Password resets
- Sending invoices or reports

https://nodemailer.com/

Create Transporter

Before sending emails, configure the mail server.

```
import nodemailer from "nodemailer";

export const transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS,
  },
});
```

- Use Gmail or SMTP server.
- Store credentials in `.env`.
- Create transporter once and reuse.

Send a Basic Email

```
async function sendMail() {
  const info = await transporter.sendMail({
    from: '"My App" <myapp@example.com>',
    to: "user@example.com",
    subject: "Welcome to My App!",
    text: "Thank you for joining us!",
  });

  console.log("Email sent:", info.messageId);
}

sendMail();
```

- Define sender and receiver.
- Add subject + message.
- Send using sendMail().

Send HTML Email

You can make emails more attractive using HTML templates.

```
async function sendHtmlEmail() {
  await transporter.sendMail({
    from: '"Support" <support@myapp.com>',
    to: "client@example.com",
    subject: "Order Confirmation",
    html: `
      <h2>Thank you for your order!</h2>
      <p>Your order is being processed.</p>
    `,
  });
}
```

- Use `html` instead of `text`.
- Embed inline styles, links, and images if needed.
- Test the email in multiple clients (Gmail, Outlook, etc.).

Exercise

Enhance your existing **Blog App** by integrating data validation, media upload, and email notification features to improve user experience and data integrity.

Exercise Objectives:

- Implement **input validation** using **Zod** to ensure all registration and post data meet the required schema before processing.
- Add a feature for users to **upload a profile picture** using a cloud storage service such as **Cloudinary**, and store the image URL in the database.
- Integrate **email notification** functionality using **Nodemailer** to send a welcome email after successful user registration.

Expected Outcome:
Users should be able to register with valid data, upload their profile photo, and receive a confirmation email automatically after registration.

Thank you