BookStore API — Complete Explanation (Challenge 1 to Mini Project)

Challenge 1: Setting Up Express Server

Approach: Install Express, create a basic server, and return a welcome message.

```
const express = require("express");
const app = express();
const port = 4000;

app.get("/", (req, res) => {
  res.send("Welcome to Express Server");
});

app.listen(port, () => console.log(`Server running on port ${port}`));
```

Challenge 2: Express Routing with Query Params

Approach: Read ?name= from URL and return JSON.

```
app.get("/products", (req, res) => {
  const name = req.query.name;
  res.json({ query: name });
});
```

Challenge 3: Custom Logging Middleware

Approach: Create middleware to log every HTTP request and apply globally.

Expected Output Example:

[GET] /products

[GET] /status

```
// Custom Logger Middleware
function logger(req, res, next) {
  console.log(`[${req.method}] ${req.url}`);
  next();
}

app.use(logger);
```

Challenge 4: CRUD REST API for Books

Approach: Use an in-memory array and implement GET, POST, PUT, DELETE.

```
// Initial data
let books = [
  { id: 1, title: "1984", author: "George Orwell" },
  { id: 2, title: "The Alchemist", author: "Paulo Coelho" }
];

// GET all books
app.get("/books", (req, res) => {
  res.json(books);
});

// POST add new book
app.post("/books", (req, res) => {
  const { title, author } = req.body;
  if (!title || !author) {
    return res.status(400).json({ error: "Title and author required" });
  }
  const newBook = { id: books.length + 1, title, author };
  books.push(newBook);
  res.json({ message: "Book added", newBook });
});

// PUT update book
app.put("/books/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const book = books.find(b => b.id === id);
  if (!book) return res.status(404).json({ error: "Book not found" });

  const { title, author } = req.body;
  book.title = title;
  book.author = author;

  res.json({ message: "Book updated", book });
});

// DELETE remove book
app.delete("/books/:id", (req, res) => {
  const id = parseInt(req.params.id);
  const index = books.findIndex(b => b.id === id);
  if (index === -1) return res.status(404).json({ error: "Book not found" });

  const deleted = books.splice(index, 1);
  res.json({ message: "Book deleted", deleted });
});
```

Challenge 5: Modular Routing & Error Handling

Approach: Move book routes to routes/books.js and add global error handlers.

```
// server.js
const bookRouter = require("./routes/books");

app.use("/books", bookRouter);

// 404 Middleware
app.use((req, res) => {
  res.status(404).json({ error: "Route not found" });
});

// Error Handler
app.use((err, req, res, next) => {
  console.error("Error:", err);
  res.status(500).json({ error: "Internal Server Error" });
});
```

Mini Project — Complete BookStore API

Approach: Combine all challenges + add CORS, validation, and GET /books/:id.

```
const cors = require("cors");
const { body, validationResult } = require("express-validator");

app.use(cors());
app.use(express.json());

router.get("/:id", (req, res) => {
  const book = books.find(b => b.id === parseInt(req.params.id));
  if (!book) return res.status(404).json({ error: "Book not found" });
  res.json(book);
});

router.post(
  "/",
  [body("title").notEmpty(), body("author").notEmpty()],
  (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    // Add book...
  }
);
```