

Copilot Workshop Labs

Hands-on exercises to master AI-Assisted Development

Start Labs →



The image shows a laptop screen with a code editor open. The code editor displays a file named `index.js` which contains Node.js code for a MongoDB application. The code includes imports for `express`, `Router`, and `User` from external files. It defines routes for `/register` (GET and POST methods) and handles user registration logic, including password matching and MongoDB insertion. The code editor interface shows line numbers, syntax highlighting, and a status bar indicating the file is a JavaScript file (JavaScript) with 0 files. The background of the slide features a blurred image of a laptop on a desk with a yellow mug.

```
Project
Coding
  100-days-of-code
    git
      FAQ.md
      log.md
      r1-log.md
      README.md
      resources.md
      rules.md
      atom-packages
    browser_persistence
    c01
    FlashcardsExpress
    freecodecamp_tribute
    JavaScript-Authentication
      git
      models
      public
      routes
        index.js
      views
      .gitignore
      app.js
      package.json
      README.md
    LocalWeatherFCC
    node-weather-zipcode
    nodeschool
    NodeWeather
    portfolio
    auth0Template
  log.md
  index.js

1 var express = require('express');
2 var router = express.Router();
3 var User = require('../models/user');
4
5 // GET /register
6 router.get('/register', function(req, res, next) {
7   return res.render('register', { title: 'Sign Up' });
8 });
9
10 // POST /register
11 router.post('/register', function(req, res, next) {
12   if (req.body.email &&
13     req.body.name &&
14     req.body.favoriteBook &&
15     req.body.password &&
16     req.body.confirmPassword) {
17     // If all inputs are valid, insert into database
18     if (req.body.password !== req.body.confirmPassword) {
19       var err = new Error('Passwords do not match.');
20       err.status = 400;
21       return next(err);
22     }
23     // Create object with form input
24     var userData = {
25       email: req.body.email,
26       name: req.body.name,
27       favoriteBook: req.body.favoriteBook,
28       password: req.body.password
29     };
30
31     // Use schema's 'create' method to insert document into Mongo
32     User.create(userData, function (error, user) {
33       if (error) {
34         return next(error);
35       }
36     });
37   }
38 });
39
40 // use schema's 'create' method to insert document into Mongo
41 User.create(userData, function (error, user) {
42   if (error) {
43     return next(error);
44   }
45 });
46
47 module.exports = router;
```

Lab Overview

We will cover the following practical exercises:

1. **Lab 01:** Basic Prompting & Algorithms
2. **Lab 02:** Web Server Generation
3. **Lab 03:** Data Manipulation & SQL
4. **Lab 04:** Unit Testing
5. **Lab 05:** Refactoring & Legacy Code
6. **Lab 06:** Using Copilot Chat

Lab 01: Basic Prompting

Lab 01: Algorithms

Goal: Use comment-driven development to generate algorithmic logic.

Instructions:

1. Create a file named `algorithms.js`.
2. Type the comment below.
3. Wait for the suggestion (Ghost text).
4. Press `Tab` to accept.

```
// Lab 01 Step 1
// Create a function that calculates the factorial of a number
// Handle negative numbers by returning null

// Lab 01 Step 2
// Create a function that checks if a string is a palindrome
// Ignore case and non-alphanumeric characters
```

Tip: If the suggestion isn't what you want, try pressing `Alt +]` (or `Option +]` on Mac) to cycle through alternatives.

Lab 02: Web Server

Lab 02: Express Server

Goal: Scaffold a complete web server file using a "Top-Down" prompt strategy.

Instructions:

1. Create a file named `server.js`.
2. Write a detailed block comment at the top of the file describing the requirements.

```
/*
Create an Express web server on port 3000.
It should have the following endpoints:
1. GET /api/users - returns a list of mock users
2. GET /api/users/:id - returns a specific user
3. POST /api/users - creates a new user
4. Middleware to parse JSON bodies
5. Error handling for 404 routes
*/
```

```
const express = require('express');
// Let Copilot generate the rest...
```

Lab 03: Data & SQL

Lab 03: SQL Queries

Goal: Generate complex SQL without memorizing syntax.

Instructions:

1. Create `queries.sql`.
2. Define your schema context (or assume a standard schema).
3. Ask for the query in natural language.

```
-- Table: Employees (id, name, department_id, salary, hire_date)
-- Table: Departments (id, name, location)
```

```
-- 1. Select the top 3 departments with the highest average
--   salary. Include the department name and the average salary rounded to
--   the nearest integer.
```

```
-- 2. Find employees hired in the last 6 months who earn more than
--   $50,000 per year. Order the results by salary in descending order.
```

Lab 03: Data Generation (Bonus)

Goal: Generate mock data for testing.

Instructions:

1. Create a JSON file `data.json` or a variable in JavaScript.
2. Start typing the structure and let Copilot fill the pattern.

```
const products = [
  { id: 1, name: "Laptop", price: 999, category: "Electronics" },
  // Press Enter and wait for Copilot to suggest the next item...
  // It should recognize the pattern and suggest relevant products.
];
```

Lab 04: Unit Testing

Lab 04: Generating Tests

Goal: Write a test suite for existing code.

Instructions:

1. Open the `algorithms.js` file from Lab 01.
2. Create a new file `algorithms.test.js`.
3. Import the functions.
4. Use natural language to describe the test cases.

```
const { factorial, isPalindrome } = require('./algorithms');

// Test suite for factorial function
// 1. Should return 120 for input 5
// 2. Should return 1 for input 0
// 3. Should return null for negative numbers
describe('factorial', () => {
    // Let Copilot fill the body
});
```

Lab 05: Refactoring

Lab 05: Improving Code

Goal: Use Copilot to modernize or clean up code.

Instructions:

1. Paste the "Legacy Code" into a file.
2. Highlight the code.
3. Open Copilot Chat (`Ctrl+I` or `Cmd+I`).
4. Type `/fix` or "Refactor this to use Arrow Functions and map".

Legacy Code Input:

```
function getNames(users) {  
  var names = [];  
  for (var i = 0; i < users.length; i++) {  
    if (users[i].age > 18) {  
      names.push(users[i].name);  
    }  
  }  
  return names;  
}
```

Lab 06: Copilot Chat

Lab 06: Explain & Document

Goal: Understand complex code and generate documentation.

Instructions:

1. Select a complex function or a Regular Expression.
2. Right-click > **Copilot** > **Explain This**.
3. Observe the explanation in the Chat panel.

Documentation:

1. Highlight a function.
2. Type `/doc` in the Chat input.
3. Copilot will generate JSDoc/DocString comments for the selected code.

Happy Coding!

Continue exploring the labs in the repository.

Workshop Repository
