# Debugging Guide

## VS Code Setup

This project includes VS Code debugging configurations for both server and client-side code.

### Files Created

1. `.vscode/launch.json` - Debug configurations
2. `.vscode/tasks.json` - Background tasks for starting servers

## Debugging Options

### 1. Debug Node.js Server Only

**Configuration:** "Debug Node.js Server"

- Click the Debug icon in VS Code sidebar (or press `Cmd+Shift+D` / `Ctrl+Shift+D`)
- Select "Debug Node.js Server" from the dropdown
- Press F5 or click the green play button
- Set breakpoints in your server-side code (`app.js`, `services.js`, `lib/**/*.js`)
- Server runs on `http://localhost:3000`

### 2. Debug Node.js Server with Auto-Restart

**Configuration:** "Debug Node.js Server (with nodemon)"

- Same as above but uses nodemon
- Server automatically restarts when you save files
- Breakpoints are preserved across restarts

### 3. Debug Client-Side Code (Chrome)

**Configuration:** "Debug Client (Chrome)"

- Automatically starts both servers (Node.js + Webpack dev server)
- Opens Chrome with debugging enabled
- Set breakpoints in your client-side code (`public/js/**/*.js`)
- Inspect variables, step through code, etc.

- Client runs on `http://localhost:3001`

**Also available for Microsoft Edge:** "Debug Client (Edge)"

### 4. Debug Full Stack (Server + Client)

**Configuration:** "Debug Full Stack" (Compound)

This is the recommended way to debug the entire application:

- Select "Debug Full Stack" from the debug dropdown
- Press F5
- Starts both Node.js server with nodemon AND Chrome debugger
- Debug both server and client simultaneously
- Set breakpoints anywhere in your codebase

## How to Use

### Setting Breakpoints

1. **Server-side** (`app.js`, `services.js`, files in `lib/`):
   - Click in the gutter (left of line numbers) to add a red dot
   - Or press F9 on the line you want to break at
2. **Client-side** (`public/js/client.js`):
   - Open the file in VS Code
   - Set breakpoints the same way
   - Or use Chrome DevTools (F12) for browser debugging

### Debug Controls

- **F5** - Continue
- **F10** - Step Over
- **F11** - Step Into
- **Shift+F11** - Step Out
- **Ctrl+Shift+F5** / **Cmd+Shift+F5** - Restart Debugger
- **Shift+F5** - Stop Debugger

### Debug Console

- View in VS Code: View → Debug Console (or `Cmd+Shift+Y` / `Ctrl+Shift+Y`)
- Evaluate expressions while paused at a breakpoint
- Type JavaScript expressions to inspect values

## Manual Debugging (Without VS Code)

### Server-Side Debugging

```bash
# Start Node.js with inspector
node --inspect app.js

# Or with nodemon
nodemon --inspect app.js

# Then open Chrome and navigate to:
chrome://inspect
```

### Client-Side Debugging

```bash
# Start both servers
npm run dev

# Open Chrome DevTools (F12)
# Navigate to Sources tab
# Find your files under webpack://
# Set breakpoints directly in Chrome
```

## Debugging Tips

### Server-Side

1. **Use console.log()** strategically
2. **Inspect async operations**: Use breakpoints in `async` functions and `.then()` chains
3. **Check database queries**: Add breakpoints in repository files (`lib/db/*.js`)
4. **Monitor API requests**: Set breakpoints at the start of route handlers in `app.js`

### Client-Side

1. **Use browser DevTools**: F12 for Chrome/Edge DevTools
2. **Network tab**: Monitor API requests and responses
3. **Console errors**: Check for JavaScript errors
4. **Inspect `window._client`**: Type `window._client` in console to inspect the client instance
5. **Source maps**: Webpack generates source maps so you see original code, not bundled code

## Common Scenarios

### Debug API Request Flow

1. Set breakpoint in `app.js` at the route handler (e.g., line for `app.get('/api/recipes')`)
2. Set breakpoint in `services.js` method (e.g., `getAllRecipes()`)
3. Set breakpoint in repository file (e.g., `lib/db/RecipeRepository.js`)
4. Trigger the request from the frontend
5. Step through the entire flow

### Debug Client-Side Click Handler

1. Set breakpoint in `client.js` in the method (e.g., `createRecipe()`)
2. Click the button in the UI
3. Step through the code
4. Watch the API call being made
5. See the response being processed

### Debug Form Submission

1. Set breakpoint in form submission handler (e.g., `saveRecipe()`)
2. Set breakpoint in the API route handler (`app.post('/api/recipes')`)
3. Submit the form
4. Step through validation, data processing, and database operations

# Troubleshooting

## Breakpoints Not Hit

- Make sure you're running the debug configuration (green "Debugging" status bar)
- Check that source maps are enabled in webpack config
- For client-side: Make sure you're debugging on `http://localhost:3001` (not 3000)
- Reload the page after setting breakpoints

### "Cannot find module" errors

- Make sure all imports include `.js` extension (ES modules requirement)
- Check that file paths are correct
- Run `npm install` to ensure all dependencies are installed

### Webpack Dev Server Issues

- Kill any processes on port 3001: `lsof -ti:3001 | xargs kill -9` (Mac/Linux)
- Clear webpack cache: `npm run clean`
- Restart the debug session

### Port Already in Use

```bash
# Kill process on port 3000 (Node.js server)
lsof -ti:3000 | xargs kill -9

# Kill process on port 3001 (Webpack dev server)
lsof -ti:3001 | xargs kill -9

# Windows:
netstat -ano | findstr :3000
taskkill /PID <PID> /F
```

## Additional Resources

- VS Code Debugging
- Node.js Debugging Guide
- Chrome DevTools
- Webpack Debugging