# Dependency Management

> Relevant source files

This document explains the npm workspaces-based dependency management system used in the IronHack Course 2 monorepo. It covers how dependencies are coordinated across the three projects, package-lock file management, and execution strategies for managing multiple Node.js applications within a single repository.

For information about the VS Code workspace configuration and debugging setup, see VS Code & Debugging. For details about the individual project architectures, see Project 1: Product Management, Project 2: Course Platform, and Project 3: Food Delivery Platform.

## Monorepo Workspace Architecture

The repository implements a npm workspaces-based monorepo structure that coordinates three independent Node.js applications while maintaining dependency isolation and enabling unified execution commands.


### Root Package Configuration

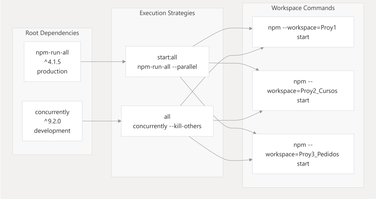
The monorepo uses npm workspaces to manage three distinct projects while maintaining a unified dependency resolution strategy.

Sources: package.json 13-16 package.json 6-12

## Root-Level Dependency Coordination

The root `package.json` defines workspace boundaries and provides orchestration tools for managing multiple applications simultaneously.

| Configuration | Value | Purpose |
|---|---|---|
| `workspaces` | `["Proy1", "Proy2_Cursos", "Proy3_Pedidos"]` | Defines workspace members |
| `lockfileVersion` | `3` | npm lockfile format version |
| `concurrently` | `^9.2.0` | Parallel execution tool |
| `npm-run-all` | `^4.1.5` | Sequential/parallel script runner |




### Execution Commands

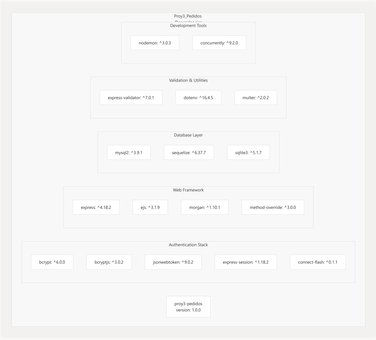
The root package provides multiple strategies for running projects:

- `start:proy1|2|3` : Individual project execution using workspace targeting
- `start:all` : Parallel execution using `npm-run-all`
- `all` : Parallel execution with process management using `concurrently`

Sources: package.json 6-12 package.json 20-25

## Project-Level Dependency Isolation

Each workspace maintains its own dependency tree while leveraging npm's workspace dependency resolution for shared packages.




### Project Dependency Characteristics

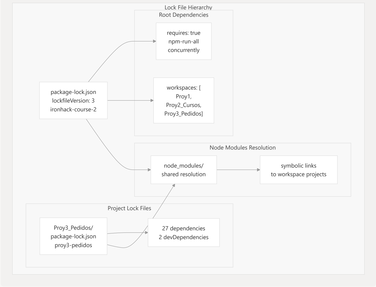
Each project demonstrates different levels of dependency complexity:

- **Proy1**: Basic Express setup with SQLite
- **Proy2_Cursos**: Express + Sequelize + MySQL with testing
- **Proy3_Pedidos**: Comprehensive stack with multiple authentication methods

Sources: Proy3_Pedidos/package-lock.json 7-33

## Package Lock File Management

The monorepo maintains both root-level and project-level package-lock files to ensure consistent dependency resolution across environments.




### Lock File Strategy

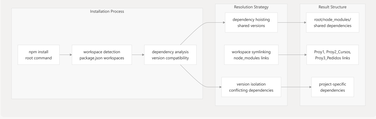
The repository uses a dual lock file approach:

1. **Root lock file**: Manages workspace orchestration dependencies
2. **Project lock files**: Ensure reproducible builds for individual applications
3. **Shared resolution**: npm workspaces creates symbolic links in root `node_modules`

Sources: package-lock.json 1-26 Proy3_Pedidos/package-lock.json 1-33

## Dependency Resolution Flow

npm workspaces provides automatic dependency hoisting and sharing while maintaining project isolation for conflicting versions.




### Workspace Benefits

The npm workspaces implementation provides:

- **Unified dependency resolution**: Shared packages installed once at root level
- **Development efficiency**: Single `npm install` for entire monorepo
- **Project isolation**: Conflicting versions resolved per workspace
- **Script coordination**: Workspace-aware npm commands for targeted execution

Sources: package.json 13-16 package-lock.json 11-16