Ultimate Dependency Manager

Project Documentation

Author: GIRISH KOR
Version: 0.1.4
Last Updated: March 21, 2025

Project Overview

The Ultimate Dependency Manager is a VS Code extension that simplifies dependency management within the Visual Studio Code environment. It provides an intuitive interface for installing, updating, and auditing project dependencies without needing to use the command line.

Table of Contents

Introduction

Problem Statement

Modern web development relies heavily on package managers like npm, yarn, pnpm, and bun to handle dependencies. Switching between the code editor and terminal to manage these dependencies breaks the developer flow and decreases productivity.

Solution

The Ultimate Dependency Manager extension integrates dependency management directly into VS Code's interface, allowing developers to:

- Install, update, and remove packages without leaving their editor
- Visualize dependency relationships
- Quickly identify and fix security vulnerabilities
- Monitor outdated packages
- Manage dependencies across different package managers

Features

Core Functionality

- **Multi-Package Manager Support**: Automatic detection and support for npm, yarn, pnpm, and bun
- **Dependency Installation**: One-click installation of all project dependencies
- **Dependency Updates**: Easy updating of outdated packages
- **Dependency Removal**: Simple removal of unused or unnecessary dependencies
- **Security Auditing**: Built-in vulnerability scanning and automatic fixes
- **Dependency Tree Visualization**: Graphical representation of project dependencies

User Interface Components

- **Status Bar Integration**: Quick access from the VS Code status bar
- **Tree View**: Hierarchical view of dependencies in the Explorer panel
- **Webview Panel**: Interactive dashboard for managing dependencies
- **Command Palette Integration**: Access to all functionality through VS Code's command palette

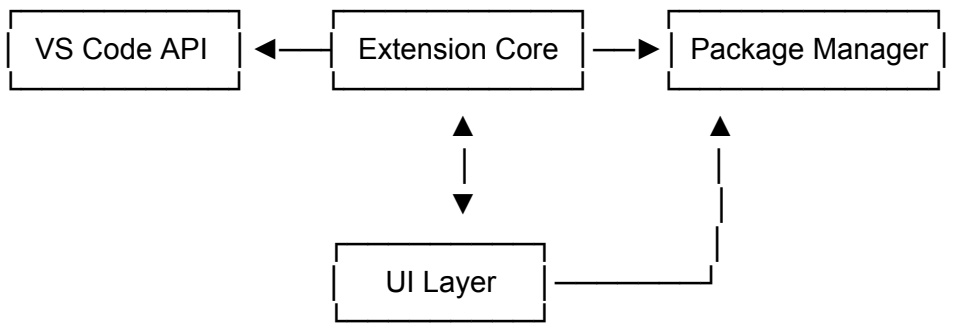
Technical Architecture

High-Level Architecture

The extension follows a modular architecture with the following key components:

1. **Activation Handler**: Initializes the extension when a workspace with a package.json file is opened
2. **Package Manager Detector**: Identifies the correct package manager based on lock files
3. **Command Executor**: Safely executes package manager commands
4. **UI Components**:
    - Status Bar Item
    - Tree View Provider
    - Webview Panel
5. **Dependency Analyzer**: Identifies outdated, vulnerable, and unused dependencies

Component Interactions



Implementation Details

Extension Activation

The extension is activated when a workspace containing a package.json file is opened. The activation function sets up all necessary components:

```
async function activate(context) {
  // Create output channel for logs
  const outputChannel = vscode.window.createOutputChannel('Ultimate Dep Manager');

  // Set up status bar item
  const statusBar = vscode.window.createStatusBarItem(vscode.StatusBarAlignment.Right, 100);
  statusBar.text = '(♧)';
  statusBar.tooltip = 'Ultimate Dependency Management';
  statusBar.command = 'ultimateDep.showPanel';
  statusBar.show();

  // Register tree data provider
  const treeDataProvider = new DependencyTreeProvider();
  vscode.window.registerTreeDataProvider('dependencyExplorer', treeDataProvider);

  // Register commands
  context.subscriptions.push(
    vscode.commands.registerCommand('ultimateDep.showPanel', showWebviewPanel),
    vscode.commands.registerCommand('ultimateDep.install', handleDependencyAction),
    // Additional commands...
  );
}
```

Package Manager Detection

The extension automatically detects the package manager by checking for lock files:

```
const detectPackageManager = async () => {
  const workspaceRoot = getWorkspaceRoot();
  for (const [pm, lockFile] of Object.entries(PM_LOCK_FILES)) {
    if (fs.existsSync(path.join(workspaceRoot, lockFile))) {
      return pm;
    }
  }
  return 'npm'; // Default fallback
};
```

Command Execution

Commands are executed in a child process with progress notification:

```
const executeCommand = command => {
  return vscode.window.withProgress(
    {
      location: vscode.ProgressLocation.Notification,
      title: `Running ${command}`,
      cancellable: false,
    },
    async () => {
      return new Promise((resolve, reject) => {
        const child = exec(command, { cwd: getWorkspaceRoot() }, (error, stdout, stderr) => {
          if (error) reject(error);
          else resolve({ stdout, stderr });
```

```
        });
        outputChannel.appendLine(`Running: ${command}`);
        child.stdout.on('data', data => outputChannel.append(data));
        child.stderr.on('data', data => outputChannel.append(data));
      });
    }
  );
};
```

Tree View Implementation

The tree view is powered by a custom TreeDataProvider:

```
class DependencyTreeProvider {
  constructor() {
    this._onDidChangeTreeData = new vscode.EventEmitter();
    this.onDidChangeTreeData = this._onDidChangeTreeData.event;
  }

  getTreeItem(element) {
    return element;
  }

  refresh() {
    this._onDidChangeTreeData.fire();
  }

  async getChildren(element) {
    return element ? this.getDependencyNodes(element) : this.getRootNodes();
  }

  // Additional methods...
}
```

User Interface

Webview Panel

The extension provides a rich interactive dashboard through a webview panel:

```
const getWebviewContent = () => {
  return `
  <!DOCTYPE html>
  <html>
  <head>
    <style>
      /* CSS styles omitted for brevity */
    </style>
  </head>
  <body>
    <div class="container">
      <h1>Ultimate Dependency Manager</h1>
      <div class="button-group">
        <button onclick="handleCommand('install')">
          Install All
        </button>
        <button onclick="handleCommand('update')">
          Update All
        </button>
```

```
    </div>
    <div class="status-section">
      <!-- Status section content -->
    </div>
  </div>
  <script>
    // JavaScript omitted for brevity
  </script>
 </body>
 </html>
`;
};
```

Tree View

The tree view organizes dependencies into three main categories:

- Outdated Dependencies
- Security Vulnerabilities
- Unused Dependencies

Each category can be expanded to show specific packages.

Status Bar

The extension adds a status bar item that serves as a quick entry point to the extension's functionality.

Extension Capabilities

Commands

The extension contributes the following commands to VS Code:

Command	Title	Description
`ultimateDep.showPanel`	Show Ultimate Dependency Manager	Opens the main webview panel
`ultimateDep.install`	Install Dependencies	Installs project dependencies
`ultimateDep.uninstall`	Uninstall Dependency	Removes a specific dependency
`ultimateDep.update`	Update Dependencies	Updates outdated dependencies
`ultimateDep.auditFix`	Run Security Audit	Scans and fixes security issues
`ultimateDep.refresh`	Refresh Dependency Tree	Refreshes the dependency view

Views

The extension contributes a custom view to the Explorer viewlet:

ID	Name	Description
`dependencyExplorer`	Dependencies	Shows the dependency tree

Menus

The extension adds context menu items to the tree view:

- Refresh button in the view title
- Update action for outdated dependencies
- Audit Fix action for security vulnerabilities

Testing Strategy

Unit Tests
Unit tests cover individual components:
- Package manager detection
- Command execution
- Tree data provider
- Webview content generation

Integration Tests
Integration tests verify:
- Extension activation
- Command registration
- UI component interaction
- Package manager operations

End-to-End Tests
E2E tests validate complete user workflows:
- Installing dependencies
- Updating packages
- Running security audits
- Uninstalling dependencies

Deployment Process

Packaging
The extension is packaged using the VS Code Extension Manager (vsce):
vsce package

This creates a `.vsix` file that can be installed directly or published to the VS Code Marketplace.

Publishing
Publishing to the VS Code Marketplace follows these steps:
1. Update version in package.json
2. Generate changelog entry
3. Package the extension
4. Publish using the Azure DevOps token

vsce publish -p <token>

Distribution Channels
The extension is available through:
- VS Code Marketplace
- Open VSX Registry
- GitHub Releases

Future Enhancements
Planned Features
1. **Advanced Dependency Visualization**: Interactive dependency graphs
2. **Dependency Size Analysis**: Identify large dependencies
3. **Custom Package Registry Support**: Use with private or custom registries
4. **Dependency Usage Tracking**: Identify how dependencies are used in code
5. **Dependency Documentation Integration**: Quick access to package documentation

Technical Improvements

1. **Performance Optimization**: Faster dependency analysis
2. **Better Error Handling**: More robust recovery from package manager errors
3. **Enhanced Security Analysis**: More detailed vulnerability information
4. **Workspace Support**: Better handling of multi-root workspaces
5. **Telemetry**: Optional usage tracking for improvement insights

Appendices

A. Extension Settings

Setting	Type	Default	Description
`ultimateDep.preferredPackageManager`	string	auto	Override automatic package manager detection
`ultimateDep.showStatusBarItem`	boolean	true	Show/hide the status bar item
`ultimateDep.automaticUpdateCheck`	boolean	true	Check for outdated packages on startup

B. Dependencies

The extension relies on the following dependencies:

- Core VS Code Extension API
- Node.js utilities
- depcheck for unused dependency analysis

C. References

- VS Code Extension API Documentation
- npm CLI Documentation
- Yarn CLI Documentation
- pnpm CLI Documentation
- Bun CLI Documentation