# ToolBatcher - Cross Platform Development Tool Manager

## Abstract

ToolBatcher is a comprehensive web application designed to streamline the development environment setup process across different operating systems. It provides an intuitive interface for developers to manage and install development tools, receive installation commands based on their operating system, and share feedback. The application simplifies the often-complex process of setting up development environments by providing verified installation commands and maintaining a consistent toolset across different platforms.

## Introduction

This web application is designed to streamline the process of installing multiple command-line tools across different operating systems by generating a custom batch installation script. Users can select the tools they need, and the site will dynamically create an installation script compatible with their chosen OS, whether it be Linux, macOS, or Windows. This platform is ideal for developers, IT professionals, and students who frequently set up development environments and want a faster, more efficient way to configure multiple tools simultaneously. By utilizing a user-friendly, responsive interface built with React and styled with TailwindCSS, the application provides an accessible solution for tool management and installation.

## Motivation

he motivation behind this application is to simplify the often-cumbersome process of installing command-line tools on new or reset systems. Installing tools individually can be time-consuming, especially when dealing with different commands for multiple operating systems. This platform was conceived to address this need by centralizing the tool-selection process, generating installation scripts, and allowing users to copy or download them for immediate use. Additionally, by enabling real-time updates and customizable installations, the app saves time and reduces potential errors, giving users a smoother setup experience across devices and operating systems.

## Modules and Functionalities

### 1. Frontend Modules

#### Core Components

* **Navbar**: The Navbar provides navigation links to essential sections of the application, including Home, Tool Selector, Documentation, Feedback, and Admin. It’s designed to be fully responsive, using TailwindCSS to adjust seamlessly across various screen sizes. Depending on the device, it may present links as dropdowns or side menus to maintain usability.
* **Hero**: The Hero component serves as the primary visual section on the landing page, showcasing core features like “Select, Generate, and Install.” It includes a call-to-action button leading to the Tool Selector page and may feature a brief animation or visual elements to engage users immediately. This area emphasizes the main purpose of the website, drawing users in with a clear, concise message.
* **ToolSelector**: ToolSelector is an interactive component allowing users to choose multiple command-line tools across different operating systems. It features a real-time update of selected tool commands pulled from MongoDB, filtering them by OS (Linux, macOS, Windows). Users can view details for each tool, with their selections updated live in the generated script.
* **CodeEditor**: The CodeEditor displays the generated installation script, formatted for specific OS requirements (.sh for Linux/macOS, .ps1 for Windows). It offers syntax highlighting for readability, a “Copy to Clipboard” button, and an option to download the script. This component ensures users can easily access and save their generated commands for future use.
* **FeedbackForm**: The FeedbackForm collects user feedback on their experience with the application, including ratings, tool suggestions, and comments. Form validation ensures complete, valid input and prevents submission errors. This component helps gather insights for potential improvements and allows users to directly influence future updates.
* **Documentation**: The Documentation section provides detailed usage guidelines, troubleshooting steps, and an FAQ to help users understand the application’s functionalities. It also includes links to official documentation for each command-line tool. This section is intended as a comprehensive resource for both new and experienced users.
* **AdminPage**: AdminPage is a secured area, requiring authentication to manage tool data and view feedback. It allows admins to add, update, or delete commands in MongoDB and organize user feedback for analysis. Additionally, it provides options to categorize user suggestions, making data management more efficient.

#### Features

* Responsive design using TailwindCSS
* Cross-browser compatibility
* Interactive UI elements
* Real-time command generation
* Copy-to-clipboard functionality
* Form validation
* Admin dashboard

### 2. Backend Modules

#### API Controllers

* **toolController**: Manages tool-related operations
* **feedbackController**: Handles feedback submission and retrieval

#### Data Models

* **ToolCommand**: Schema for tool installation commands
* **Feedback**: Schema for user feedback storage

#### Routes

* **/api/tools**: Tool management endpoints
* **/api/feedback**: Feedback management endpoints

## Software Stack

### Frontend

* React.js (v18)
* Vite.js
* TailwindCSS
* PostCSS

### Backend

* Node.js
* Express.js
* MongoDB
* Mongoose ODM

### Development Tools

* Git
* npm/yarn
* VS Code
* MongoDB Compass

This software stack combines powerful and modern technologies that streamline both frontend and backend development. The choice of tools enhances collaboration, improves code quality, and accelerates the development process, making it suitable for building robust and scalable applications. Whether it's crafting an engaging user interface with React and TailwindCSS or setting up a RESTful API with Node.js and Express, this stack is well-equipped to handle a wide range of project requirements.

## Directory Architecture

├── .gitignore

├── .vscode/

│ └── settings.json

├── backend/

│ ├── .env

│ ├── config/

│ │ └── database.js

│ ├── controllers/

│ ├── models/

│ ├── routes/

│ ├── index.js

│ └── package.json

├── frontend/

│ ├── dist/

│ ├── public/

│ ├── src/

│ │ ├── components/

│ │ ├── App.jsx

│ │ └── main.jsx

│ ├── tailwind.config.cjs

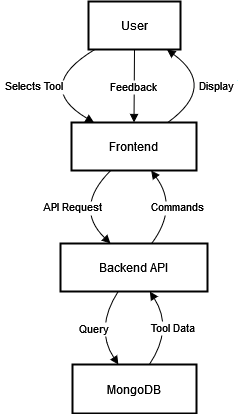
│ └── vite.config.js

├── package.json

└── README.md

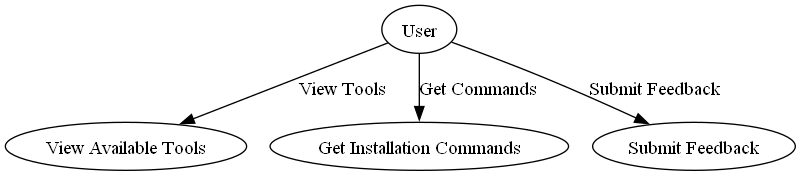
## System Architecture Diagrams

### Data Flow Diagram



### Use Case Diagram

#### User Module:



#### Admin Module:

### 

## Database Schema

### ToolCommand Collection

{  
 toolName: String,  
 versions: [String],  
 commands: {  
 linux: String,  
 macos: String,  
 windows: String  
 }  
}

### Feedback Collection

{  
 userId: String,  
 toolName: String,  
 rating: Number,  
 comments: String,  
 timestamp: Date  
}

## Source Code

### 1. Backend API Setup (index.js)

const express = require('express');  
const cors = require('cors');  
const connectDB = require('./config/database');  
const toolRoutes = require('./routes/tools');  
const feedbackRoutes = require('./routes/feedback');  
  
const app = express();  
  
// Connect to MongoDB  
connectDB();  
  
const corsOptions = {  
 origin: 'http://localhost:3000',  
 optionsSuccessStatus: 200  
};  
app.use(cors(corsOptions));  
  
app.use(express.json());  
  
// Routes  
app.use('/api/tools', toolRoutes);  
app.use('/api/feedback', feedbackRoutes);  
  
const PORT = process.env.PORT || 3002;  
app.listen(PORT, () => {  
 console.log(`Backend running on port ${PORT}`);  
});

### 2. Frontend Tool Selector Component

import React, { useState, useEffect } from 'react';  
import axios from 'axios';  
  
const ToolSelector = () => {  
 const [tools, setTools] = useState([]);  
 const [selectedTool, setSelectedTool] = useState(null);  
 const [commands, setCommands] = useState({});  
 const [os, setOs] = useState('windows');  
  
 useEffect(() => {  
 const fetchTools = async () => {  
 try {  
 const response = await axios.get('http://localhost:3002/api/tools');  
 setTools(response.data);  
 } catch (error) {  
 console.error('Error fetching tools:', error);  
 }  
 };  
 fetchTools();  
 }, []);  
  
 const handleToolSelect = async (tool) => {  
 setSelectedTool(tool);  
 try {  
 const response = await axios.get(`http://localhost:3002/api/tools/${tool.\_id}`);  
 setCommands(response.data.commands);  
 } catch (error) {  
 console.error('Error fetching commands:', error);  
 }  
 };  
  
 return (  
 <div className="container mx-auto p-4">  
 <div className="mb-4">  
 <select   
 onChange={(e) => setOs(e.target.value)}  
 className="border p-2 rounded"  
 >  
 <option value="windows">Windows</option>  
 <option value="macos">MacOS</option>  
 <option value="linux">Linux</option>  
 </select>  
 </div>  
   
 <div className="grid grid-cols-3 gap-4">  
 {tools.map((tool) => (  
 <div   
 key={tool.\_id}  
 className={`p-4 border rounded cursor-pointer ${  
 selectedTool?.\_id === tool.\_id ? 'bg-blue-100' : ''  
 }`}  
 onClick={() => handleToolSelect(tool)}  
 >  
 <h3 className="font-bold">{tool.toolName}</h3>  
 <p>Versions: {tool.versions.join(', ')}</p>  
 </div>  
 ))}  
 </div>  
   
 {selectedTool && commands[os] && (  
 <div className="mt-4 p-4 bg-gray-100 rounded">  
 <h4 className="font-bold mb-2">Installation Command:</h4>  
 <pre className="bg-black text-white p-2 rounded">  
 {commands[os]}  
 </pre>  
 </div>  
 )}  
 </div>  
 );  
};  
  
export default ToolSelector;

### 3. Database Configuration

const mongoose = require('mongoose');  
  
const connectDB = async () => {  
 try {  
 await mongoose.connect('mongodb://localhost:27017/toolbatcher', {  
 useNewUrlParser: true,  
 useUnifiedTopology: true,  
 });  
 console.log('MongoDB connected successfully');  
 } catch (error) {  
 console.error('MongoDB connection error:', error);  
 process.exit(1);  
 }  
};  
  
module.exports = connectDB;

### 4. API Routes Implementation

const express = require('express');  
const router = express.Router();  
const ToolCommand = require('../models/ToolCommand');  
  
// Get all tools  
router.get('/', async (req, res) => {  
 try {  
 const tools = await ToolCommand.find();  
 res.json(tools);  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
});  
  
// Get specific tool  
router.get('/:id', async (req, res) => {  
 try {  
 const tool = await ToolCommand.findById(req.params.id);  
 if (tool) {  
 res.json(tool);  
 } else {  
 res.status(404).json({ message: 'Tool not found' });  
 }  
 } catch (error) {  
 res.status(500).json({ message: error.message });  
 }  
});  
  
// Add new tool  
router.post('/', async (req, res) => {  
 const tool = new ToolCommand({  
 toolName: req.body.toolName,  
 versions: req.body.versions,  
 commands: req.body.commands  
 });  
  
 try {  
 const newTool = await tool.save();  
 res.status(201).json(newTool);  
 } catch (error) {  
 res.status(400).json({ message: error.message });  
 }  
});  
  
module.exports = router;

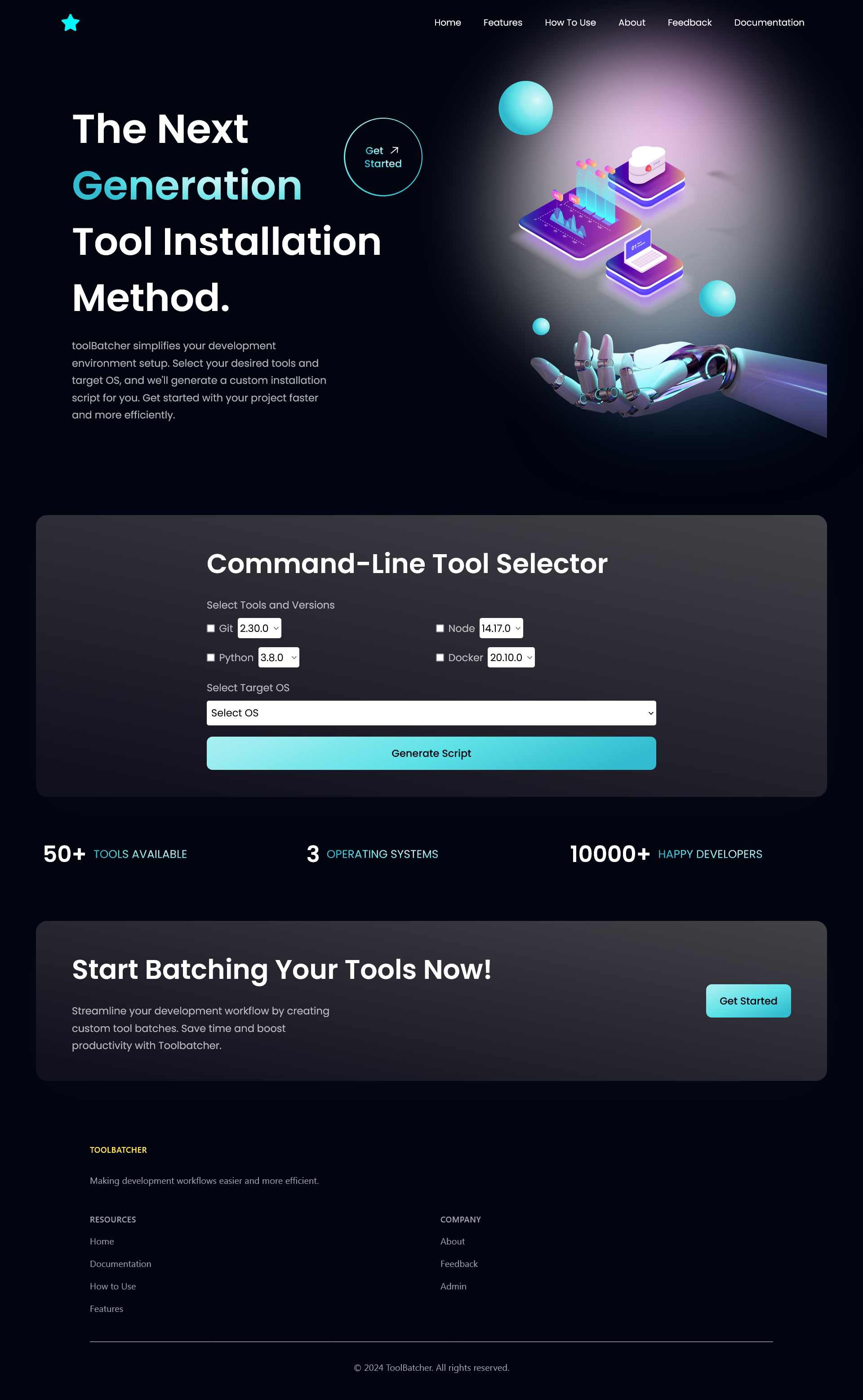
## Output

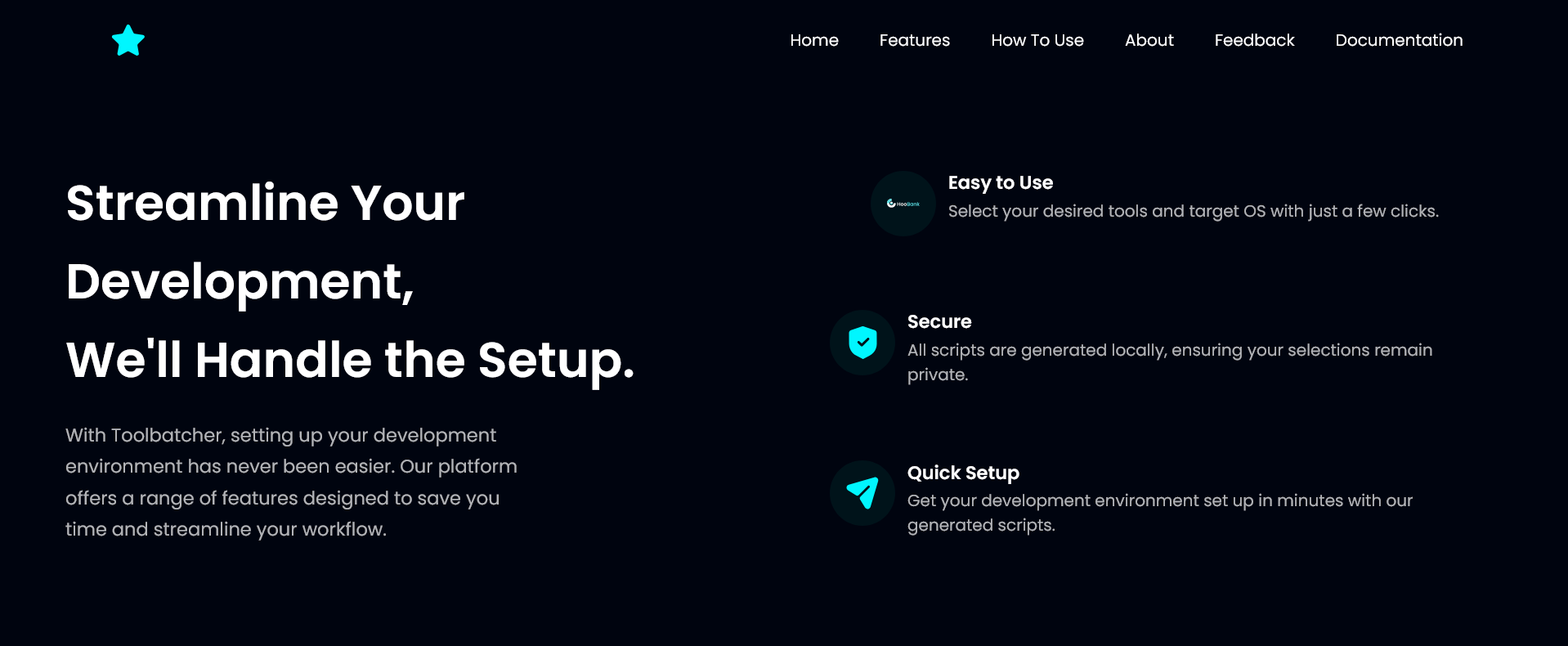
### MongoDB Screenshots



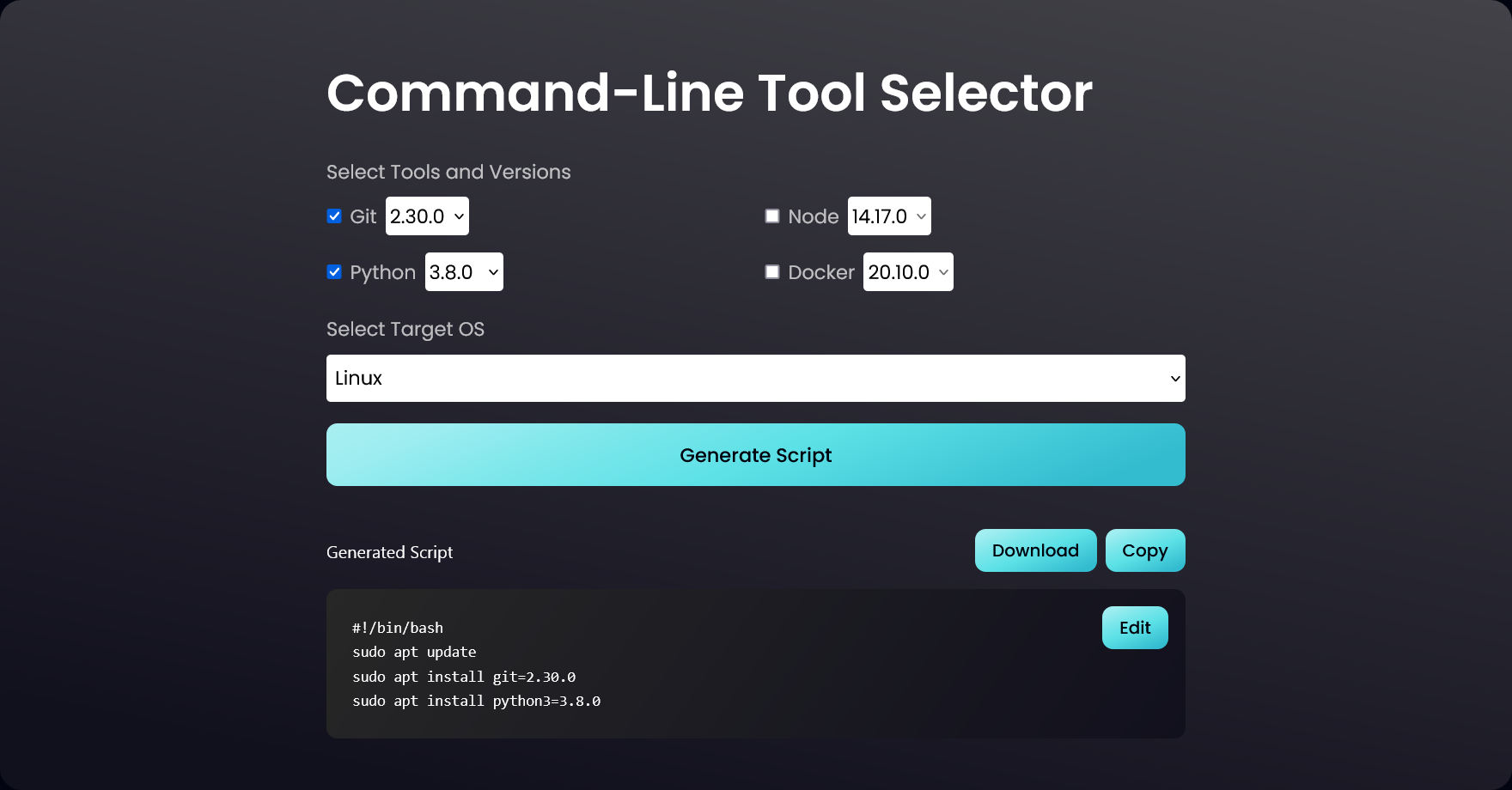
****

**Frontend Interface**

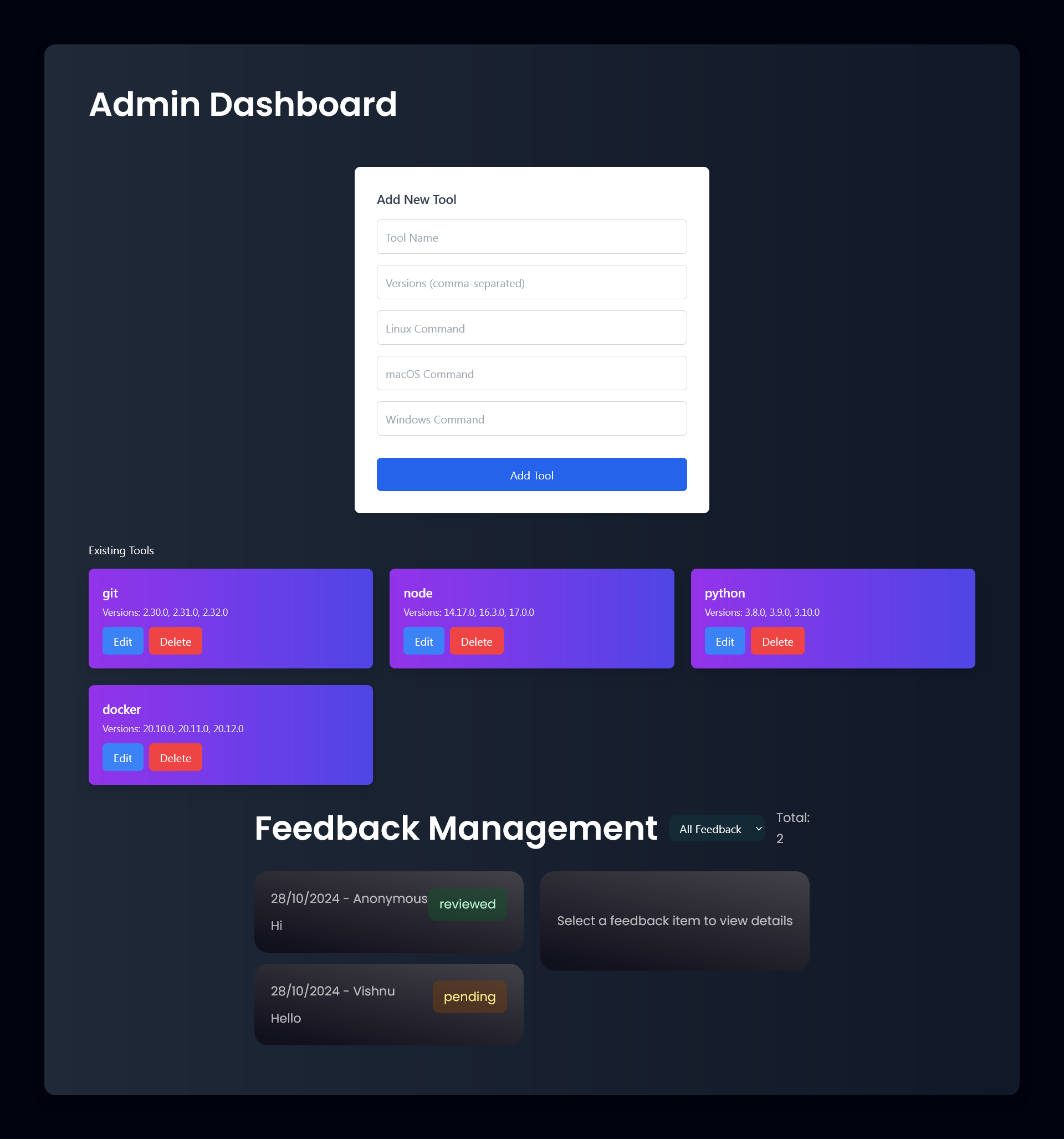




**Tool Batcher**



**Admin Panel**



**References**

Website reference – https://ninite.com/