🚀 Vue + Shadcn/Vue ESP32 Palletizer - Complete Setup Guide

📋 Project Overview

Building a modern, industrial-grade web interface for ESP32 Palletizer Control System using:

- **Vue 3** (Composition API)
- **Shadcn/Vue** (Zero CSS framework)
- **Tailwind CSS** (Utility-first styling)
- **Vite** (Modern build tool)
- **Single HTML deployment** (ESP32 optimized)

🎯 Project Goals

- ✅ **Zero manual CSS** - Shadcn handles all styling
- ✅ **Modern industrial UI** - Professional control panel look
- ✅ **Small bundle size** - Optimized for ESP32 memory constraints
- ✅ **Clean architecture** - Separated business logic from UI
- ✅ **Dark mode ready** - Built-in theme switching
- ✅ **Mobile responsive** - Touch-friendly for embedded displays

---

📁 Final Project Structure

```
esp32-palletizer-vue/
├── 📁 public/
│   └── index.html                 # Base HTML template
├── 📁 src/
│   ├── 📁 components/
│   │   ├── 📁 ui/                 # Shadcn components (auto-generated)
│   │   │   ├── button.vue
│   │   │   ├── card.vue
│   │   │   ├── input.vue
│   │   │   ├── slider.vue
│   │   │   ├── badge.vue
│   │   │   ├── textarea.vue
│   │   │   ├── tabs.vue
│   │   │   ├── switch.vue
│   │   │   └── alert.vue
│   │   ├── 📁 layout/             # Layout components
│   │   │   ├── AppHeader.vue
│   │   │   ├── AppSidebar.vue
│   │   │   └── AppLayout.vue
│   │   └── 📁 features/           # Feature-specific components
│   │       ├── 📁 system-control/
│   │       │   ├── SystemControls.vue
│   │       │   └── SystemStatus.vue
│   │       ├── 📁 speed-control/
│   │       │   ├── SpeedMatrix.vue
│   │       │   ├── SpeedSlider.vue
│   │       │   └── AxisControl.vue
│   │       ├── 📁 command-editor/
│   │       │   ├── CommandEditor.vue
│   │       │   ├── FileUpload.vue
│   │       │   └── CommandHistory.vue
│   │       └── 📁 monitoring/
│   │           ├── SystemMonitor.vue
│   │           ├── ConnectionStatus.vue
│   │           └── ErrorDisplay.vue
│   ├── 📁 composables/            # Business logic (Vue's custom hooks)
│   │   ├── useSystemControl.js    # System state & commands
│   │   ├── useSpeedControl.js     # Speed management
│   │   ├── useCommandEditor.js    # Command editing logic
│   │   ├── useWebSocket.js        # Real-time communication
│   │   ├── useFileUpload.js       # File handling
│   │   ├── useTheme.js           # Dark mode management
│   │   └── useApi.js             # API integration
│   ├── 📁 services/               # API & External services
│   │   ├── api.js                # REST API calls
│   │   ├── websocket.js          # WebSocket connection
│   │   └── fileService.js        # File operations
│   ├── 📁 utils/                  # Helper functions
│   │   ├── constants.js          # App constants
│   │   ├── validators.js         # Input validation
│   │   └── formatters.js         # Data formatting
│   ├── 📁 assets/                 # Static assets
│   │   └── index.css             # Global styles + Tailwind
│   ├── 📁 lib/                    # Shadcn utilities
│   │   └── utils.js              # Shadcn helper functions
│   │
│   ├── App.vue                 # Root component
│   └── main.js                 # Application entry point
├── 📁 dist/                    # Build output (after build)
├── build-esp32.js              # ESP32 deployment script
├── components.json             # Shadcn configuration
├── tailwind.config.js          # Tailwind CSS config
├── vite.config.js              # Vite build configuration
├── package.json                # Dependencies & scripts
├── .eslintrc.cjs               # ESLint configuration
├── .prettierrc                 # Prettier configuration
└── README.md                   # Project documentation
```

🛠️ Step 1: Project Initialization

Create Vue Project

```bash
# Create new Vue project
npm create vue@latest esp32-palletizer-vue

# Configuration options:
✅ Project name: esp32-palletizer-vue
❌ TypeScript? No (keep simple for embedded)
❌ JSX Support? No
❌ Vue Router? No (single page app)
❌ Pinia for state management? No (use composables)
❌ Vitest for Unit Testing? No
❌ End-to-End Testing? No
✅ ESLint for code quality? Yes
✅ Prettier for code formatting? Yes

# Navigate to project
cd esp32-palletizer-vue
```

Initial Dependencies Install

```bash
# Install base dependencies
npm install

# Install Vue ecosystem
npm install @vueuse/core

# Install HTTP client (optional, can use fetch)
npm install axios
```

🎨 Step 2: Tailwind CSS Setup

Install Tailwind

```bash
# Install Tailwind CSS and plugins
npm install -D tailwindcss postcss autoprefixer
npm install -D @tailwindcss/forms @tailwindcss/typography

# Initialize Tailwind config
npx tailwindcss init -p
```

Configure Tailwind

tailwind.config.js:

```
javascript
```

```
/** @type {import('tailwindcss').Config} */
export default {
  darkMode: ['class'],
  content: [
    './index.html',
    './src/**/*.{vue,js,ts,jsx,tsx}',
  ],
  theme: {
    extend: {
      colors: {
        // Shadcn color system
        border: 'hsl(var(--border))',
        input: 'hsl(var(--input))',
        ring: 'hsl(var(--ring))',
        background: 'hsl(var(--background))',
        foreground: 'hsl(var(--foreground))',
        primary: {
          DEFAULT: 'hsl(var(--primary))',
          foreground: 'hsl(var(--primary-foreground))'
        },
        secondary: {
          DEFAULT: 'hsl(var(--secondary))',
          foreground: 'hsl(var(--secondary-foreground))'
        },
        destructive: {
          DEFAULT: 'hsl(var(--destructive))',
          foreground: 'hsl(var(--destructive-foreground))'
        },
        muted: {
          DEFAULT: 'hsl(var(--muted))',
          foreground: 'hsl(var(--muted-foreground))'
        },
        accent: {
          DEFAULT: 'hsl(var(--accent))',
          foreground: 'hsl(var(--accent-foreground))'
        },
        popover: {
          DEFAULT: 'hsl(var(--popover))',
          foreground: 'hsl(var(--popover-foreground))'
        },
        card: {
          DEFAULT: 'hsl(var(--card))',
          foreground: 'hsl(var(--card-foreground))'
        },
        // Industrial color palette
        success: {
          DEFAULT: 'hsl(142 76% 36%)',
          foreground: 'hsl(355 7% 97%)'
        },
        warning: {
          DEFAULT: 'hsl(33 95% 54%)',
          foreground: 'hsl(0 0% 9%)'
        },
        danger: {
          DEFAULT: 'hsl(0 84% 60%)',
          foreground: 'hsl(210 40% 98%)'
        }
      },
      borderRadius: {
        lg: 'var(--radius)',
        md: 'calc(var(--radius) - 2px)',
        sm: 'calc(var(--radius) - 4px)'
      },
      fontFamily: {
        mono: ['JetBrains Mono', 'Fira Code', 'Monaco', 'Consolas', 'monospace']
      }
    }
  },
  plugins: [
    require('@tailwindcss/forms'),
    require('@tailwindcss/typography')
  ]
}
```

🧩 Step 3: Shadcn/Vue Setup

Install Shadcn/Vue

bash

```bash
# Install Shadcn/Vue CLI
npx shadcn-vue@latest init

# Configuration options:
❌ Would you like to use TypeScript? No
✅ Which framework are you using? Vite
✅ Which style would you like to use? Default
✅ Which color would you like to use as base color? Blue
✅ Where is your global CSS file? src/assets/index.css
✅ Would you like to use CSS variables for colors? Yes
✅ Where is your tailwind.config.js located? tailwind.config.js
✅ Configure the import alias for components? src/components
✅ Configure the import alias for utils? src/lib/utils
```

Install Required Components

bash

```bash
# Core UI components for Palletizer
npx shadcn-vue@latest add button
npx shadcn-vue@latest add card
npx shadcn-vue@latest add input
npx shadcn-vue@latest add slider
npx shadcn-vue@latest add badge
npx shadcn-vue@latest add textarea
npx shadcn-vue@latest add tabs
npx shadcn-vue@latest add switch
npx shadcn-vue@latest add alert
npx shadcn-vue@latest add progress
npx shadcn-vue@latest add separator
npx shadcn-vue@latest add tooltip
npx shadcn-vue@latest add dialog
npx shadcn-vue@latest add select
```

⚙️ Step 4: Vite Configuration

Configure Vite for ESP32

vite.config.js:

javascript

```javascript
import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
import path from 'path'

export default defineConfig({
  plugins: [vue()],
  resolve: {
    alias: {
      '@': path.resolve(__dirname, './src')
    }
  },
  build: {
    // ESP32 optimizations
    target: 'es2015',
    minify: 'terser',
    cssMinify: true,
    rollupOptions: {
      output: {
        // Single bundle for ESP32
        inlineDynamicImports: true,
        manualChunks: undefined,
        // Remove hash from filenames for predictable names
        entryFileNames: 'assets/[name].js',
        chunkFileNames: 'assets/[name].js',
        assetFileNames: 'assets/[name].[ext]'
      }
    },
    // Size optimization for embedded systems
    chunkSizeWarningLimit: 1000,
    assetsInlineLimit: 4096,
    // Remove console.log in production
    terserOptions: {
      compress: {
        drop_console: true,
        drop_debugger: true
      }
    }
  },
  css: {
    devSourcemap: false
  },
  server: {
    host: '0.0.0.0', // Allow external connections for testing
    port: 3000
  }
})
```

📦 Step 5: Package.json Configuration

Update Scripts

package.json scripts section:

json

```json
{
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "build:esp32": "vite build && node build-esp32.js",
    "lint": "eslint . --ext .vue,.js,.jsx,.cjs,.mjs --fix --ignore-path .gitignc
    "format": "prettier --write src/",
    "deploy": "npm run build:esp32 && echo 'Deploy dist/index.html to ESP32'"
  }
}
```

🔧 Step 6: ESP32 Build Script

Create ESP32 Deployment Script

build-esp32.js:

```
javascript
```

```javascript
const fs = require('fs')
const path = require('path')

console.log('🚀 Building for ESP32 deployment...')

const distPath = path.join(__dirname, 'dist')
const htmlPath = path.join(distPath, 'index.html')

if (!fs.existsSync(htmlPath)) {
  console.error('❌ Build files not found. Run "npm run build" first.')
  process.exit(1)
}

// Read the built HTML file
let html = fs.readFileSync(htmlPath, 'utf8')

// Get all CSS files
const assetsPath = path.join(distPath, 'assets')
if (fs.existsSync(assetsPath)) {
  const cssFiles = fs.readdirSync(assetsPath).filter(file => file.endsWith('.css

  // Inline CSS files
  cssFiles.forEach(cssFile => {
    const cssContent = fs.readFileSync(path.join(assetsPath, cssFile), 'utf8')
    const linkRegex = new RegExp(`<link[^>]*href="[^"]*${cssFile}"[^>]*>`, 'g')
    html = html.replace(linkRegex, `<style>${cssContent}</style>`)
  })

  // Get all JS files
  const jsFiles = fs.readdirSync(assetsPath).filter(file => file.endsWith('.js')

  // Inline JS files
  jsFiles.forEach(jsFile => {
    const jsContent = fs.readFileSync(path.join(assetsPath, jsFile), 'utf8')
    const scriptRegex = new RegExp(`<script[^>]*src="[^"]*${jsFile}"[^>]*></scri
    html = html.replace(scriptRegex, `<script>${jsContent}</script>`)
  })
}

// Write the inlined HTML
fs.writeFileSync(htmlPath, html)

// Clean up separate asset files (optional)
if (fs.existsSync(assetsPath)) {
  fs.rmSync(assetsPath, { recursive: true, force: true })
}

// Get file size
const stats = fs.statSync(htmlPath)
const fileSizeInKb = (stats.size / 1024).toFixed(2)

console.log(`✅ ESP32 build complete!`)
console.log(`📄 Output: dist/index.html`)
console.log(`📏 Size: ${fileSizeInKb} KB`)
console.log(`🚀 Ready to upload to ESP32`)
```

```
// Validate size for ESP32
if (stats.size > 1024 * 1024) { // 1MB limit
  console.warn(`⚠️ Warning: File size (${fileSizeInKb} KB) may be too large fo
}
```

🎨 Step 7: Global Styles Setup

Create Global CSS

src/assets/index.css:

css

```css
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
  :root {
    /* Light theme */
    --background: 0 0% 100%;
    --foreground: 222.2 84% 4.9%;
    --card: 0 0% 100%;
    --card-foreground: 222.2 84% 4.9%;
    --popover: 0 0% 100%;
    --popover-foreground: 222.2 84% 4.9%;
    --primary: 221.2 83.2% 53.3%;
    --primary-foreground: 210 40% 98%;
    --secondary: 210 40% 96%;
    --secondary-foreground: 222.2 84% 4.9%;
    --muted: 210 40% 96%;
    --muted-foreground: 215.4 16.3% 46.9%;
    --accent: 210 40% 96%;
    --accent-foreground: 222.2 84% 4.9%;
    --destructive: 0 84.2% 60.2%;
    --destructive-foreground: 210 40% 98%;
    --border: 214.3 31.8% 91.4%;
    --input: 214.3 31.8% 91.4%;
    --ring: 221.2 83.2% 53.3%;
    --radius: 0.75rem;
  }

  .dark {
    /* Dark theme - Industrial look */
    --background: 224 71% 4%;
    --foreground: 213 31% 91%;
    --card: 224 71% 4%;
    --card-foreground: 213 31% 91%;
    --popover: 224 71% 4%;
    --popover-foreground: 213 31% 91%;
    --primary: 210 40% 98%;
    --primary-foreground: 222.2 84% 4.9%;
    --secondary: 215 28% 17%;
    --secondary-foreground: 210 40% 98%;
    --muted: 215 28% 17%;
    --muted-foreground: 217.9 10.6% 64.9%;
    --accent: 215 28% 17%;
    --accent-foreground: 210 40% 98%;
    --destructive: 0 63% 31%;
    --destructive-foreground: 210 40% 98%;
    --border: 215 28% 17%;
    --input: 215 28% 17%;
    --ring: 216 34% 17%;
  }

  * {
    @apply border-border;
  }
  body {
    @apply bg-background text-foreground;
    font-feature-settings: "rlig" 1, "calt" 1;
  }
}

/* Industrial Control Panel Styling */
@layer components {
  .control-panel {
    @apply bg-card border border-border rounded-lg shadow-lg;
  }

  .status-indicator {
    @apply inline-flex items-center px-2.5 py-0.5 rounded-full text-xs font-medi
  }

  .status-indicator.running {
    @apply bg-green-100 text-green-800 dark:bg-green-900 dark:text-green-300;
  }

  .status-indicator.idle {
    @apply bg-gray-100 text-gray-800 dark:bg-gray-800 dark:text-gray-300;
  }

  .status-indicator.error {
    @apply bg-red-100 text-red-800 dark:bg-red-900 dark:text-red-300;
  }

  .command-input {
    @apply font-mono text-sm bg-muted border border-input rounded-md px-3 py-2;
  }
}

/* Animation utilities */
@layer utilities {
  .animate-pulse-soft {
    animation: pulse-soft 2s cubic-bezier(0.4, 0, 0.6, 1) infinite;
  }

  @keyframes pulse-soft {
    0%, 100% {
      opacity: 1;
    }
    50% {
      opacity: .8;
    }
  }
}

/* Custom scrollbar for dark theme */
.dark {
  scrollbar-width: thin;
  scrollbar-color: hsl(var(--border)) transparent;
}
```

```
.dark ::-webkit-scrollbar {
  width: 6px;
}

.dark ::-webkit-scrollbar-track {
  background: transparent;
}

.dark ::-webkit-scrollbar-thumb {
  background-color: hsl(var(--border));
  border-radius: 3px;
}
```

🎛 Step 8: Basic App Structure

Root App Component

src/App.vue:

vue

```vue
<template>
  <div class="min-h-screen bg-background">
    <AppLayout>
      <!-- Main content will go here -->
      <div class="p-6">
        <h1 class="text-2xl font-bold mb-6">ESP32 Palletizer Control</h1>
        <p class="text-muted-foreground">Vue + Shadcn/Vue setup complete! 🚀</p
      </div>
    </AppLayout>
  </div>
</template>

<script setup>
import AppLayout from '@/components/layout/AppLayout.vue'
</script>
```

Basic Layout Component

src/components/layout/AppLayout.vue:

vue

```
<template>
  <div class="flex min-h-screen">
    <!-- Sidebar placeholder -->
    <aside class="w-64 bg-card border-r border-border">
      <div class="p-4">
        <h2 class="font-semibold">Navigation</h2>
      </div>
    </aside>

    <!-- Main content -->
    <main class="flex-1">
      <slot />
    </main>
  </div>
</template>

<script setup>
// Layout logic will go here
</script>
```

Entry Point

src/main.js:

javascript

```
import { createApp } from 'vue'
import App from './App.vue'
import './assets/index.css'

const app = createApp(App)

// Global error handler
app.config.errorHandler = (err, vm, info) => {
  console.error('Vue error:', err, info)
}

app.mount('#app')
```

🧪 Step 9: Test the Setup

Run Development Server

bash

```
# Start development server
npm run dev

# Expected output:
# ➜  Local:   http://localhost:3000/
# ➜  Network: http://192.168.1.xxx:3000/
```

Test ESP32 Build

```bash
# Build for ESP32
npm run build:esp32

# Expected output:
# ✅ ESP32 build complete!
# 📄 Output: dist/index.html
# 📏 Size: ~50-100 KB
# 🚀 Ready to upload to ESP32
```

Verify Build Output

```bash
# Check the generated file
ls -la dist/

# Should contain only:
# index.html (single inlined file)
```

🚀 Step 10: Development Workflow

Development Commands

```bash
# Start development with hot reload
npm run dev

# Build for production
npm run build

# Build and prepare for ESP32
npm run build:esp32

# Code formatting
npm run format

# Linting
npm run lint
```

File Upload to ESP32

1. Build the project: `npm run build:esp32`
2. Copy `dist/index.html` content
3. Upload to ESP32 LittleFS as `/index.html`
4. Access via ESP32 IP address

🔍 Next Steps

After completing this setup, you'll be ready to:

1. **Create business logic composables** (`useSystemControl`, `useSpeedControl`, etc.)
2. **Build feature components** (System controls, Speed matrix, Command editor)
3. **Integrate with ESP32 API** (REST endpoints, WebSocket events)
4. **Implement real-time monitoring** (Status updates, error handling)
5. **Add advanced features** (File upload, command history, analytics)

🎯 Expected Results

✅ **Modern Vue 3 + Shadcn/Vue project**
✅ **Zero manual CSS required**
✅ **Industrial-grade UI components**
✅ **ESP32-optimized build pipeline**
✅ **Dark mode ready**
✅ **Mobile responsive**
✅ **Small bundle size (~50-100KB)**
✅ **Professional development workflow**

🆘 Troubleshooting

Common Issues

Build fails:

```bash
# Clear cache and reinstall
rm -rf node_modules package-lock.json
npm install
```

Shadcn components not found:

```bash
# Reinstall shadcn components
npx shadcn-vue@latest add button card input
```

Large bundle size:

- Check `vite.config.js` optimization settings
- Remove unused dependencies
- Enable tree-shaking

ESP32 upload issues:

- Verify file size < 1MB
- Check ESP32 LittleFS capacity
- Ensure single HTML file output

📚 Resources

- Vue 3 Documentation

- Shadcn/Vue Documentation
- Tailwind CSS Documentation
- Vite Documentation
- ESP32 LittleFS Guide

🎉 **Setup Complete! Ready to build the ESP32 Palletizer Control Interface!**