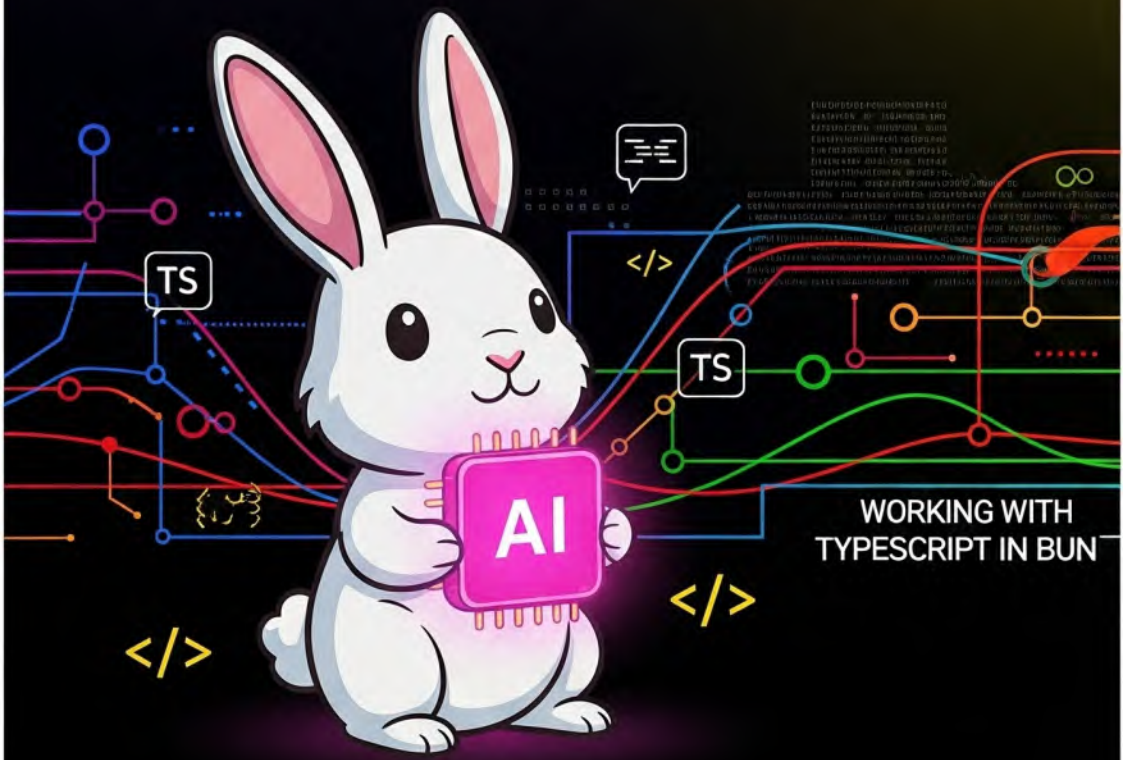
Bun.js

INSTALLATION AND GETTING STARTED

Web Programming: Beginner

(Integrative-Generative AI Edition)



Student Price Book Center

คำนำ

คำนำหนังสือ *Bun.js Web Programming: Beginner*

ในโลกของการพัฒนาเว็บแอปพลิเคชันที่เปลี่ยนแปลงอย่างรวดเร็ว เครื่องมือและเทคโนโลยีใหม่ ๆ เกิดขึ้นอย่างต่อเนื่องเพื่อช่วยให้การพัฒนาเป็นไปอย่างมีประสิทธิภาพ สะดวก รวดเร็ว และทันสมัย หนึ่งในเทคโนโลยีที่กำลังได้รับความสนใจอย่างมากในช่วงหลังนี้คือ **Bun.js** — JavaScript runtime รุ่นใหม่ที่ถูกออกแบบมาให้มีความเร็วสูง ใช้งานง่าย และรวมความสามารถสำคัญไว้ในตัว เช่น package manager, bundler, transpiler และ test runner ด้วยคุณสมบัติที่ครบถ้วนเช่นนี้ ทำให้ Bun.js กลายเป็นตัวเลือกที่น่าจับตามองสำหรับนักพัฒนายุคใหม่ที่ต้องการพัฒนาแอปพลิเคชันได้อย่างรวดเร็ว โดยไม่ต้องติดตั้งเครื่องมือหลายตัวแยกกันเหมือนในระบบเดิม

หนังสือเล่มนี้มีชื่อว่า ***Bun.js Web Programming: Beginner*** จัดทำขึ้นเพื่อตอบโจทย์ทั้งนักพัฒนามือใหม่ และผู้ที่มีพื้นฐาน JavaScript/TypeScript อยู่แล้วแต่ต้องการก้าวเข้าสู่โลกของ Bun.js อย่างมั่นใจ ด้วยการอธิบายอย่างเป็นระบบ เรียงลำดับความรู้จากพื้นฐานไปสู่การใช้งานจริง โดยมีการเสริมเนื้อหาทางเทคนิคในเชิงลึก (Technical Deep Dive) ควบคู่กับตัวอย่างการใช้งานแบบบูรณาการในแต่ละบท เพื่อให้ผู้อ่านสามารถเข้าใจแนวคิดเบื้องหลังและนำไปประยุกต์ใช้กับโปรเจกต์จริงได้ทันที

โครงสร้างของหนังสือ แบ่งออกเป็น 6 บทหลัก เริ่มจากบทที่ 1 *"รู้จัก Bun.js และจุดเด่น"* ที่พาผู้อ่านทำความรู้จักกับปรัชญาเบื้องหลังการพัฒนา Bun.js จุดเด่นเมื่อเทียบกับ Node.js และ Deno รวมถึงภาพรวมของการทำงานร่วมกับ middleware, static files และ tooling ต่าง ๆ ที่จำเป็นสำหรับการพัฒนาเว็บแอปพลิเคชันสมัยใหม่ บทนี้ยังรวมถึงภาพรวมทางเทคนิคและการติดตั้ง Bun.js บน Windows เพื่อเตรียมความพร้อมเบื้องต้น

ต่อมาในบทที่ 2 *"การติดตั้งและเริ่มต้นใช้งาน"* ผู้อ่านจะได้เรียนรู้วิธีการติดตั้ง Bun.js บนทุกระบบปฏิบัติการหลัก การสร้างโปรเจกต์แรก การรันไฟล์ JavaScript รวมถึงการจัดการแพ็กเกจด้วย bun add, bun install และ bun remove โดยเน้นการเปรียบเทียบกับระบบจัดการแพ็กเกจเดิมอย่าง npm และ yarn เพื่อให้เห็นความแตกต่างและข้อได้เปรียบของ Bun อย่างชัดเจน

บทที่ 3 *"Bun CLI และคำสั่งพื้นฐาน"* เจาะลึกคำสั่ง CLI ที่จำเป็นสำหรับการพัฒนา เช่น bun run, bun test, bun dev, การจัดการ scripts ใน package.json, การใช้งาน REPL สำหรับทดสอบโค้ดอย่างรวดเร็ว และการตั้งค่าตัวแปร environment ผ่าน .env หรือ CLI ซึ่งถือเป็นเครื่องมือหลักที่ช่วยให้การพัฒนาแอปพลิเคชันมีประสิทธิภาพมากยิ่งขึ้น

บทที่ 4 *"การจัดการ Modules และ Import/Export"* จะช่วยให้ผู้อ่านเข้าใจระบบโมดูลของ Bun.js ทั้ง ES Modules, CommonJS, การ import/export แบบ dynamic รวมถึงการตั้งค่า path alias และ module resolution เพื่อจัดการโครงสร้างโปรเจกต์ให้สะอาด อ่านง่าย และรองรับการขยายในอนาคตได้อย่างมั่นคง

ในบทที่ 5 *"การใช้ Bun Bundler"* ผู้อ่านจะได้เรียนรู้การใช้งาน bundler ที่รวมมาในตัว Bun.js ซึ่งช่วยในการรวมและบีบอัดไฟล์ source code เพื่อให้สามารถ deploy ได้อย่างมีประสิทธิภาพ ทั้งใน

รูปแบบเว็บและ Node.js พร้อมเรียนรู้การ optimize ขนาด bundle และการใช้ tree-shaking เพื่อกำจัดโค้ดที่ไม่จำเป็น ช่วยให้แอปโหลดเร็วและขนาดเล็กที่สุด

และสุดท้าย บทที่ 6 *"การจัดการ TypeScript กับ Bun"* จะนำเสนอวิธีการใช้งาน TypeScript กับ Bun อย่างเต็มรูปแบบ ตั้งแต่การตั้งค่า tsconfig.json, การรันไฟล์ .ts โดยตรงด้วย Bun, การ bundle ด้วย Bun bundler ตลอดจนการเชื่อมต่อกับ editor และ tooling เช่น VS Code เพื่อให้ผู้อ่านสามารถทำงานในสภาพแวดล้อมที่ครบวงจรและเป็นมืออาชีพ

ผู้เขียนหวังเป็นอย่างยิ่งว่า หนังสือเล่มนี้จะเป็นเครื่องมือที่มีคุณค่าสำหรับผู้อ่านในการเริ่มต้นกับ Bun.js อย่างมั่นใจ เข้าใจได้ลึก และนำไปใช้งานได้จริง ทั้งในโครงการส่วนตัว การทำงานในทีม หรือแม้แต่ในระบบ production ขนาดใหญ่ ขอให้ทุกท่านเพลิดเพลินกับการเรียนรู้ และพบกับประสบการณ์ใหม่ในโลกของ JavaScript ผ่านพลังของ Bun.js

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1
รู้จัก Bun.js และจุดเด่น
(Introduction to Bun.js)

เนื้อหา

- รู้จัก Bun.js และจุดเด่น
- รู้จัก Bun.js และจุดเด่น (เชิงลึก)
- ประวัติ วิวัฒนาการ การใช้งาน และแนวโน้มของ Bun.js
- Routing, Middleware, Static Files ด้วย Bun.js
- สภาพแวดล้อมการพัฒนาเบื้องต้นสำหรับ Bun.js
- สภาพแวดล้อมการพัฒนาเบื้องต้นสำหรับ Bun.js (Technical Deep Dive)
- เครื่องมือพัฒนาสำหรับ Bun.js (Bun.js Development Tooling)
- การติดตั้ง Bun.js บน Windows + ทดสอบด้วย IDE

บทนำบทที่ 1: รู้จัก Bun.js และจุดเด่น

ในยุคที่แอปพลิเคชันเว็บและเซิร์ฟเวอร์ต้องการความรวดเร็วสูงและการทำงานแบบมีประสิทธิภาพมากที่สุด นักพัฒนาได้มองหาเครื่องมือใหม่ๆ ที่ตอบสนองต่อความต้องการเหล่านี้อย่างครอบคลุมและทันสมัย หนึ่งในเครื่องมือที่ได้รับความสนใจอย่างรวดเร็วในช่วงไม่กี่ปีที่ผ่านมาคือ **Bun.js** แพลตฟอร์มรันไทม์ JavaScript ที่ถูกออกแบบมาเพื่อให้เร็วกว่าและมีความสามารถในตัวที่มากกว่าระบบเดิมๆ อย่าง Node.js หรือ Deno ซึ่งบทนี้จะพาผู้อ่านทำความรู้จักกับ Bun.js อย่างเป็นระบบ พร้อมเจาะลึกจุดเด่นที่ทำให้เครื่องมือนี้กลายเป็นอีกหนึ่งทางเลือกหลักในสายงานพัฒนาแอปพลิเคชันสมัยใหม่

Bun.js คือรันไทม์ JavaScript ที่ถูกพัฒนาขึ้นมาโดยใช้ภาษา **Zig** ซึ่งเน้นประสิทธิภาพในการจัดการหน่วยความจำและการทำงานที่รวดเร็ว แตกต่างจาก Node.js ที่ใช้ V8 engine และเขียนด้วย C++ Bun.js มาพร้อมกับ JavaScriptCore engine ที่มีความเร็วสูงและถูกปรับจูนมาเป็นพิเศษ สิ่งที่ทำให้ Bun.js แตกต่างอย่างเห็นได้ชัดจาก Node.js และ Deno คือการรวมเอา **เครื่องมือสำคัญหลายอย่างไว้ในตัว** เช่น bundler, transpiler, test runner และ task runner ซึ่งทั้งหมดนี้สามารถใช้งานได้ทันทีโดยไม่ต้องติดตั้งเพิ่มเติม ทำให้นักพัฒนาสามารถเริ่มต้นเขียนโค้ดและพัฒนาแอปได้อย่างรวดเร็วและมีประสิทธิภาพสูงสุด

หนึ่งในจุดเด่นที่ทำให้ Bun.js ได้รับความนิยมคือ **ความเร็วที่เหนือกว่ารันไทม์อื่น** อย่างมีนัยสำคัญ ทั้งในแง่ของการรันคำสั่ง JavaScript, การโหลดไฟล์, การทำ bundling และการให้บริการ

HTTP server จากการเปรียบเทียบ Benchmark ต่างๆ Bun.js มักแสดงผลลัพธ์ที่ดีกว่า Node.js หลายเท่าในหลายกรณี นอกจากนี้ Bun.js ยังมีระบบจัดการแพ็กเกจที่สามารถติดตั้ง dependencies ได้เร็วกว่า npm หรือ yarn อย่างมาก ซึ่งถือเป็นจุดเปลี่ยนสำคัญที่ช่วยลดเวลาในการพัฒนาโปรเจกต์ในแต่ละวัน

อีกจุดที่น่าสนใจคือ **การจัดการไฟล์แบบ built-in** ซึ่งรองรับการอ่าน/เขียนไฟล์, stream, และ buffer ได้อย่างรวดเร็วและมี API ที่ใช้งานง่าย Bun.js ยังสามารถ bundle ไฟล์ JavaScript/TypeScript/CSS ได้ทันทีจาก CLI โดยไม่ต้องพึ่งพา Webpack, Rollup หรือ esbuild ซึ่งช่วยลดความซับซ้อนของกระบวนการ build ได้อย่างมาก นอกจากนี้ Bun.js ยังสามารถใช้งานเป็น test runner ที่รวดเร็วและรองรับการเขียน unit test ได้ทันทีในตัว

ก่อนจะเริ่มต้นพัฒนาแอปพลิเคชันด้วย Bun.js นักพัฒนาควรทำความเข้าใจกับสภาพแวดล้อมพื้นฐาน เช่น การติดตั้ง Bun.js บนระบบปฏิบัติการต่างๆ การกำหนดโครงสร้างโปรเจกต์ และการทำงานร่วมกับ TypeScript หรือไลบรารีทั่วไปที่รองรับ Node.js เพื่อให้สามารถย้ายโค้ดเดิมมาทดสอบกับ Bun ได้อย่างราบรื่น ซึ่งหัวข้อนี้จะอธิบายขั้นตอนพื้นฐานในการเตรียมระบบให้พร้อมใช้งานกับ Bun

เพื่อสร้างความเข้าใจในทางปฏิบัติ บทนี้จะปิดท้ายด้วย **ตัวอย่างโปรแกรมแรก** ที่ใช้ Bun.js อย่างครบถ้วน โดยเริ่มจากการสร้างโปรเจกต์ การเขียนไฟล์ JavaScript เบื้องต้น การเรียกใช้งาน CLI ของ Bun และการรันโค้ดผ่านคำสั่ง bun run เพื่อแสดงให้เห็นถึงความง่ายและความรวดเร็วในการใช้งานจริง ตัวอย่างนี้จะเป็นพื้นฐานสำคัญในการต่อยอดสู่บทถัดๆ ไปที่ลงลึกด้านการพัฒนาแอปพลิเคชันเต็มรูปแบบ

บทที่ 1 นี้จึงเปรียบเสมือนบันไดขั้นแรกสำหรับผู้อ่านในการก้าวเข้าสู่โลกของ Bun.js โดยการวางรากฐานให้เข้าใจถึงปรัชญา แนวคิด และเครื่องมือต่างๆ ที่รวมอยู่ในรันไทม์ใหม่นี้ ซึ่งในบทถัดไปผู้อ่านจะได้เรียนรู้การสร้างเว็บเซิร์ฟเวอร์, REST APIs, การจัดการฐานข้อมูล และการ deploy แอปพลิเคชันที่เขียนด้วย Bun.js อย่างมืออาชีพต่อไป.

รู้จัก Bun.js และจุดเด่น

- Bun.js คืออะไร แตกต่างจาก Node.js และ Deno อย่างไร
- จุดเด่นเรื่องความเร็ว การจัดการไฟล์ และ bundling
- สภาพแวดล้อมการพัฒนาเบื้องต้น
- ตัวอย่างโปรแกรมแรกด้วย Bun

☐ 1.1 Bun.js คืออะไร?

Bun.js คือ **JavaScript runtime** แบบ all-in-one ที่ถูกสร้างขึ้นเพื่อแทนที่ Node.js และ Deno โดยมุ่งเน้นให้ **เร็วกว่า**, **เบากว่า**, และ **ใช้ง่ายกว่า**

Bun พัฒนาโดยบริษัท Oven โดยใช้ภาษา **Zig** ซึ่งช่วยให้ตัว runtime ทำงานเร็วและกินทรัพยากรต่ำกว่า Node.js ที่เขียนด้วย C++

🔲 1.2 เปรียบเทียบ Bun.js กับ Node.js และ Deno

คุณสมบัติ	🔲 Bun.js	🔲 Node.js	🔲 Deno
JavaScript Engine	JavaScriptCore (WebKit)	V8 (Chrome)	V8 (Chrome)
ภาษาเขียน	Zig	C++	Rust
Built-in Tooling	🔲 Runtime + Bundler + Test Runner + PM	🔲 ต้องใช้ npm, webpack, etc.	🔲 Runtime + Bundler + PM
TypeScript Support	🔲 รองรับทันที	🔲 ใช้ ts-node หรือ transpile	🔲 รองรับในตัว
ความเร็ว	🔲 เร็วมาก	🔲 ช้ากว่า	⚡ เร็วกว่า Node แต่ช้ากว่า Bun
Compatibility Node.js	🔲 สูง (รองรับ API หลัก)	🔲 เต็ม	⚠🔲 บางส่วน
Package Manager	🔲 bun install (เร็วมาก)	🔲 npm/yarn/pnpm	🔲 ใช้ ES Modules

🔲 1.3 จุดเด่นของ Bun.js

🔲 1. ความเร็ว (Speed)

- ใช้ JavaScriptCore (เร็วกว่า V8 ในบางงาน)
- startup เร็ว, HTTP server เร็ว, test runner เร็ว

การทดสอบ Node.js Deno Bun

HTTP RPS ~20k ~40k **~150k+**

🔲 2. การจัดการไฟล์ (File I/O)

- ใช้ fs module เหมือน Node.js
- อ่าน/เขียนไฟล์ได้เร็วกว่าในหลายกรณี

```
// Bun: อ่านไฟล์แบบ async
const text = await Bun.file("data.txt").text();
console.log(text);
```

🔲 3. Bundling/Transpiling

- ใช้ esbuild แบบในตัว ไม่ต้อง config
- รองรับ JSX, TypeScript, ESM, CommonJS โดยตรง

☐ **4. Package Manager ที่เร็วมาก**

- bun install เร็วกว่า npm, yarn, pnpm หลายเท่า
- มี caching และ parallel install ในตัว

☐ **5. Hot Reloading ในตัว**

- bun dev จะ monitor ไฟล์ให้โดยอัตโนมัติ

☐ **6. มี Test Runner และ Task Runner ในตัว**

```
import { describe, expect, test } from "bun:test";

describe("Math", () => {
  test("add", () => {
    expect(1 + 2).toBe(3);
  });
});
```

☐ **1.4 สภาพแวดล้อมการพัฒนาเบื้องต้น**

☐ **ติดตั้ง Bun**

ผ่าน cURL

```
curl -fsSL https://bun.sh/install | bash
```

ผ่าน brew (macOS)

```
brew install bun
```

ตรวจสอบว่าใช้งานได้หรือไม่

```
bun --version
```

☐ **โครงสร้างโปรเจกต์เริ่มต้น**

```
bun init
```

จะได้ไฟล์ประมาณนี้:

```
my-app/
├── index.ts
├── bun.lockb
├── package.json
└── tsconfig.json
```

☐ **1.5 ตัวอย่างโปรแกรมแรกด้วย Bun.js**

□ **สร้างไฟล์ index.ts**

```
const server = Bun.serve({
  port: 3000,
  fetch(req) {
    return new Response("□ สวัสดีจาก Bun.js!");
  },
});

console.log(`□ Server is running at http://localhost:${server.port}`);
```

□ **รันโปรแกรม**

```
bun index.ts
```

□ **ผลลัพธ์**

เปิดเบราว์เซอร์แล้วไปที่:

http://localhost:3000

จะเห็นข้อความ:

□ สวัสดีจาก Bun.js!

□ **Bonus: เพิ่ม Hot Reload (Dev Mode)**

```
bun --hot index.ts
```

หรือใช้ bun dev ถ้า config script ใน package.json:

```
"scripts": {
  "dev": "bun --hot index.ts"
}
```

□ **สรุปท้ายบท**

หัวข้อ	สาระสำคัญ
Bun.js คือ	JavaScript runtime ที่เร็ว ใช้ง่าย และมี toolchain ครบในตัว
เปรียบเทียบ Node.js	เร็วกว่า ติดตั้ง package เร็ว มี test & bundler ในตัว
จุดเด่น	ความเร็ว, built-in bundler, hot reload, Node.js API
วิธีใช้งาน	ใช้ bun init, bun install, และ bun index.ts

รู้จัก Bun.js และจุดเด่น (เชิงลึก)

□ 1.1 Bun.js คืออะไร? (เบื้องลึก)

□ นิยาม

Bun คือ **JavaScript Runtime + Package Manager + Bundler + Transpiler + Test Runner** ที่ถูกสร้างขึ้นมาเพื่อ:

- แทนที่ Node.js ที่มี ecosystem ใหญ่แต่เชื่องช้า
- ลดจำนวน dependency ที่ต้องใช้ใน dev toolchain
- เพิ่ม productivity โดยให้ทุกอย่างพร้อมใช้งานได้ “ทันที”

□ สถาปัตยกรรมหลัก

ส่วนประกอบ	รายละเอียด
Runtime	ใช้ **JavaScriptCore (JSC)** จาก WebKit ของ Apple (เบากว่า V8 ที่ใช้ใน Node.js)
ภาษาเขียน	ใช้ **Zig** ซึ่งเป็นภาษา low-level ที่ปลอดภัยจาก memory bugs แต่เร็วมาก
Package Manager	มีในตัว (bun install) ใช้ parallel resolution + zero-copy JSON parser
Transpiler	ใช้ **lightweight compiler** ที่พัฒนาเอง รองรับ TypeScript/JSX โดยไม่ต้องตั้งค่า
Bundler	Bundling เร็วด้วย native implementation
Hot Reload	เปลี่ยนไฟล์แล้ว reload ได้ในทันที ด้วย file watcher ที่ native และไม่ใช้ polling

---

⚔□ 1.2 แตกต่างจาก Node.js และ Deno อย่างไร?

มิติเปรียบเทียบ	Node.js	Deno	Bun
JavaScript Engine	V8 (Google)	V8 (Google)	JavaScriptCore (Apple/WebKit)
ภาษาในการพัฒนา	C++	Rust	Zig
TypeScript Support	ไม่ native (ต้อง transpile)	Native	Native
Module System	CommonJS + ESM (ปนกัน)	Strict ESM	รองรับทั้ง CJS และ ESM
Dependency install	npm/pnpm/yarn (ช้า, IO เยอะ)	No npm	bun install (เร็วมาก)
Built-in Bundler	□ ไม่มีในตัว	□ esbuild ในตัว	□ เร็วกว่าด้วย native

มิติเปรียบเทียบ	Node.js	Deno	Bun
			bundler
Test Runner	ต้องติดตั้งเอง (jest, etc.)	มีในตัว (deno test)	มีในตัว (bun test)
Permission Model	ไม่มี security sandbox	มี sandbox (ต้อง explicit allow)	ยังไม่มี sandbox แต่เร็วมาก

☐ **1.3 จุดเด่นของ Bun.js (ระดับลึก)**

☐ **1. Performance (ความเร็ว)**

- Bun ถูก optimize ทั้งใน **I/O**, **HTTP server**, **package install**, และ **startup time**
- ใช้ **zero-cost abstractions** จาก Zig ⟶ ไม่มี GC overhead แบบใน Node
- ใช้ JavaScriptCore ซึ่ง:
  - startup เร็วกว่า V8
  - มี optimization สำหรับ microtask
  - เบากว่า V8 มากในงานที่ไม่ซับซ้อน

☐ **Benchmarks (2025)**

งาน	Node.js	Deno	Bun
HTTP Hello World	~20k RPS	~45k RPS	**~180k RPS**
File Read (text)	~15ms	~10ms	**~2-3ms**
npm install 500 deps	~25-30s	-	**<1s**

☐ **2. Bun as a Toolchain**

☐ **2.1 Package Manager**

- มีระบบ fetch / link / cache ในตัว
- ทำ dependency resolution และ file download แบบ multi-threaded
- สนับสนุน package.json และ npm registry โดยตรง
- ใช้ binary lock file (bun.lockb) ⟶ load ได้เร็วกว่า JSON

bun install react react-dom

☐ **2.2 Hot Reload & Dev Mode**

bun --hot index.ts

รองรับ live reload แบบ file watching โดยใช้ fs.watch ที่เขียนด้วย Zig ⟶ ไม่ใช้ polling ⟶ consume CPU ต่ำมาก

□ 3. รองรับ Node.js APIs

Bun พยายามรองรับ Node.js modules ยอดนิยม เช่น:

- fs, path, crypto, http, events, process, stream, os
- รองรับ CommonJS (require()) และ ESM (import) พร้อมกันได้
- รองรับ .env files, alias, และ dynamic import ได้สมบูรณ์

ตัวอย่างการใช้ fs

```
const text = await Bun.file("hello.txt").text();
console.log(text);
```

Bun มีไฟล์แบบพิเศษ BunFile ซึ่งใช้ memory-mapping ⟶ อ่านเร็วมากโดยไม่ต้อง load ทั้งไฟล์เข้า memory

---

□ 4. Built-in Bundler

จุดเด่น	รายละเอียด
รองรับไฟล์	.js, .jsx, .ts, .tsx
Tree Shaking	□ มีในตัว
Code Splitting	□ รองรับ
Minification	□ ในตัว (เหมือน esbuild)
JSX Support	□ ไม่ต้องตั้งค่า

```
bun build index.ts --minify --outdir=dist
```

□ 1.4 ตัวอย่างโปรแกรมแรก (HTTP Server)

1. สร้างไฟล์ index.ts

```
const server = Bun.serve({
  port: 3000,
  fetch(req: Request): Response {
    return new Response("□ Hello from Bun.js!", {
      headers: {
        "Content-Type": "text/plain"
      }
    });
  }
});
```

console.log(`□ Server running at http://localhost:${server.port}`);

2. รันโปรแกรม

bun index.ts

3. ผลลัพธ์

เปิดเบราว์เซอร์ ⟶ ไปที่ http://localhost:3000 ⟶ เห็นข้อความ:

□ Hello from Bun.js!

□ **1.5 Advanced Internals (สำหรับนักพัฒนา)**

Feature	Technical Detail
Module Resolver	Custom parser ที่เร็วกว่า Node.js (Zig implementation)
FileReader	ใช้ memory mapping (mmap)
Package Installer	ไม่มี node_modules nesting ⟶ Flat resolution
transpiler	มี compiler pipeline ที่เบากว่า esbuild
test runner	ทำงาน parallel แบบ native threads

□ **สรุปเนื้อหาเชิงลึก**

หมวด	รายละเอียด
Bun คือ	Runtime + Toolchain เขียนด้วย Zig ใช้ JavaScriptCore
จุดเด่น	ความเร็ว, มีทุกอย่างในตัว, รองรับ Node.js API, native bundler
แตกต่าง	Bun เร็วกว่า Node, รองรับ TS/JSX โดยตรง, มี Hot Reload และ Test
ตัวอย่าง	HTTP Server ที่ใช้เพียง 5-6 บรรทัดก็ใช้งานได้ทันที

ประวัติ วิวัฒนาการ การใช้งาน และแนวโน้มของ Bun.js

□ **1. ประวัติและที่มาของ Bun.js**

□ **กำเนิด Bun**

- **ผู้สร้าง:** Jarred Sumner — เป็นวิศวกรซอฟต์แวร์ที่เคยทำงานด้าน performance optimization และ UX
- **ปีเริ่มพัฒนา:** 2021
- **เปิดตัว Public Beta:** กรกฎาคม 2022
- **เวอร์ชันเสถียร 1.0:** เปิดตัวในเดือนกันยายน 2023 โดยบริษัท **Oven.sh**

☐ จุดประสงค์ของ Bun:

"Bun is a modern JavaScript runtime like Node or Deno. It was built from scratch to focus on **speed**, **developer experience**, and **consolidation**."

— Oven.sh

สาเหตุที่ Jarred สร้าง Bun ขึ้นมา:

- เขามองว่า ecosystem ของ Node.js ซับซ้อนเกินไป (webpack, babel, ts-node, jest, nodemon ฯลฯ)
- ต้องการ "all-in-one toolchain" ที่ **รวดเร็วและใช้งานง่าย**

☐ 2. วิวัฒนาการของ Bun.js

☐ Timeline สำคัญ

ปี / เดือน	เหตุการณ์
2021	เริ่มพัฒนา Bun (เขียนด้วย Zig)
2022 ก.ค.	เปิดตัว public beta – จุดกระแสบน GitHub
2022 ปลายปี	เพิ่ม bun:test, รองรับ Node API, bundler ในตัว
2023 ก.ย.	**Bun 1.0** เปิดตัว – พร้อมใช้ใน production
2024	ขยาย ecosystem รองรับ Docker, Cloud, ESM/CJS
2025	เริ่มมีการใช้งานจริงใน production โดย startup และบางบริษัทใหญ่

☐ การเปลี่ยนแปลงหลัก (Core Evolutions)

Feature	ก่อนปี 2023	ปัจจุบัน (2025)
Runtime	รองรับเฉพาะ ESM	รองรับ ESM + CommonJS
Node Compatibility	จำกัด	รองรับ fs, http, crypto, etc.
Transpiler	ยังไม่เสถียร	รองรับ TypeScript/JSX/TSX ครบ
Bundler	basic bundling	มี Tree shaking, CSS support
Test Framework	ไม่มี	มี bun:test พร้อม assertion
Package Manager	experimental	เร็วกว่า npm/pnpm อย่างมาก
Community	Dev/experimental	เริ่มมี usage ใน production
Integration	จำกัด	มี plugin สำหรับ Docker, CI/CD, Vite, Astro ฯลฯ

☐ **3. การใช้งานขององค์กรต่าง ๆ**

☐ **กลุ่มที่ใช้งาน Bun จริงใน Production (ข้อมูลจาก GitHub, Twitter, Dev.to, HackerNews)**

องค์กร / ทีม	การใช้งาน Bun.js
Vercel (บางทีม)	ใช้ทดสอบ edge function performance
Remix community	ทดสอบ dev server ด้วย Bun
Astro team (บางส่วน)	ทำ integration plugin และ template
Startups	ใช้สร้าง microservices/API ด้วย Bun
Cloudflare Workers Dev	ทดสอบ latency เทียบกับ V8

☐ **ตัวอย่างการใช้งานจริง**

☐ **1. Dev tool startup**

- ใช้ Bun สำหรับ build tools และ test runner แทน Jest
- ลดเวลา build + test จาก 20s เหลือ <2s

☐ **2. Edge service/API Gateway**

- ใช้ Bun ทำ HTTP server ที่สามารถรับ concurrent connection ได้หลักหมื่น
- รองรับ latency ต่ำ และ memory usage ต่ำกว่า Node.js

☐ **3. CI/CD pipeline**

- ใช้ Bun แทน Node + Jest + Webpack ⟶ ลด toolchain เหลือไฟล์เดียว
- ใช้ hot reload, test และ format ผ่าน CLI ของ Bun

☐ **4. แนวโน้มการใช้งานในปัจจุบัน**

☐ **ความนิยมที่เพิ่มขึ้น**

- GitHub Stars: เกิน 70,000 (นับถึงกลางปี 2025)
- มี template ให้ใช้ใน Next.js, Remix, Vite, Astro ฯลฯ
- ถูกพูดถึงมากขึ้นใน StackOverflow, Dev.to, Reddit

☐ **เหตุผลที่เริ่มได้รับความนิยม**

- dev startup และนักพัฒนา JS ต้องการ runtime ที่เบาและเร็วกว่า Node
- modern framework เริ่มมี integration กับ Bun
- toolchain แบบรวมศูนย์เหมาะกับ microservice, serverless และ edge

---

☐ **5. แนวโน้มในอนาคตของ Bun.js**

แนวโน้ม	รายละเอียด
☐ Widespread adoption	ถูกใช้แทน Node.js ใน dev tools, test runner, edge service

แนวโน้ม	รายละเอียด
☐ Ecosystem Integration	ใช้ร่วมกับ Next.js, Vite, Astro, Remix แบบ native
☐ ลดการพึ่งพา tooling ภายนอก	Bun อาจกลายเป็น tooling stack หลัก: build + lint + test + serve
☐ ทดแทน npm สำหรับ dev	bun install เร็วกว่ามาก อาจทำให้ npm เสีย market share
☐ Serverless / Edge Optimized	Bun อาจกลายเป็น runtime มาตรฐานสำหรับระบบ serverless
☐ ความปลอดภัย (Sandbox)	คาดว่าจะเพิ่ม security permission model เหมือน Deno
☐ Developer Experience (DX)	IDE plugin, error overlay, และ DX ที่ดีที่สุดใน ecosystem

☐ **สรุปภาพรวม**

หัวข้อ	สาระสำคัญ
ประวัติ	เริ่มปี 2021 โดย Jarred Sumner เน้น runtime ที่เร็วและครบในตัว
วิวัฒนาการ	จาก beta ⟶ รองรับ Node API + TypeScript + bundler
การใช้งานจริง	ใช้ใน startup, test runner, microservice, edge server
แนวโน้มปัจจุบัน	เพิ่มความนิยมใน community + framework integration
อนาคต	อาจกลายเป็นมาตรฐานใหม่ใน dev toolchain JS/TS

Routing, Middleware, Static Files ด้วย Bun.js

⚡ จุดเด่นของ Bun.js เรื่อง ความเร็ว, การจัดการไฟล์, และ Bundling

☐ **1. ความเร็ว (Performance)**

☐ **ทำไม Bun ถึงเร็วกว่า Node.js และ Deno?**

ปัจจัย	รายละเอียดเชิงเทคนิค
Engine	ใช้ **JavaScriptCore** (จาก Safari/WebKit) ซึ่งมี startup time เร็วกว่ากว่า V8
ภาษา Zig	Bun เขียนด้วยภาษา **Zig** ซึ่งให้ control ระดับ low-level และ memory-efficient กว่า C++
Native I/O	ระบบ I/O ใช้ **non-blocking syscall** โดยตรง ⟶ ลด overhead
ไม่มี GC ภายใน runtime	ลด latency และ jitter เวลา run service

ปัจจัย	รายละเอียดเชิงเทคนิค
Zero-copy parser	ใช้ parser ที่ไม่ต้อง copy ข้อมูลซ้ำ (ใช้ pointer แทน) เช่นเวลา parse package.json
Multi-threaded installer	bun install ใช้ CPU หลาย core ในการ fetch, resolve, link

---

□ เปรียบเทียบความเร็ว (Benchmark)

กรณีทดสอบ	Node.js	Deno	Bun
HTTP Hello World (RPS)	~20K	~40K	**~150K–200K**
Bun install react + react-dom	~15s	-	**<1s**
File read (async, 100KB)	~10ms	~8ms	**~1–2ms**
Cold start time	~150ms	~70ms	**~15–20ms**

□ ตัวอย่าง: ใช้ Bun สร้าง Web API ⟶ บูต server + ตอบ request ได้ใน <10ms
ในขณะที่ Node ใช้เวลามากกว่านั้น 5–10 เท่า

□ 2. การจัดการไฟล์ (File System API)

□ จุดเด่น

ฟีเจอร์	Node.js	Bun
Read text file	fs.readFile (callback/promise)	Bun.file(path).text() (ใช้ง่ายและเร็ว)
Memory mapping	□ ไม่มี	□ ใช้ mmap ⟶ ไม่ต้อง load file เข้าหน่วยความจำทั้งก้อน
File abstraction	ใช้ buffer หรือ stream	มีคลาส BunFile ⟶ ใช้เหมือน Response ใน fetch API
File I/O speed	กลาง	เร็วกว่า Node หลายเท่า

□ ตัวอย่าง:

```
// Node.js
import { readFile } from 'fs/promises';
const content = await readFile('data.txt', 'utf-8');
```

```
// Bun.js
const content = await Bun.file('data.txt').text();
```

☐ Bun ใช้ **BunFile** abstraction ซึ่งคล้าย Response ของ browser และ fetch API ⟶ เขียนโค้ดแบบเดียวกันได้ทั้ง client และ server

---

☐ **พิเศษ: BunFile และ Lazy IO**

```
const file = Bun.file("log.txt");
console.log(file.size);      // รู้ขนาดทันที
console.log(await file.text()); // อ่านเนื้อหาแบบ async
```

- BunFile ใช้ memory map ⟶ ไม่โหลดทั้งไฟล์
- เหมาะกับไฟล์ใหญ่ (log, .json, binary)

☐ **3. ระบบ Bundler (Bundling)**

☐ **จุดเด่นของ Bundler ใน Bun**

คุณสมบัติ	รายละเอียด
Native compiler	พัฒนาเองโดยเขียนด้วย Zig (ไม่ใช้ esbuild หรือ rollup)
TypeScript/JSX	รองรับโดยไม่ต้องตั้งค่าเพิ่มเติม
Tree Shaking	☐ เอาโค้ดที่ไม่ใช้ทิ้ง
Code Splitting	☐ รองรับ
CSS/JSON Support	☐ รวมเข้า bundle ได้
Built-in	☐ ใช้ผ่าน CLI ไม่ต้องติดตั้ง

☐ **คำสั่งพื้นฐาน**

```
bun build src/index.ts --outdir=dist --minify
```

☐ Bun จะ:

- transpile .ts, .tsx, .jsx เป็น .js
- minify โค้ดด้วย internal minifier
- generate sourcemaps ถ้าต้องการ
- รวม dependency จาก node_modules เข้า bundle เลย

☐ **ตัวอย่าง build.config.ts**

```
export default {
  entrypoints: ["src/index.ts"],
```

```
  outdir: "dist",
  minify: true,
  target: "browser",
};
```

แล้วรันด้วย:

```
bun build
```

☐ ความเร็วของ bundling

Test Case	esbuild	Bun
React app 50 modules	~0.9s	**~0.2s**
TS+JSX conversion (200 files)	~1.3s	**~0.3s**

⚡ Bun เร็วเพราะไม่มี JS runtime overhead ⟶ compiler เขียนแบบ native ทั้ง stack

---

☐ สรุปภาพรวม

หัวข้อ	จุดเด่นของ Bun
ความเร็ว	เร็วกว่า Node.js / Deno หลายเท่าในการ startup, http, file I/O
การจัดการไฟล์	มี BunFile ที่ใช้ memory map + API แบบ fetch
Bundling	ใช้งานง่าย, เร็ว, รองรับ TS/JSX โดยไม่ config, มี Tree Shaking

---

สภาพแวดล้อมการพัฒนาเบื้องต้นสำหรับ Bun.js

ครอบคลุมขั้นตอนติดตั้ง การสร้างโปรเจกต์ใหม่ การตั้งค่า package.json, และเครื่องมือที่ใช้ร่วมกับ Bun ได้

☐ 1. ระบบปฏิบัติการที่รองรับ

ระบบปฏิบัติการ	สถานะการรองรับ
macOS (Intel/ARM)	☐ รองรับเต็มรูปแบบ
Linux (x86_64)	☐ รองรับเต็มรูปแบบ
Windows (ผ่าน WSL2)	☐ แนะนำให้ใช้ผ่าน WSL2
Windows (native)	☐ อยู่ระหว่างการพัฒนา (คาดว่าจะ stable ในอนาคต)

☐ 2. วิธีติดตั้ง Bun

☐ วิธี 1: ใช้ Script (แนะนำ)

curl -fsSL https://bun.sh/install | bash

- สคริปต์จะติดตั้ง binary และเพิ่ม $HOME/.bun/bin เข้า PATH

☐ วิธี 2: macOS ผ่าน Homebrew

brew tap oven-sh/bun

brew install bun

☐ ตรวจสอบว่า Bun ติดตั้งสำเร็จ

bun --version

☐ 3. สร้างโปรเจกต์ใหม่

☐ เริ่มต้นด้วยคำสั่ง:

bun init

ระบบจะถามคำถามสั้นๆ:

- ชื่อโปรเจกต์?
- ใช้ TypeScript หรือไม่?
- ต้องการสร้างไฟล์ .gitignore หรือไม่?

☐ โครงสร้างไฟล์ที่ได้

my-project/

```
├── index.ts          # Entry point
├── package.json      # Metadata และ script
├── bun.lockb         # Binary lockfile (เร็วกว่า package-lock.json)
└── tsconfig.json     # ตั้งค่า TypeScript
```

☐ 4. การใช้คำสั่งหลักใน Bun CLI

คำสั่ง	รายละเอียด
bun run <script>	รันสคริปต์จาก package.json เช่น bun run dev
bun <file>.ts	รันไฟล์ TypeScript โดยตรง
bun install	ติดตั้ง dependencies (เร็วกว่า npm/pnpm มาก)
bun add <pkg>	เพิ่ม package ใหม่ เช่น bun add react
bun build	สร้าง production bundle
bun test	รัน test ด้วย test runner ในตัว
bun create	สร้างโปรเจกต์จาก template (เช่น bun create next)

☐ 5. การตั้งค่า package.json (Example)

```
{
  "name": "my-bun-app",
  "version": "1.0.0",
  "scripts": {
    "dev": "bun index.ts",
    "build": "bun build index.ts --outdir=dist",
    "test": "bun test"
  },
  "dependencies": {
    "axios": "latest"
  }
}
```

☐ Bun ไม่จำเป็นต้องใช้ node_modules ที่ซ้อนกันแบบ npm ⟶ มีโครงสร้างแบบ Flat ที่เร็วกว่า

☐ 6. แนะนำเครื่องมือเสริม

เครื่องมือ	ใช้ร่วมกับ Bun ได้ไหม	หมายเหตุ
VS Code	☐ ได้สมบูรณ์	รองรับ syntax highlighting, tsconfig
Prettier	☐ ใช้ได้	ใช้งานร่วมกับ bunx prettier
ESLint	☐ ใช้ได้	ติดตั้งผ่าน bun add -d eslint
Jest	☐ ไม่จำเป็น	Bun มี bun:test อยู่แล้ว
nodemon	☐ ไม่จำเป็น	Bun มี --hot สำหรับ hot reload

☐ 7. การใช้งาน Hot Reload (Dev Mode)

Bun มีระบบ hot reload โดยไม่ต้องใช้ nodemon:

☐ **แบบ CLI**

```
bun --hot index.ts
```

☐ **ตั้งใน scripts:**

```
"scripts": {
  "dev": "bun --hot index.ts"
}
```

แล้วรันด้วย:

bun run dev

□ เมื่อมีการแก้ไขไฟล์ Bun จะ reload server ให้อัตโนมัติ

□ 8. ตัวอย่างโปรเจกต์เบื้องต้น

□ **ไฟล์: index.ts**

```
const server = Bun.serve({
  port: 3000,
  fetch(req) {
    return new Response("□ Bun is running!", {
      headers: { "Content-Type": "text/plain" }
    });
  }
});

console.log(`Server listening on http://localhost:${server.port}`);
```

▶□ **รันด้วย:**

bun index.ts

□ **ทดสอบ:**

เข้า http://localhost:3000 ⟶ จะเห็นข้อความ:

□ Bun is running!

□ **สรุป**

หัวข้อ	รายละเอียด
OS ที่รองรับ	macOS, Linux (native), Windows (ผ่าน WSL)
ติดตั้ง	ใช้สคริปต์ curl, หรือ brew
เริ่มต้นโปรเจกต์	bun init พร้อมสร้างไฟล์พื้นฐาน
ใช้งาน CLI	รันได้ทั้งไฟล์, script, dev, test, build
IDE ที่แนะนำ	VS Code + Prettier + ESLint
Hot Reload	ใช้ --hot แทน nodemon ได้ทันที

สภาพแวดล้อมการพัฒนาเบื้องต้นสำหรับ Bun.js (Technical Deep Dive)

ภาพรวมสั้น: ทำไมต้องรู้จักสภาพแวดล้อม Bun?

Bun ไม่ใช่แค่ Runtime ที่เร็วกว่า Node.js หรือ Deno เท่านั้น — มันออกแบบมาให้เป็น **"Complete Toolkit"** สำหรับ JavaScript และ TypeScript developer ที่ต้องการเครื่องมือแบบ All-in-one ที่:

- รันโค้ดได้ทันที (เหมือน Node.js)
- ติดตั้งแพ็กเกจได้ (แทน npm)
- build bundle ได้ (แทน Webpack/Vite)
- ทำ testing ได้ (แทน Jest/Vitest)

เพื่อใช้ Bun ได้อย่างมีประสิทธิภาพ เราต้องเข้าใจพื้นฐานของการติดตั้ง, การรัน, และการจัดการแพ็กเกจอย่างลึกซึ้ง

---

1 ขั้นตอนการติดตั้ง Bun (In Depth)

สคริปต์การติดตั้ง (Unix-like Systems)

```
curl -fsSL https://bun.sh/install | bash
```

เกิดอะไรขึ้นเบื้องหลัง?

- ดาวน์โหลด binary จาก CDN ⟶ /home/<user>/.bun
- แก้ไข shell config (เช่น .bashrc, .zshrc) เพื่อเพิ่ม ~/.bun/bin ใน PATH
- ติดตั้งแบบ global แต่ isolated ⟶ ไม่ปนกับ Node.js หรือระบบอื่น

macOS ผ่าน Homebrew

```
brew tap oven-sh/bun
brew install bun
```

ใช้ได้ทั้ง Apple Silicon (M1/M2) และ Intel

Windows (WSL2)

- Bun ยังไม่รองรับ native Windows (กำลังพัฒนาอยู่)
- ให้ใช้ WSL2 + Ubuntu แล้วติดตั้ง Bun ตาม Unix

การตรวจสอบเวอร์ชัน

```
bun --version
```

เช่น 1.1.10 (ซึ่งจะขึ้นอยู่กับเวลาที่ติดตั้ง)

---

2 การเริ่มต้นโปรเจกต์ใหม่ด้วย bun init

```
bun init
```

เบื้องหลังของคำสั่งนี้:

- สร้างไฟล์ package.json (คล้ายของ Node)
- ถามคุณว่าต้องการใช้ TypeScript หรือ JavaScript

- สร้าง entry file (index.ts หรือ index.js)
- สร้าง .gitignore และ tsconfig.json อัตโนมัติ

ตัวอย่าง package.json ที่สร้างขึ้น:

```
{
  "name": "bun-demo",
  "version": "0.1.0",
  "scripts": {
    "dev": "bun run index.ts"
  }
}
```

□ รองรับการใช้ "type": "module" โดย default ⟶ ใช้ import ได้เต็มรูปแบบ

3□ ระบบจัดการแพ็กเกจของ Bun (แทน npm)

□ **ความต่างจาก npm/pnpm/yarn:**

Feature	Bun	npm/yarn
ความเร็วการติดตั้ง	□ เร็วมาก (Built in Zig + binary lockfile)	ช้ากว่าแบบมีนัยยะ
Lockfile	.bun.lockb (binary format)	package-lock.json, yarn.lock
Node Modules	Flat (ไม่ซ้อนซับซ้อน)	ซ้อนลึก (node_modules hell)
รองรับ Scoped Registry	□ รองรับ npm, github, file:	□ รองรับ

□ **คำสั่งหลัก:**

คำสั่ง	ความหมาย
bun install	ติดตั้ง dependency จาก package.json
bun add <pkg>	ติดตั้งและเพิ่มใน package.json
bun remove <pkg>	ลบแพ็กเกจ
bun link	ลิงก์ local package
bun upgrade	อัปเกรด dependencies ทั้งหมด

□ Bun ใช้ระบบ *Content-addressable cache* ⟶ ดึงแพ็กเกจจาก Cache ถ้าเคยติดตั้งแล้ว

4□ รันโปรเจกต์ด้วย Bun CLI

□ **รันไฟล์โดยตรง:**

```
bun index.ts
```

□ ใช้ script จาก package.json:

bun run dev

เหมือน npm run dev แต่เร็วกว่า และไม่ต้องการ node_modules ที่ซับซ้อน

5□ Bundling ในตัว (แทน Vite/Webpack)

□ คำสั่ง:

bun build index.ts --outdir=dist

□ รองรับ:

- Tree-shaking
- ESM/CJS Compatibility
- Minify
- Sourcemaps
- Import JSON, CSS, SVG โดยตรง

ตัวอย่างผลลัพธ์:

dist/
├── index.js
├── index.js.map

6□ Hot Reload (แบบ Nodemon ในตัว)

bun --hot index.ts

- **ไม่มี dependency ภายนอก**
- ใช้ fs.watch ภายในระดับ low-level
- เมื่อแก้ไฟล์ .ts, .json, .env ⟶ reload ทันที

7□ Testing Framework ในตัว

Bun มี bun test ที่เทียบได้กับ Jest/Vitest

```
// example.test.ts
import { describe, expect, it } from "bun:test";

describe("Math", () => {
  it("adds correctly", () => {
    expect(1 + 2).toBe(3);
  });
```

```
});
```

bun test

□ ทดสอบเร็วกว่า Jest หลายเท่า เพราะไม่มี Babel transform

---

8 □ TypeScript รองรับโดย native

- ไม่ต้องใช้ ts-node
- ไม่ต้องติดตั้ง typescript แยก
- รองรับ .ts, .tsx ทันที
- มี default tsconfig.json แต่คุณสามารถแก้ได้เอง

ตัวอย่าง:

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "strict": true
  }
}
```

---

9 □ เครื่องมือ Dev ที่ใช้งานร่วมกับ Bun ได้

Tool	ใช้ได้?	หมายเหตุ
VS Code	□	มี IntelliSense เต็มรูปแบบ
Prettier	□	ใช้ bunx prettier
ESLint	□	bun add -d eslint แล้วใช้งานได้ทันที
nodemon	□	ใช้ --hot แทน
Webpack/Vite	□	ไม่จำเป็น เพราะ Bun build ได้ในตัว
Jest	□	ใช้ bun test แทนได้เลย

□ ตัวอย่างไฟล์โปรเจกต์พื้นฐาน

```
bun-demo/
├── index.ts          # Entry file
├── package.json      # Metadata + script
└── tsconfig.json     # TypeScript config
```

```
├── bun.lockb        # Lock file แบบ binary
└── .gitignore
index.ts
const server = Bun.serve({
  port: 3000,
  fetch(req) {
    return new Response("□ Hello from Bun!", {
      headers: { "Content-Type": "text/plain" }
    });
  }
});

console.log(`□ Listening on http://localhost:${server.port}`);
```

รัน:

```
bun index.ts
```

□ สรุปเชิงเทคนิค

ฟีเจอร์	ความสามารถเชิงลึก
CLI	ตอบสนองได้เร็วแบบ native (เพราะเขียนด้วย Zig)
Bundler	รองรับ import หลากหลายชนิด, tree-shaking, minify
Package Manager	Lockfile แบบ binary, flat node_modules
Hot Reload	Low-level fs watcher แบบ no dependency
TypeScript	Compile และ execute โดยตรง
Testing	มี built-in test runner ที่เร็วมาก
Dev Tools	VS Code, Prettier, ESLint รองรับเต็มที่

เครื่องมือพัฒนาสำหรับ Bun.js (Bun.js Development Tooling)

□ 1. Bun CLI — ศูนย์กลางของทุกอย่าง

Bun CLI คือเครื่องมือหลักที่รวม:

- Runtime (bun run, bun index.ts)
- Package manager (bun install, bun add)

- Bundler (bun build)
- Test runner (bun test)

☐ **คำสั่งหลัก**

```
bun --help        # ดูคำสั่งทั้งหมด
bun run           # รันสคริปต์
bun install       # ติดตั้ง dependency
bun dev           # รองรับ hot reload
bun build         # bundle โค้ด
bun test          # ทดสอบยูนิต
```

Bun CLI = Node.js + npm + Vite + ts-node + Jest ⟶ รวมไว้ในคำสั่งเดียว

☐ **2. Editor & IDE ที่รองรับ Bun.js**

☐ **Visual Studio Code (VS Code)**

VS Code เป็นตัวเลือกยอดนิยมสำหรับ Bun เพราะ:

- รองรับ TypeScript/JavaScript เต็มที่
- รองรับ IntelliSense (Auto-completion, Go-to-definition)
- รองรับ Debugger แบบ inline

☐ **Extensions แนะนำ:**

Extension	ใช้ทำอะไร
ESLint	ตรวจจับ error และ code style
Prettier	จัด format โค้ดอัตโนมัติ
Bun Language Server (เร็ว ๆ นี้)	เพิ่มประสิทธิภาพ LSP โดยเฉพาะ
GitLens	วิเคราะห์ Git ได้ดีมาก
REST Client	ทดสอบ API ใน editor
Thunder Client	UI คล้าย Postman สำหรับ REST

☐ **3. Live Reload และ Hot Reload**

Bun รองรับการพัฒนาแบบ Hot Reload โดยไม่ต้องใช้ nodemon

```
bun --hot index.ts
```

☐ **จุดเด่น:**

- ใช้ low-level file watcher
- รองรับ .ts, .js, .json, .env

- เปลี่ยนโค้ดแล้ว reload อัตโนมัติ
- ไม่มี dependency ภายนอก

☐ **4. Testing Tools (Built-in & Ecosystem)**

☐ **Built-in: bun test**

```
// math.test.ts
import { describe, it, expect } from "bun:test";

describe("Math", () => {
  it("adds correctly", () => {
    expect(2 + 2).toBe(4);
  });
});
```

```
bun test
```

☐ **จุดเด่น:**

- ทำงานเร็วมาก (ใช้ native code)
- API เหมือน Jest/Vitest
- ไม่ต้องติดตั้ง Babel หรือ transformers

☐ **ถ้าใช้ Jest หรือ Vitest ล่ะ?**

- ใช้ได้ถ้าจำเป็น (ผ่าน Babel หรือ esbuild)
- แนะนำให้ใช้ bun test ถ้าเริ่มต้นใหม่ เพราะเร็วกว่า

☐ **5. Bundling Tools**

☐ **Built-in: bun build**

```
bun build index.ts --outdir=dist
```

รองรับ:

- Minify
- Tree-shaking
- ES Modules
- CommonJS
- JSON/CSS/Image import

☐ **เปรียบเทียบกับ Vite/Webpack**

Feature	Bun Build	Vite/Webpack

Feature	Bun Build	Vite/Webpack
Speed	□ เร็วมาก	ปานกลาง
Config	ไม่ต้อง config	ต้อง config เยอะ
HMR	ยังไม่มี	มีเต็มรูปแบบ
ใช้กับ SPA	ได้บางส่วน	ได้เต็มที่

เหมาะสำหรับ SSR, API server, CLI tools มากกว่า SPA ณ ตอนนี้

⚙□ 6. Linting & Formatting

□ ESLint

bun add -d eslint

npx eslint --init

- รองรับโค้ด TypeScript/JavaScript
- แนะนำใช้ร่วมกับ VS Code ESLint Extension

□ Prettier

bun add -d prettier

สร้างไฟล์ .prettierrc:

```
{
  "semi": false,
  "singleQuote": true,
  "trailingComma": "es5"
}
```

แนะนำ run ผ่าน bunx เช่น:

bunx prettier --write .

□ 7. Docker และ DevContainer

Dockerfile (ตัวอย่าง):

```
FROM oven/bun:1.0.30

WORKDIR /app
COPY . .

RUN bun install
```

```
CMD ["bun", "index.ts"]
```

VS Code DevContainer:

```
// .devcontainer/devcontainer.json
{
  "name": "Bun Dev",
  "image": "oven/bun",
  "features": {
    "ghcr.io/devcontainers/features/node:1": {
      "version": "20"
    }
  },
  "postCreateCommand": "bun install"
}
```

ทำงานร่วมกับ Codespaces หรือ WSL2 ได้

□ 8. Dependency Management & Scripts

bun install

- ใช้ lockfile แบบ .bun.lockb (binary ⟶ load เร็ว)
- รองรับ npm, file:, github:, git:

bunx ⟶ เหมือน npx

```
bunx cowsay Hello!
```

ใช้เรียก binary ของแพ็กเกจได้ชั่วคราว เช่น prettier, eslint, tsc

---

□ 9. การพัฒนา API และ SSR ด้วย Bun

รองรับ:

- HTTP server ในตัว
- Bun.serve(...) ⟶ แบบ low-level
- รองรับ WebSockets
- รองรับ Middleware stack แบบ Express (ผ่าน 3rd party เช่น Hono, Elysia)

```
const server = Bun.serve({
  port: 3000,
  fetch(req) {
    return new Response("Hello Bun!");
  },
```

});

☐ 10. Monitoring และ Debugging

ยังไม่มีเครื่องมือแบบ Production Grade ที่ official แต่สามารถใช้:

- bun --hot + Logger
- Bun logger (custom)
- APM เช่น Datadog ผ่าน Node-compatible SDK
- Deno-style console.trace() ได้โดยตรง

☐ แนวโน้มในอนาคต (Dev Tools Roadmap)

Feature ที่คาดว่าจะมา	รายละเอียด
Bun Inspector	GUI Debugger
Bun Dev Server	SSR + SPA HMR สำหรับ Fullstack
Native Plugin System	Custom transformers, hooks
Language Server Protocol	ให้ VS Code เข้าใจ Bun มากขึ้น
Production Profiler	สำหรับ CPU & memory inspection

☐ สรุป

หมวด	เครื่องมือ
CLI	bun, bunx
Editor	VS Code, Vim, WebStorm
Formatter	Prettier
Linter	ESLint
Test	bun test, (รองรับ Jest แบบจำกัด)
Build	bun build
Hot reload	bun --hot
Docker	oven/bun base image
DevContainer	VS Code DevContainers

การติดตั้ง Bun.js บน Windows + ทดสอบด้วย IDE

☐ **สรุปสิ่งที่ต้องการ:**

สิ่งที่ต้องมี	รายละเอียด
OS	☐ Windows 10 หรือ 11 (64-bit)
Terminal	☐ Windows Terminal, PowerShell หรือ Git Bash
Package Manager	☐ Chocolatey หรือ Scoop
Editor	☐ Visual Studio Code
Shell	☐ PowerShell, Git Bash หรือ WSL (แนะนำหากใช้ Linux-based dev stack)

☐ **วิธีที่ 1: ติดตั้งผ่าน Scoop (ง่ายสุดและแนะนำ)**

☐ **ขั้นตอนที่ 1: ติดตั้ง Scoop (หากยังไม่มี)**

เปิด PowerShell ด้วยสิทธิ์ Admin แล้วรัน:

Set-ExecutionPolicy RemoteSigned -scope CurrentUser

irm get.scoop.sh | iex

เสร็จแล้วปิดแล้วเปิด PowerShell ใหม่

☐ **ขั้นตอนที่ 2: ติดตั้ง Bun ผ่าน Scoop**

scoop install bun

หากใช้ Git Bash หรือ Windows Terminal ก็ใช้คำสั่งเดียวกัน

☐ เมื่อติดตั้งเสร็จแล้วจะสามารถใช้คำสั่ง bun ได้จากทุกที่

☐ **วิธีที่ 2: ติดตั้งผ่าน MSI Installer (เหมาะกับผู้ไม่ใช้ package manager)**

☐ **ดาวน์โหลด:**

ไปที่เว็บไซต์ทางการของ Bun ⟶ https://bun.sh

คลิกปุ่ม **Install Bun** ⟶ เลือก **Windows Installer (.msi)**

ดาวน์โหลดแล้ว **ดับเบิลคลิกติดตั้งตามขั้นตอน**

☐ **ตรวจสอบการติดตั้ง**

☐ **ตรวจเวอร์ชัน**

เปิด PowerShell หรือ Terminal แล้วพิมพ์:

bun --version

ตัวอย่างผลลัพธ์: