

# BMAD

## BMad Method - Hướng Dẫn Toàn Diện Bằng Tiếng Việt

| Phương pháp phát triển phần mềm hiện đại với sự hỗ trợ của 7 chuyên gia AI

### Section 1: Tổng Quan BMad Method

#### BMad Method là gì?

BMad Method là một phương pháp phát triển phần mềm kết hợp:

- **Agile principles** - Phát triển linh hoạt theo từng giai đoạn nhỏ
- **7 AI agents chuyên nghiệp** - Mỗi agent phụ trách một công việc cụ thể
- **Document-driven development** - Tất cả thông tin được lưu trong tài liệu
- **Context persistence** - Agents nhớ được công việc dù bạn tắt máy

#### Tại sao nên dùng BMad Method?

Ưu điểm so với cách làm thông thường:

1. **Có quy trình rõ ràng** - Không bị lạc hướng, biết làm gì tiếp theo
2. **Mỗi agent chuyên về một việc** - Chất lượng cao, không bị lẫn lộn
3. **Kết quả đồng nhất** - Dùng templates chuẩn, không bị sai lệch
4. **Nhớ được context** - Agent biết bạn đang làm gì dù tắt máy
5. **Có quality assurance** - Luôn có người review code trước khi hoàn thành

So sánh với các cách khác:

Đặc điểm	BMad Method	AI thông thường	Làm thủ công
Quy trình	Có cấu trúc rõ ràng	Tùy hứng	Tùy kinh nghiệm
Chất lượng	Đồng nhất	Không ổn định	Tùy người
Nhớ context	Lưu trong file	Quên khi tắt	Phải ghi chép
Phân công	Rõ ràng từng vai trò	Một AI làm hết	Một người làm hết

#### Khi nào nên dùng BMad Method?

Nên dùng khi:

- Dự án mất từ 3 tuần trở lên
- Cần tài liệu chi tiết để bảo trì sau này
- Làm việc nhóm hoặc cần maintain lâu dài
- Muốn code chất lượng cao

- Dự án có khả năng mở rộng sau này

#### **Không cần dùng khi:**

- Làm prototype nhanh trong vài ngày
- Script đơn giản một lần dùng
- Học code cá nhân
- Chỉ cần chứng minh ý tưởng

## **7 Chuyên Gia AI Trong Team**

### **Analyst - Cô Mai:** Chuyên nghiên cứu

- Nghiên cứu thị trường, phân tích đối thủ
- Brainstorming ý tưởng
- Tạo project brief từ ý tưởng ban đầu

### **Product Manager - Anh Huy:** Quản lý sản phẩm

- Tạo PRD (Product Requirements Document)
- Định nghĩa tính năng và yêu cầu
- Lập kế hoạch phát triển

### **UX Expert - Chị Lan:** Thiết kế trải nghiệm

- Thiết kế giao diện người dùng
- Tạo wireframe và user flow
- Tối ưu trải nghiệm người dùng

### **Scrum Master - Anh Minh:** Quản lý quy trình

- Tạo user stories từ PRD
- Lập kế hoạch sprint
- Quản lý backlog

### **Product Owner - Chị Linh:** Kiểm tra chất lượng

- Validate tài liệu và stories
- Chia nhỏ tài liệu lớn
- Đảm bảo tính nhất quán

### **Developer - Anh Tuấn:** Lập trình viên

- Code tính năng theo user stories
- Viết test cases
- Debug và fix lỗi

### **QA Engineer - Anh Quang:** Kiểm tra chất lượng code

- Review code như senior developer
  - Refactor code để tốt hơn
  - Đảm bảo coding standards
- 

## Section 2: Workflow Từ Ý Tưởng đến Sản Phẩm

### Quy trình 2 giai đoạn

BMad Method chia làm 2 giai đoạn chính:

#### Giai đoạn 1: Lập Kế Hoạch (Planning)

- Nghiên cứu ý tưởng
- Tạo tài liệu yêu cầu
- Thiết kế hệ thống
- Chuẩn bị user stories

#### Giai đoạn 2: Phát Triển (Development)

- Code từng tính năng
- Test và review
- Deploy sản phẩm

### Giai đoạn 1: Lập Kế Hoạch

#### Bước 1: Nghiên cứu ý tưởng (30-60 phút)

Ý tưởng → /analyst → Project Brief

- **Input:** Ý tưởng ban đầu của bạn
- **Công việc:** Nghiên cứu thị trường, phân tích đối thủ
- **Output:** Tài liệu mô tả dự án (Project Brief)

#### Bước 2: Tạo yêu cầu sản phẩm (1-2 tiếng)

Project Brief → /pm → PRD Complete

- **Input:** Project Brief từ bước 1
- **Công việc:** Tạo PRD với tính năng, yêu cầu kỹ thuật, user stories
- **Output:** Product Requirements Document

#### Bước 3: Thiết kế UX (1-3 tiếng, tùy chọn)

PRD → /ux-expert → UI/UX Specifications

- **Input:** Yêu cầu từ PRD
- **Công việc:** Thiết kế giao diện, wireframe, user flow
- **Output:** Tài liệu thiết kế frontend

#### Bước 4: Thiết kế hệ thống (1-2 tiếng)

PRD + UX Specs → /architect → Architecture Document

- **Input:** PRD và tài liệu UX
- **Công việc:** Chọn công nghệ, thiết kế database, API
- **Output:** Tài liệu kiến trúc hệ thống

#### Bước 5: Kiểm tra và chuẩn bị (30 phút)

All Documents → /po → Ready for Development

- **Input:** Tất cả tài liệu từ các bước trước
- **Công việc:** Kiểm tra tính đầy đủ, chia nhỏ tài liệu
- **Output:** Tài liệu sẵn sàng để code

### Giai đoạn 2: Phát Triển

Chu trình lặp lại cho từng tính năng:

#### Bước 1: Tạo User Story

PRD → /sm → Detailed User Story

- Anh Minh đọc PRD và tạo story chi tiết
- Mỗi story bao gồm: mô tả, acceptance criteria, tasks

#### Bước 2: Kiểm tra Story

Draft Story → /po → Validated Story

- Chị Linh kiểm tra story có đủ thông tin không
- Đảm bảo story phù hợp với PRD

#### Bước 3: Lập trình

Validated Story → /dev → Working Code + Tests

- Anh Tuấn đọc story và code từng task
- Viết test cases và chạy thử
- Cập nhật tiến độ vào story file

#### Bước 4: Review chất lượng

Completed Code → /qa → Production Ready

- Anh Quang review code như senior
- Refactor code để tốt hơn
- Đảm bảo coding standards

**Lặp lại chu trình này cho đến khi hoàn thành tất cả tính năng**

### Ước tính thời gian

#### Dự án nhỏ (1-2 tính năng):

- Lập kế hoạch: 2-4 tiếng
- Phát triển: 1-2 tuần
- Tổng cộng: 1-2 tuần

#### Dự án trung bình (5-10 tính năng):

- Lập kế hoạch: 4-8 tiếng
- Phát triển: 3-6 tuần
- Tổng cộng: 1-2 tháng

#### Dự án lớn (20+ tính năng):

- Lập kế hoạch: 1-2 ngày
- Phát triển: 2-6 tháng
- Tổng cộng: 3-6 tháng

## Section 3: Các Tình Huống Thực Tế

### Tình huống 1: Làm dự án mới từ đầu

**Khi nào:** Bạn có ý tưởng mới, chưa có code gì

**Quy trình:**

Ý tưởng → Nghiên cứu → Tạo PRD → Thiết kế UX → Kiến trúc → Code → Deploy

### Các lệnh cần dùng:

```
# Bước 1: Nghiên cứu ý tưởng
/analyst
*create-project-brief

# Bước 2: Tạo PRD
/pm
*create-prd

# Bước 3: Thiết kế UX (nếu cần)
/ux-expert
*create-front-end-spec

# Bước 4: Thiết kế hệ thống
/architect
*create-architecture

# Bước 5: Chuẩn bị development
/po
*execute-checklist-po
*shard-doc docs/prd.md docs/prd
*shard-doc docs/architecture.md docs/architecture

# Bước 6: Chu trình development
/sm → *draft (tạo stories)
/dev → *develop-story (code)
/qa → *review (kiểm tra)
```

**Thời gian:** Lập kế hoạch 4-8 tiếng, phát triển tùy scope

## Tình huống 2: Nâng cấp dự án có sẵn

**Khi nào:** Bạn có app/website rồi, muốn thêm tính năng

### Đặc điểm:

- Code đã có sẵn
- Kiến trúc đã định hình
- Cần tích hợp với hệ thống hiện tại

### Quy trình:

App hiện tại → Phân tích → Tạo epic mới → Code → Deploy

### Các lệnh cần dùng:

```
# Bước 1: Phân tích hệ thống hiện tại
/analyst
*document-project

# Bước 2: Tạo epic cho tính năng mới
/pm
*create-brownfield-epic

# Bước 3: Kiểm tra tính khả thi
/po
*validate-story-draft

# Bước 4: Development
/sm → *draft
/dev → *develop-story
/qa → *review
```

**Thời gian:** Nhanh hơn làm mới 20-50%

### Tình huống 3: Sửa lỗi và bảo trì

**Khi nào:** App có bug cần fix hoặc cần tối ưu performance

**Quy trình:**

```
Bug report → Tạo story sửa lỗi → Code fix → Review → Deploy
```

**Các lệnh cần dùng:**

```
# Bước 1: Tạo story mô tả bug
/sm
*draft # Mô tả bug và cách fix

# Bước 2: Code fix
/dev
*develop-story # Fix lỗi và viết test

# Bước 3: Review kỹ lưỡng
/qa
*review # Đảm bảo fix không gây lỗi khác
```

**Thời gian:** Bug đơn giản 2-4 tiếng, phức tạp 1-3 ngày

### Tình huống 4: Cải tiến tính năng

**Khi nào:** Muốn làm tính năng hiện tại tốt hơn hoặc tăng performance

**Quy trình:**

Phân tích hiện trạng → Lập kế hoạch cải tiến → Code → Deploy

### Các lệnh cần dùng:

```
# Bước 1: Phân tích tình trạng hiện tại
/analyst
*brainstorm "cải tiến tính năng X"

# Bước 2: Lập kế hoạch cải tiến
/pm
*create-brownfield-story

# Bước 3: Thực hiện cải tiến
/dev
*develop-story

# Bước 4: Kiểm tra hiệu quả
/qa
*review # Đảm bảo cải tiến đạt mục tiêu
```

**Thời gian:** Cải tiến nhỏ 4-8 tiếng, lớn 1-2 tuần

### Tình huống 5: Phát triển liên tục

**Khi nào:** Dự án dài hạn với nhiều releases theo kế hoạch

### Chu trình hàng tháng:

Sprint planning → Development → QA → Release → Retrospective

### Quy trình chi tiết:

```
# Tuần 1: Lập kế hoạch sprint
/po → *execute-checklist-po
/sm → *draft # Tạo nhiều stories cho sprint

# Tuần 2-3: Development
/dev → *develop-story # Story 1
/qa → *review
/dev → *develop-story # Story 2
/qa → *review

# Tuần 4: Release và tổng kết
/po → validation và documentation
Deploy lên production
/sm → retrospective và plan sprint tiếp
```

### Lợi ích:



- Tiến độ đều đặn có thể dự đoán
  - Chất lượng được cải thiện liên tục
  - Feedback thường xuyên từ user
  - Nhịp độ phát triển bền vững
- 

## Section 4: Hướng Dẫn Từng Bước Chi Tiết

### Bắt đầu sử dụng BMad Method

#### Bước 1: Cài đặt

```
# Tạo project mới
mkdir du-an-cua-toi
cd du-an-cua-toi

# Cài đặt BMad Method
npm install bmad-method

# Hoặc copy từ project có sẵn
cp -r /path/to/bmad-project/.bmad-core .
cp -r /path/to/bmad-project/.claude .
```

#### Bước 2: Kiểm tra cài đặt

```
# Test các agent có hoạt động không
/analyst # Phải hiện "Xin chào! Tôi là Cô Mai..."
*exit

/pm # Phải hiện "Xin chào! Tôi là Anh Huy..."
*exit
```

#### Bước 3: Cấu hình project

```
# Chỉnh sửa config chính
nano .bmad-core/core-config.yaml

# Tùy chỉnh technical preferences
nano .bmad-core/data/technical-preferences.md
```

### Các lệnh cơ bản theo từng agent

#### Analyst - Cô Mai (Nghiên cứu):

Đề vào chế độ Analyst:

```
/analyst
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*create-project-brief # Tạo tài liệu mô tả dự án từ ý tưởng
*perform-market-research # Nghiên cứu thị trường cho sản phẩm
*create-competitor-analysis # Phân tích các đối thủ cạnh tranh
*brainstorm "chủ đề" # Brainstorming về chủ đề cụ thể
*research-prompt "chủ đề" # Tạo câu hỏi nghiên cứu sâu
*exit # Thoát khỏi chế độ Analyst
```

**Ví dụ sử dụng:**

```
/analyst
*brainstorm "app todo có AI"
# Cô Mai sẽ hướng dẫn brainstorming session
# Kết quả: Danh sách tính năng, user personas, cơ hội thị trường

*create-project-brief
# Quy trình tương tác tạo project brief
# Kết quả: File docs/project-brief.md
```

**Product Manager - Anh Huy (Quản lý sản phẩm):**

Đổi vào chế độ PM:

```
/pm
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*create-prd # Tạo Product Requirements Document
*create-brownfield-prd # Tạo PRD cho dự án có sẵn
*shard-prd # Chia PRD thành nhiều file nhỏ
*create-epic # Tạo epic mới
*create-story # Tạo user story
*correct-course # Điều chỉnh hướng dự án
*yolo # Bật chế độ nhanh (bỏ qua confirmations)
*exit # Thoát khỏi chế độ PM
```

**Ví dụ sử dụng:**

```
/pm
*create-prd
# Quy trình tương tác tạo PRD
# Hỏi về mục tiêu, yêu cầu, tính năng
# Kết quả: File docs/prd.md

*shard-prd
# Chia PRD thành nhiều file nhỏ
# Kết quả: docs/prd/goals.md, docs/prd/features.md, v.v.
```

### UX Expert - Chị Lan (Thiết kế UX):

Đề vào chế độ UX Expert:

```
/ux-expert
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*create-front-end-spec # Tạo tài liệu thiết kế frontend
*generate-ui-prompt # Tạo prompts cho AI tạo UI
*exit # Thoát khỏi chế độ UX Expert
```

### Ví dụ sử dụng:

```
/ux-expert
*create-front-end-spec
# Quy trình thiết kế UX tương tác
# Kết quả: docs/frontend-spec.md với wireframes, user flows

*generate-ui-prompt
# Tạo prompts tối ưu cho v0.dev hoặc Lovable
# Kết quả: Prompts chất lượng cao cho AI tạo UI
```

### Scrum Master - Anh Minh (Quản lý quy trình):

Đề vào chế độ Scrum Master:

```
/sm
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*draft # Tạo user story tiếp theo từ epic
*correct-course # Điều chỉnh hướng dự án
*story-checklist # Kiểm tra chất lượng story
*exit # Thoát khỏi chế độ SM
```

### Ví dụ sử dụng:

```
/sm
*draft
# Đọc PRD và architecture
# Tạo user story chi tiết với acceptance criteria
# Kết quả: docs/stories/story-1-1.md
```

### Product Owner - Chị Linh (Kiểm tra chất lượng):

Đổi vào chế độ Product Owner:

```
/po
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*execute-checklist-po # Chạy master checklist
*shard-doc "tài liệu" "thư mục đích" # Chia tài liệu thành parts
*validate-story-draft "story" # Kiểm tra chất lượng story
*create-epic # Tạo epic cho brownfield
*correct-course # Điều chỉnh course
*yolo # Bật/tắt confirmations
*exit # Thoát khỏi chế độ PO
```

### Ví dụ sử dụng:

```
/po
*shard-doc docs/prd.md docs/prd
# Tự động chia PRD thành:
# docs/prd/goals.md
# docs/prd/features.md
# docs/prd/technical-requirements.md

*validate-story-draft docs/stories/story-1-1.md
# Kiểm tra story có đầy đủ thông tin không
# Validate với yêu cầu trong PRD
```

### Developer - Anh Tuấn (Lập trình viên):

Đổi vào chế độ Developer:

```
/dev
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*develop-story # Implement story từ đầu đến cuối
*run-tests # Chạy tests và linting
*explain # Giải thích code vừa viết
*exit # Thoát khỏi chế độ Dev
```

### Ví dụ sử dụng:

```
/dev
*develop-story
# Đọc story requirements
# Implement từng task theo thứ tự
# Viết tests
# Cập nhật tiến độ vào story
# Đánh dấu story "Ready for Review"
```

### QA Engineer - Anh Quang (Kiểm tra chất lượng code):

Đề vào chế độ QA:

```
/qa
```

Các lệnh có thể dùng:

```
*help # Hiện tất cả lệnh có sẵn
*review "story" # Review code và refactor
*exit # Thoát khỏi chế độ QA
```

### Ví dụ sử dụng:

```
/qa
*review docs/stories/story-1-1.md
# Review code từ dev
# Active refactoring với giải thích chi tiết
# Cập nhật "QA Results" section
# Đánh dấu "Approved" hoặc "Changes Required"
```

## Quy trình làm việc thông thường

Tạo tính năng mới hoàn chỉnh:

```
# 1. Lập kế hoạch tính năng
/analyst
*brainstorm "hệ thống đăng nhập người dùng"

# 2. Tạo yêu cầu
/pm
*create-prd

# 3. Thiết kế UX (nếu cần)
/ux-expert
*create-front-end-spec

# 4. Tạo stories
/sm
*draft

# 5. Kiểm tra story
/po
*validate-story-draft docs/stories/story-1-1.md

# 6. Lập trình
/dev
*develop-story

# 7. Review chất lượng
/qa
*review docs/stories/story-1-1.md
```

#### Sửa bug nhanh:

```
# 1. Tạo bug story
/sm
*draft # Mô tả bug và cách fix

# 2. Implement fix
/dev
*develop-story # Fix + tests

# 3. Senior review
/qa
*review # Đảm bảo không gây lỗi khác
```

#### Kiểm tra tình trạng dự án:

```
# Kiểm tra document alignment
/po
*execute-checklist-po

# Review chất lượng code
/qa
*review # Review stories gần đây

# Lập kế hoạch iteration tiếp
/sm
*draft # Tạo stories tiếp theo
```

## Cấu trúc thư mục

```
du-an-cua-toi/
├── .bmad-core/                # Core files của BMad Method
│   ├── agents/               # Định nghĩa agents
│   ├── tasks/                # Workflow tasks
│   ├── templates/            # Document templates
│   ├── data/                 # Knowledge base
│   └── core-config.yaml      # Config chính
├── .claude/                  # Config của Claude Code
│   └── commands/BMad/        # Định nghĩa lệnh agents
├── docs/                     # Tài liệu dự án
│   ├── prd.md                # Product Requirements
│   ├── architecture.md       # System Architecture
│   ├── prd/                  # PRD được chia nhỏ
│   ├── architecture/         # Architecture được chia nhỏ
│   └── stories/               # User stories
├── src/                      # Source code
└── README.md                 # Tổng quan dự án
```

## Cấu hình quan trọng

### Core Config (.bmad-core/core-config.yaml):

```
# Files mà Dev agent luôn phải đọc
devLoadAlwaysFiles:
  - docs/architecture/coding-standards.md
  - docs/architecture/tech-stack.md
  - docs/architecture/project-structure.md

# Cấu hình PRD
prd:
  prdFile: docs/prd.md
  prdSharded: true
  prdShardedLocation: docs/prd
```

```
# Cấu hình Architecture
architecture:
  architectureFile: docs/architecture.md
  architectureSharded: true
  architectureShardedLocation: docs/architecture

# Vị trí lưu stories
devStoryLocation: docs/stories
```

### Technical Preferences (.bmad-core/data/technical-preferences.md):

```
# Technical Preferences

## Frontend
- Framework: React với TypeScript
- State Management: Zustand
- Styling: Tailwind CSS
- UI Components: shadcn/ui

## Backend
- Runtime: Node.js
- Framework: Express.js
- Database: PostgreSQL
- ORM: Prisma

## Testing
- Unit Tests: Jest
- Integration Tests: Supertest
- E2E Tests: Playwright

## Deployment
- Platform: Vercel (Frontend) + Railway (Backend)
- Database: Neon PostgreSQL
```

## Mẹo tăng năng suất

### Tăng tốc độ làm việc:

#### 1. Dùng YOLO Mode khi cần nhanh:

```
/po
*yolo # Bỏ qua confirmations
```

#### 2. Làm nhiều công việc cùng lúc:



```
/sm
*draft # Tạo nhiều stories
*draft
*draft
```

### 3. Kiểm tra định kỳ:

```
/po
*execute-checklist-po # Kiểm tra hàng tuần
```

### Đảm bảo chất lượng:

#### 1. Luôn validate stories:

```
/po
*validate-story-draft "story" # Trước khi dev
```

#### 2. Dùng QA cho mọi code:

```
/qa
*review # Không bao giờ bỏ qua QA review
```

#### 3. Giữ tài liệu cập nhật:

```
/po
*shard-doc # Sau khi thay đổi PRD lớn
```

### Tối ưu quy trình làm việc:

#### 1. Bắt đầu nhỏ, mở rộng dần:

- Làm 1-2 tính năng cốt lõi trước
- Thêm complexity từ từ
- Thu thập feedback thường xuyên

#### 2. Duy trì chất lượng tài liệu:

- Tài liệu chia nhỏ load nhanh hơn
- Cập nhật technical preferences thường xuyên
- Review architecture định kỳ

#### 3. Tận dụng điểm mạnh từng agent:

- Analyst: Nghiên cứu rộng và ideation
- PM: Yêu cầu có cấu trúc
- Dev: Implementation tập trung
- QA: Chất lượng và mentorship

## Xử lý sự cố thường gặp

### Vấn đề: Agent không nhớ context

```
# Giải pháp: Kiểm tra cấu trúc tài liệu
/po
*execute-checklist-po # Verify documents có aligned không
```

### Vấn đề: Stories thiếu chi tiết

```
# Giải pháp: Validation tốt hơn
/po
*validate-story-draft "story" # Trước khi development
```

### Vấn đề: Chất lượng code không đồng nhất

```
# Giải pháp: Luôn dùng QA
/qa
*review # Không bỏ qua reviews
```

### Vấn đề: Development quá chậm

```
# Giải pháp: Kiểm tra kích thước story
/sm
*draft # Chia nhỏ stories lớn
```

### Tối ưu hiệu suất:

#### 1. Document sharding:

```
/po
*shard-doc docs/prd.md docs/prd
```

#### 2. Dùng technical preferences:

- Cập nhật .bmad-core/data/technical-preferences.md
- Agents sẽ theo preferences này

#### 3. Dọn dẹp định kỳ:

- Xóa files không dùng
- Archive stories đã hoàn thành
- Cập nhật core config

---

## Section 5: Ví Dụ Thực Tế - Xây Dựng Todo App + Trello Integration

## Tổng quan dự án

**Ý tưởng:** Xây dựng ứng dụng quản lý công việc kết hợp tốt nhất của Todo app truyền thống và tính năng collaboration của Trello.

**Mục tiêu:** Tạo ra một ứng dụng web hoàn chỉnh từ con số 0, deploy lên production và có thể sử dụng thực tế.

**Timeline dự kiến:** 4-6 tuần (part-time)

## Giai đoạn 1: Lập Kế Hoạch (Planning Phase)

**Thời gian:** 4-6 tiếng trong 1-2 ngày

### Bước 1: Nghiên cứu và Brainstorming (45 phút)

**Bắt đầu với Analyst:**

```
/analyst
```

**Lệnh đầu tiên:**

```
*brainstorm "todo app kết hợp trello cho team nhỏ"
```

**Cô Mai sẽ hỏi:**

- "Bạn muốn ứng dụng này phục vụ ai chủ yếu?"
- "Điểm khác biệt với Todoist, Asana hiện tại là gì?"
- "Team size bao nhiêu người?"

**Bạn trả lời ví dụ:**

- "Team 3-10 người, startup hoặc agency nhỏ"
- "Cần đơn giản hơn Asana nhưng collaborative hơn todo app thường"
- "Tích hợp Trello để sync được với workflow hiện tại"

**Kết quả từ brainstorming:**

- Target users: Small teams, freelancers, startups
- Core value: Simplicity + Collaboration
- Key differentiator: Trello integration
- MVP scope: Personal todos + team boards + Trello sync

**Tiếp theo, tạo Project Brief:**

```
*create-project-brief
```

### Cô Mai sẽ hướng dẫn tạo:

```
# TodoFlow - Project Brief
```

#### ## Problem Statement

Small teams cần tool quản lý task đơn giản nhưng có khả năng collaborate. Các tool hiện tại hoặc quá phức tạp (Asana) hoặc quá cá nhân (simple todo apps).

#### ## Target Users

- Startup teams (3-10 người)
- Creative agencies
- Freelance teams
- Remote teams cần sync work

#### ## Core Solution

Web app kết hợp personal todo management với team collaboration, tích hợp Trello để leverage existing workflows.

#### ## Success Metrics

- User có thể tạo và manage personal todos trong 30 giây
- Team có thể collaborate trên shared boards
- Trello sync hoạt động realtime
- 95% uptime, responsive design

### Thoát khỏi Analyst:

```
*exit
```

## Bước 2: Tạo Product Requirements Document (2 tiếng)

### Chuyển sang Product Manager:

```
/pm
```

### Tạo PRD chi tiết:

```
*create-prd
```

### Anh Huy sẽ hỏi về Project Brief:

- "Đã có Project Brief chưa?"
- Bạn: "Có - docs/project-brief.md"

### Anh Huy sẽ dẫn dắt qua từng section:

#### Goals & Background:

- Tạo todo app hybrid personal/team với Trello integration

- Serve small teams cần simplicity + collaboration
- MVP launch trong 6 tuần

### Functional Requirements:

- FR1: User authentication (email/password)
- FR2: Personal todo CRUD với priority, due date
- FR3: Team boards với member invitation
- FR4: Real-time collaboration trên shared todos
- FR5: Trello board import/sync
- FR6: Basic notifications
- FR7: Mobile responsive design

### Non-Functional Requirements:

- NFR1: Load time < 2 giây
- NFR2: 99% uptime
- NFR3: Support 100+ concurrent users
- NFR4: GDPR compliant data handling
- NFR5: Works trên mobile browsers

### UI Design Goals:

- Clean, minimal interface như Todoist
- Drag & drop như Trello
- Fast keyboard shortcuts
- Dark/light theme support

### Technical Assumptions:

- React + TypeScript frontend
- Node.js + Express backend
- PostgreSQL database
- Real-time với [Socket.io](https://socket.io/)
- Deploy trên Vercel + Railway
- Trello API integration

### Epic List:

1. **Epic 1: Foundation & Auth** - User registration, login, basic profile
2. **Epic 2: Personal Todo Management** - CRUD todos, priorities, filters
3. **Epic 3: Team Collaboration** - Shared boards, member management
4. **Epic 4: Real-time Features** - Live updates, notifications
5. **Epic 5: Trello Integration** - Import boards, bi-directional sync

## 6. **Epic 6: Polish & Deploy** - Performance, testing, production deploy

### **Epic Details sẽ có User Stories cụ thể:**

#### **Epic 1: Foundation & Auth**

- Story 1.1: User Registration với email verification
- Story 1.2: Login/logout với session management
- Story 1.3: User profile và settings
- Story 1.4: Password reset flow

#### **Epic 2: Personal Todo Management**

- Story 2.1: Create/edit/delete todos với rich text
- Story 2.2: Todo priorities (High, Medium, Low) và due dates
- Story 2.3: Todo categories/tags và filtering
- Story 2.4: Todo search và sorting options

### **Và cứ thế cho đến Epic 6...**

### **Anh Huy sẽ tạo file:**

```
docs/prd.md - 50+ pages chi tiết
```

### **Kết thúc PM phase:**

```
*exit
```

## **Bước 3: UX Design (1 tiếng, optional nhưng recommended)**

### **Chuyển sang UX Expert:**

```
/ux-expert
```

### **Tạo Frontend Specification:**

```
*create-front-end-spec
```

### **Chị Lan sẽ thiết kế:**

### **Overall UX Vision:**

- "Todoist meets Trello" - Simple cá nhân, powerful khi collaborate
- Progressive disclosure - bắt đầu simple, unlock team features

### **Key Interaction Paradigms:**

- Quick add todo với Cmd+K
- Drag & drop organizing
- Keyboard shortcuts cho power users
- Context menus cho advanced actions

### Core Screens:

1. **Dashboard** - Overview của personal + team todos
2. **Personal Todos** - List/board view của tasks cá nhân
3. **Team Board** - Shared workspace với members
4. **Settings** - Profile, preferences, integrations

### Wireframes (text-based):

Dashboard Layout:

+-----+-----+	
Personal	Team Boards
[ + Add Todo ]	[ + New Board]
Todo 1 [H]	Board 1 (3)
Todo 2 [M]	Board 2 (7)
Todo 3 [L]	
+-----+-----+	

### Mobile Responsiveness:

- Bottom tab navigation
- Swipe gestures for actions
- Optimized touch targets

### Chỉ Lan tạo file:

```
docs/frontend-spec.md
```

### Generate UI prompts cho v0/Lovable:

```
*generate-ui-prompt
```

### Kết quả:

Optimized prompts để tạo UI components:

- "Create a clean todo app dashboard với personal and team sections..."
- "Design a todo item component với priority indicators và drag handles..."

### Thoát UX Expert:

```
*exit
```

## Bước 4: System Architecture (1.5 tiếng)

### Chuyển sang Architect:

```
/architect
```

### Tạo Architecture Document:

```
*create-architecture
```

### Architect sẽ thiết kế:

#### Tech Stack:

##### Frontend:

- React 18 với TypeScript
- Zustand cho state management
- Tailwind CSS + shadcn/ui components
- React Router cho navigation
- React Query cho API calls
- Socket.io-client cho realtime

##### Backend:

- Node.js 20+ với Express
- TypeScript end-to-end
- PostgreSQL với Prisma ORM
- Socket.io cho realtime
- JWT authentication
- Trello API integration

##### Infrastructure:

- Frontend: Vercel deployment
- Backend: Railway deployment
- Database: Neon PostgreSQL
- File storage: Cloudinary (if needed)
- Monitoring: Sentry error tracking

### Database Schema:

```
Users: id, email, password_hash, name, avatar_url, created_at
Todos: id, user_id, title, description, priority, due_date, completed, board_id
Boards: id, name, owner_id, is_personal, trello_board_id, created_at
BoardMembers: board_id, user_id, role, joined_at
TrelloSync: board_id, trello_board_id, last_sync, sync_enabled
```



## API Design:

### Authentication:

```
POST /auth/register - User registration
POST /auth/login - User login
POST /auth/logout - User logout
GET /auth/me - Get current user
```

### Todos:

```
GET /todos - List user todos với filters
POST /todos - Create new todo
PUT /todos/:id - Update todo
DELETE /todos/:id - Delete todo
```

### Boards:

```
GET /boards - List user boards
POST /boards - Create team board
PUT /boards/:id - Update board
POST /boards/:id/members - Invite member
```

### Trello Integration:

```
POST /trello/connect - Connect Trello account
GET /trello/boards - List Trello boards
POST /trello/import/:boardId - Import Trello board
```

## Security Considerations:

- JWT tokens với refresh token rotation
- Rate limiting trên API endpoints
- Input validation và sanitization
- CORS configuration
- Environment variables cho secrets

## Deployment Architecture:

```
Frontend (Vercel) → Backend (Railway) → Database (Neon)
↓
Trello API
```

## File được tạo:

```
docs/architecture.md - 30+ pages
```

## Thoát Architect:

```
*exit
```

## Bước 5: Document Validation & Sharding (30 phút)

Chuyển sang Product Owner:

```
/po
```

Chạy master checklist:

```
*execute-checklist-po
```

Chị Linh sẽ kiểm tra:

- PRD có align với Project Brief không?
- Architecture có cover hết functional requirements không?
- Epic sequencing có hợp lý không?
- Dependencies giữa các components đã clear chưa?

Sau khi validate, shard documents:

```
*shard-doc docs/prd.md docs/prd
```

Kết quả:

```
docs/prd/  
├── index.md - Overview  
├── goals-and-background.md  
├── functional-requirements.md  
├── non-functional-requirements.md  
├── ui-design-goals.md  
├── technical-assumptions.md  
├── epic-1-foundation-auth.md  
├── epic-2-personal-todo-management.md  
├── epic-3-team-collaboration.md  
├── epic-4-realtime-features.md  
├── epic-5-trello-integration.md  
└── epic-6-polish-deploy.md
```

Shard architecture:

```
*shard-doc docs/architecture.md docs/architecture
```

Kết quả:

```
docs/architecture/
├── index.md
├── tech-stack.md
├── database-design.md
├── api-design.md
├── security-considerations.md
├── deployment-architecture.md
├── coding-standards.md
└── project-structure.md
```

### Thoát PO:

```
*exit
```

## Giai đoạn 2: Development Phase

**Thời gian:** 4-5 tuần (part-time, ~15-20 tiếng/tuần)

### Sprint 1: Foundation & Authentication (Tuần 1)

#### Story 1.1: User Registration

#### Tạo story với SM:

```
/sm
*draft
```

#### Anh Minh đọc Epic 1 và tạo:

```
# Story 1.1: User Registration

## Story
As a new user,
I want to register for an account with email and password,
So that I can start using TodoFlow to manage my tasks.

## Acceptance Criteria
1. User can access registration form at /register
2. Form includes email, password, confirm password, display name fields
3. Email validation ensures valid format
4. Password validation: min 8 characters, 1 uppercase, 1 number
5. Password confirmation must match
6. Display appropriate error messages for validation failures
7. Successful registration sends verification email
8. User redirected to "check email" page after registration
9. Email verification link activates account
10. Verified users can login immediately
```

## ## Tasks

- [ ] Setup project structure với React + TypeScript
- [ ] Create registration form component
- [ ] Implement client-side validation
- [ ] Setup backend API endpoint POST /auth/register
- [ ] Implement password hashing với bcrypt
- [ ] Setup email service với SendGrid/Resend
- [ ] Create email verification system
- [ ] Add error handling và loading states
- [ ] Write unit tests cho form validation
- [ ] Write integration tests cho registration flow

### Validate story với PO:

```
/po
*validate-story-draft docs/stories/story-1-1.md
```

### Chị Linh kiểm tra và approve:

"Story đầy đủ thông tin, acceptance criteria rõ ràng, có thể implement được."

### Implement story với Developer:

```
/dev
*develop-story
```

### Anh Tuấn sẽ làm từng task:

#### Task 1: Setup project structure

```
# Frontend setup
npx create-react-app todoflow-frontend --template typescript
cd todoflow-frontend
npm install zustand react-router-dom @tanstack/react-query
npm install -D tailwindcss @types/node

# Backend setup
mkdir todoflow-backend
cd todoflow-backend
npm init -y
npm install express typescript prisma @prisma/client bcrypt jsonwebtoken
npm install -D @types/express @types/bcrypt @types/jsonwebtoken ts-node nodemon

# Database setup
npx prisma init
```

#### Task 2: Create registration form component

```

// src/components/auth/RegistrationForm.tsx
import React, { useState } from 'react';

interface FormData {
  email: string;
  password: string;
  confirmPassword: string;
  displayName: string;
}

export const RegistrationForm: React.FC = () => {
  const [formData, setFormData] = useState<FormData>({
    email: '', password: '', confirmPassword: '', displayName: ''
  });
  const [errors, setErrors] = useState<Partial<FormData>>({});
  const [loading, setLoading] = useState(false);

  const validateForm = () => {
    const newErrors: Partial<FormData> = {};
    // Email validation
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(formData.email)) {
      newErrors.email = 'Please enter a valid email address';
    }

    // Password validation
    const passwordRegex = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d@$!%*?&]{8,}$/;
    if (!passwordRegex.test(formData.password)) {
      newErrors.password = 'Password must be 8+ characters with uppercase, lowercase, and
number';
    }

    // Confirm password
    if (formData.password !== formData.confirmPassword) {
      newErrors.confirmPassword = 'Passwords do not match';
    }

    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!validateForm()) return;

    setLoading(true);
    try {
      const response = await fetch('/api/auth/register', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(formData)
      });
    } catch {
      // Error handling
    }
  };
};

```

```

});

if (response.ok) {
  // Redirect to check email page
  window.location.href = '/check-email';
} else {
  const error = await response.json();
  setErrors({ email: error.message });
}
} catch (error) {
  setErrors({ email: 'Registration failed. Please try again.' });
} finally {
  setLoading(false);
}
};

return (
  <form onSubmit={handleSubmit} className="max-w-md mx-auto p-6 bg-white rounded-lg shadow">
    <h2 className="text-2xl font-bold mb-6 text-center">Create Account</h2>
    <div className="mb-4">
      <label className="block text-sm font-medium mb-2">Display Name</label>
      <input
        type="text"
        value={formData.displayName}
        onChange={(e) => setFormData({ ...formData, displayName: e.target.value })}
        className="w-full px-3 py-2 border rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500"
        required
      />
    </div>

    <div className="mb-4">
      <label className="block text-sm font-medium mb-2">Email</label>
      <input
        type="email"
        value={formData.email}
        onChange={(e) => setFormData({ ...formData, email: e.target.value })}
        className="w-full px-3 py-2 border rounded-md focus:outline-none focus:ring-2 focus:ring-blue-500"
        required
      />
      {errors.email && <p className="text-red-500 text-sm mt-1">{errors.email}</p>}
    </div>

    <div className="mb-4">
      <label className="block text-sm font-medium mb-2">Password</label>
      <input
        type="password"
        value={formData.password}
        onChange={(e) => setFormData({ ...formData, password: e.target.value })}

```

```

    className="w-full px-3 py-2 border rounded-md focus:outline-none focus:ring-2
focus:ring-blue-500"
    required
  />
  {errors.password} && <p className="text-red-500 text-sm mt-1">{errors.password}</p>
</div>

<div className="mb-6">
<label className="block text-sm font-medium mb-2">Confirm Password</label>
<input
  type="password"
  value={formData.confirmPassword}
  onChange={(e) => setFormData({ ...formData, confirmPassword: e.target.value })}
  className="w-full px-3 py-2 border rounded-md focus:outline-none focus:ring-2
focus:ring-blue-500"
  required
  />
  {errors.confirmPassword} && <p className="text-red-500 text-sm
mt-1">{errors.confirmPassword}</p>
</div>

<button
  type="submit"
  disabled={loading}
  className="w-full bg-blue-600 text-white py-2 px-4 rounded-md hover:bg-blue-700
disabled:opacity-50"
  >
  {loading ? 'Creating Account...' : 'Create Account'}
</button>
</form>
);
};

```

**Và tiếp tục implement các tasks khác...**

**Sau khi hoàn thành:**

```
*run-tests # Chạy all tests
```

**Anh Tuấn cập nhật story:**

```

## Tasks
- [x] Setup project structure với React + TypeScript
- [x] Create registration form component
- [x] Implement client-side validation
- [x] Setup backend API endpoint POST /auth/register
- [x] Implement password hashing với bcrypt
- [x] Setup email service với SendGrid
- [x] Create email verification system

```

- [x] Add error handling và loading states
- [x] Write unit tests cho form validation
- [x] Write integration tests cho registration flow

## Dev Agent Record

### File List

- src/components/auth/RegistrationForm.tsx (new)
- src/pages/RegisterPage.tsx (new)
- backend/routes/auth.ts (new)
- backend/services/emailService.ts (new)
- backend/models/User.ts (new)
- tests/auth/registration.test.tsx (new)

### Status: Ready for Review

### QA Review:

/qa  
\*review docs/stories/story-1-1.md

### Anh Quang review và refactor:

#### Code Quality Assessment:

"Implementation tốt, form validation comprehensive. Có một số điểm cần optimize để maintainable hơn."

#### Refactoring Performed:



- **File:** src/components/auth/RegistrationForm.tsx
- **Change:** Extract validation logic thành custom hook useFormValidation
- **Why:** Separation of concerns, reusable across forms
- **How:** Move validation logic ra khỏi component, easier testing
- **File:** backend/routes/auth.ts
- **Change:** Add request rate limiting
- **Why:** Prevent spam registration attacks
- **How:** Use express-rate-limit middleware

**Final Status: Approved - Ready for Done**



### Story 1.2: Login System

Tiếp tục với login system theo cùng quy trình...

#### Sprint Summary sau 1 tuần:

-  Story 1.1: User Registration - DONE
-  Story 1.2: Login/Logout - DONE



-  Story 1.3: User Profile - DONE
-  Story 1.4: Password Reset - DONE

**Epic 1 hoàn thành! Có foundation vững chắc để build các tính năng tiếp theo.**

## **Sprint 2: Personal Todo Management (Tuần 2)**

**Stories trong sprint này:**

- Story 2.1: Todo CRUD Operations
- Story 2.2: Todo Priorities & Due Dates
- Story 2.3: Todo Categories & Filtering
- Story 2.4: Todo Search & Sorting

**Quy trình tương tự:**

```
/sm → *draft (tạo stories)
/po → *validate-story-draft (kiểm tra)
/dev → *develop-story (implement)
/qa → *review (quality check)
```

## **Sprint 3: Team Collaboration (Tuần 3)**

**Stories:**

- Story 3.1: Team Board Creation
- Story 3.2: Member Invitation System
- Story 3.3: Shared Todo Management
- Story 3.4: Role-based Permissions

## **Sprint 4: Real-time Features (Tuần 4)**

**Stories:**

- Story 4.1: [Socket.io](https://socket.io) Integration
- Story 4.2: Live Todo Updates
- Story 4.3: Real-time Notifications
- Story 4.4: Online/Offline Status

## **Sprint 5: Trello Integration (Tuần 5)**

**Stories:**

- Story 5.1: Trello OAuth Connection
- Story 5.2: Import Trello Boards
- Story 5.3: Bi-directional Sync
- Story 5.4: Sync Conflict Resolution

## Sprint 6: Polish & Deploy (Tuần 6)

### Stories:

- Story 6.1: Performance Optimization
- Story 6.2: Error Handling & Monitoring
- Story 6.3: Production Deployment
- Story 6.4: User Testing & Feedback

## Production Deployment

### Frontend Deploy (Vercel):

```
# Connect GitHub repo to Vercel
# Auto-deploy trên every push to main

# Environment variables:
REACT_APP_API_URL=https://todoflow-backend.railway.app
REACT_APP_SOCKET_URL=https://todoflow-backend.railway.app
```

### Backend Deploy (Railway):

```
# Connect GitHub repo to Railway
# Auto-deploy backend service

# Environment variables:
DATABASE_URL=postgresql://...neon.tech
JWT_SECRET=...
SENDGRID_API_KEY=...
TRELLO_CLIENT_ID=...
TRELLO_CLIENT_SECRET=...
```

### Database (Neon PostgreSQL):

```
# Provision PostgreSQL database
# Connection string: postgresql://...neon.tech
# Run migrations: npx prisma migrate deploy
```

## Kết quả cuối cùng

### Sau 6 tuần, bạn sẽ có:

#### Live Application:

- URL: <https://todoflow.vercel.app>
- Fully functional todo app với team collaboration
- Trello integration hoạt động

- Mobile responsive
- Real-time updates

#### Technical Stack:

- React + TypeScript frontend
- Node.js + Express backend
- PostgreSQL database
- [Socket.io](https://socket.io/) realtime
- Trello API integration
- Production-ready deployment

#### Features Implemented:

- ☒ User authentication với email verification
- ☒ Personal todo management với priorities
- ☒ Team boards với member invitations
- ☒ Real-time collaboration
- ☒ Trello import/sync
- ☒ Mobile responsive design
- ☒ Error monitoring với Sentry

#### Documentation:

- Complete PRD với 50+ pages
- System architecture documentation
- API documentation
- User stories với acceptance criteria
- Test coverage reports
- Deployment guides

### Lessons Learned từ dự án này

#### BMad Method Benefits thực tế:

1. **Clear Direction:** Không bao giờ bị stuck "làm gì tiếp theo"
2. **Quality Assurance:** Mọi code đều được review trước khi merge
3. **Documentation:** Tự động có complete docs cho maintenance
4. **Context Persistence:** Có thể dừng/tiếp tục bất kỳ lúc nào
5. **Team Ready:** Structure sẵn sàng cho team collaboration

#### Time Savings:

- Planning phase: 6 tiếng → Save 2-3 ngày debugging later
- Development: Structured approach → 30% faster than ad-hoc

- Maintenance: Complete docs → 50% faster onboarding

#### **Code Quality:**

- Consistent patterns across codebase
- Comprehensive test coverage
- Senior-level refactoring từ QA
- Production-ready architecture

### **Scaling Up sau MVP**

#### **Phase 2 Features (nếu muốn mở rộng):**

- Advanced reporting & analytics
- Third-party integrations (Slack, Asana)
- Mobile native apps
- Advanced team permissions
- Custom workflows
- API for developers

#### **BMad Method sẽ support scaling:**

```
/analyst → *brainstorm "advanced reporting features"  
/pm → *create-brownfield-epic  
/sm → *draft (new stories)  
# Repeat process với established codebase
```

### **Total Investment vs Returns**

#### **Time Investment:**

- Planning: 6 tiếng
- Development: 80-100 tiếng (part-time over 6 tuần)
- Total: ~106 tiếng

#### **Returns:**

- Production-ready application
- Complete documentation suite
- Scalable architecture
- Team-ready processes
- Maintainable codebase
- Real user value

**ROI:** Cách làm BMad Method cho quality cao hơn 50% so với ad-hoc development, đồng thời save 30% debugging/refactoring time long-term.

---

**Chúc mừng! Bạn đã hoàn thành một dự án thực tế với BMad Method từ ý tưởng đến production deployment!**