Documentación del Proyecto Claude Code

1 Documentación del Proyecto: Claude Code - Presentación de Capacitación

Repositorio: https://github.com/pcarvajalr/claude-code-presentacion.git **Fecha:** Octubre 2025 **Autor:** Pedro Carvajal

---

1.1 1. Introducción y Visión General

Este proyecto es un **portal educativo completo y autónomo** diseñado para capacitar a equipos de desarrollo en el uso de Claude Code, el IDE oficial de Anthropic. El proyecto combina teoría, ejemplos prácticos y herramientas automatizadas para mejorar la productividad del desarrollo de software.

1.1.1 Objetivos Principales

- Proporcionar un plan de capacitación estructurado de 2-4 semanas sobre Claude Code
- Enseñar las capacidades avanzadas de automatización, plugins y agentes especializados
- Aplicar conocimientos en proyectos reales de Laravel y .NET
- Desarrollar habilidades en MCPs (Model Context Protocol), hooks y comandos personalizados
- Crear agentes y skills específicos para las necesidades del equipo

1.1.2 Público Objetivo

Equipo de desarrollo de software especializado en: - Laravel (PHP) - .NET Core (C#) - Desarrollo de aplicaciones web modernas - Integración de APIs y servicios externos

---

1.2 2. Estructura del Proyecto

```
claude-code/
├── .claude/                          # Configuración de Claude Code
│   ├── hooks/                        # Scripts de automatización
│   │   ├── EVENTOS-HOOKS.md         # Documentación completa de hooks
│   │   └── HOOKS-RESUMEN.md         # Resumen rápido de hooks
│   ├── skills/                       # Habilidades personalizadas
│   │   ├── pdf-generator/           # Skill para generación de PDFs
│   │   │   ├── scripts/             # Scripts de conversión
│   │   │   ├── assets/              # Plantillas LaTeX y CSS
│   │   │   ├── references/          # Documentación técnica
│   │   │   └── SKILL.md             # Documentación del skill
│   │   └── skill-creator/           # Herramienta para crear skills
│   ├── settings.json                 # Configuración global
│   └── settings.local.json           # Configuración local
├── .git/                             # Repositorio Git
├── .gitignore                        # Configuración de Git
├── .vercel/                          # Configuración de Vercel
├── CLAUDE.md                         # Instrucciones del proyecto
├── Entregable.md                     # Documento de entregables
├── index.html                        # Página principal (52 KB)
├── modulo-1-fundamentos.html         # Módulo 1: Fundamentos (80 KB)
├── modulo-2-hooks-comandos.html      # Módulo 2: Hooks y Comandos (67 KB)
├── modulo-3-plugins-agentes.html     # Módulo 3: Plugins y Agentes (131 KB)
├── presentacion-claude-code.html     # Presentación general (52 KB)
├── plan-capacitacion-claude-code.md  # Plan de capacitación (4.8 KB)
└── vercel.json                       # Configuración de despliegue
```

1.2.1 Archivos Principales

Archivo	Tamaño	Descripción
`index.html`	52 KB	Página principal con módulos en acordeón
`modulo-1-fundamentos.html`	80 KB	Fundamentos y configuración de Claude Code
`modulo-2-hooks-comandos.html`	67 KB	Automatización con hooks y comandos
`modulo-3-plugins-agentes.html`	131 KB	Plugins, agentes especializados y skills
`presentacion-claude-code.html`	52 KB	Presentación general del curso

Archivo	Tamaño	Descripción
`plan-capacitacion-claude-code.md`	4.8 KB	Plan estratégico del curso

1.3 3. Tecnologías Utilizadas

1.3.1 Frontend

- **HTML5**: Semántica moderna, accesibilidad
- **CSS3**:
  - Animaciones y transiciones
  - Gradientes y efectos visuales
  - Flexbox y Grid para layouts responsivos
  - Variables CSS para tematización
- **JavaScript Vanilla**:
  - Interactividad de menús y navegación
  - Smooth scroll entre secciones
  - Animaciones dinámicas
  - Manejo de acordeones y modales

1.3.2 Despliegue y Hosting

- **Vercel**: Hosting de la aplicación web
- **Git/GitHub**: Control de versiones
- **Node.js**: Scripts de conversión y automatización
- **PowerShell**: Scripts de batch processing

1.3.3 Herramientas de Generación de Contenido

- **Pandoc**: Conversión de markdown a PDF con LaTeX
- **wkhtmltopdf**: Conversión de HTML estático a PDF
- **Playwright**: Conversión de presentaciones dinámicas a PDF
- **LaTeX**: Generación de documentos profesionales

1.3.4 Características de Diseño

- **Responsive Design**: Adaptable a móviles, tablets y escritorio
- **Mobile-First**: Diseño optimizado primero para dispositivos móviles
- **Accesibilidad**: Navegación por teclado, semántica HTML
- **Optimización**: Carga rápida, assets optimizados

1.4 4. Módulos del Curso

El curso está estructurado en **5 módulos teóricos** que cubren todos los aspectos de Claude Code.

1.4.1 Módulo 1: Fundamentos y Configuración

Duración: 3-4 sesiones **Archivo:** `modulo-1-fundamentos.html`

Contenido: - Introducción a Claude Code y sus capacidades como IDE - Configuración inicial del entorno de trabajo - Comandos básicos integrados: - `/release-notes`: Ver novedades de la plataforma - `/plugins`: Explorar marketplace de plugins - Comandos de navegación y ayuda - Interfaz y flujo de trabajo básico - Mejores prácticas para comenzar - Configuración de archivos CLAUDE.md para instrucciones personalizadas

Objetivos de Aprendizaje: - Comprender qué es Claude Code y cómo puede mejorar la productividad - Configurar correctamente el entorno de desarrollo - Dominar los comandos básicos esenciales - Establecer buenas prácticas desde el inicio

1.4.2 Módulo 2: Automatización con Hooks y Comandos

Duración: 3-4 sesiones **Archivo:** `modulo-2-hooks-comandos.html`

Contenido: - **Sistema de Hooks**: 9 tipos de eventos automatizables - SessionStart: Validación de entorno al iniciar - UserPromptSubmit: Validación de prompts del usuario - PreToolUse: Ejecución de linters antes de cambios - PostToolUse: Formateo automático de código - Notification: Sistema de logging centralizado - Stop: Validación de calidad de respuestas - SubagentStop: Validación de outputs de agentes - PreCompact: Backup antes de compactar contexto - SessionEnd: Generación de reportes y limpieza

- **Slash Commands Personalizados**:
  - Creación de comandos custom (.md en .claude/commands/)
  - Ejemplos: `/review-laravel`, `/generate-api`, `/deploy-staging`
  - Reutilización de flujos comunes
- **Documentación y Configuración**:
  - Archivos CLAUDE.md para instrucciones globales y de proyecto
  - settings.json para configuración de Claude Code

Casos de Uso Prácticos: - Validación automática pre-commit - Aplicación de estándares de código - Detección de secretos en código - Reportes automáticos de sesiones

1.4.3 Módulo 3: Plugins, Agentes Especializados y Skills

Duración: 4-5 sesiones **Archivo:** `modulo-3-plugins-agentes.html`

Contenido:

A. Marketplace de Plugins: - Exploración del marketplace (`/plugins`) - Instalación y configuración de plugins oficiales - Plugins destacados: - **code-reviewer**: Revisión automática de código - **enforcing-standards**: Aplicación de estándares de código - **pr-reviewer**: Análisis de pull requests - **test-generator**: Generación automática de tests

B. Agentes Especializados: - Concepto de agentes: IA especializada en frameworks específicos - Creación de agentes personalizados para: - Laravel: Mejores prácticas, Eloquent, Blade - .NET Core: Clean Architecture, Entity Framework, LINQ - React/Vue: Componentes, hooks, estado - Configuración de agentes en settings.json - Invocación y uso de agentes en proyectos

C. Skills (Habilidades para Archivos Específicos): - Concepto de skills: Capacidades para tipos de archivo - Skills incluidos: - **pdf-generator**: Conversión de MD/HTML a PDF - **skill-creator**: Creación de nuevos skills - Creación de skills personalizados - Estructura de un skill (.claude/skills/)

Ejemplos Prácticos: - Configurar agente Laravel para proyecto PWA - Crear skill para generación de documentación - Instalar y usar plugin code-reviewer en pipeline

1.4.4 Módulo 4: MCPs (Model Context Protocol) e Integraciones

Duración: 2-3 sesiones

Contenido: - **¿Qué son los MCPs?** - Protocolo para extender capacidades de Claude Code - Conexión con servicios externos - Acceso a APIs, bases de datos, herramientas de diseño

- **MCPs Principales:**
  - **Figma MCP**: Importar diseños y componentes
  - **GitHub MCP**: Integración profunda con repositorios
  - **Database MCPs**: Conexión directa a bases de datos
  - **API MCPs**: Consumo de APIs REST/GraphQL
- **Instalación de MCPs:**

  ```
  claude mcp add --transport http figma https://mcp.figma.com/mcp
  ```

- **Casos de Uso:**
  - Convertir diseños de Figma a código React/Vue
  - Consultar bases de datos para análisis de esquemas
  - Automatizar workflows con APIs externas

1.4.5 Módulo 5: Aplicación a Proyectos Reales

Duración: 4-5 sesiones

Contenido: - Integración de Claude Code en proyectos existentes - Resolución de desafíos técnicos reales del equipo - Optimización de código y flujos de trabajo - Configuración de agentes específicos para Laravel y .NET - Refactorización de código legacy - Generación automática de tests - Mejora de documentación de proyectos - Automatización de tareas repetitivas

Metodología: - Análisis de proyecto existente - Identificación de áreas de mejora - Aplicación de herramientas de Claude Code - Medición de impacto en productividad

---

1.5 5. Proyectos Prácticos

El curso incluye **3 proyectos prácticos** para aplicar los conocimientos adquiridos.

1.5.1 Proyecto 1: PWA de Chat Multiusuario (Laravel)

Duración: 4 sesiones **Nivel:** Intermedio-Avanzado

Tecnologías: - Laravel 10+ - Progressive Web App (PWA) - WebSockets (Laravel Broadcasting) - WhatsApp Business API - Figma para diseño UI/UX - MySQL/PostgreSQL

Características Principales: - Sistema de chat en tiempo real entre múltiples usuarios - Integración con WhatsApp Business API para recibir/enviar mensajes - Sincronización bidireccional: web ↔ WhatsApp - Sistema de webhooks para eventos de WhatsApp - Notificaciones push en PWA - Gestión de sesiones y presencia de usuarios - Almacenamiento offline (Service Workers) - Diseño responsivo mobile-first

Herramientas de Claude Code Utilizadas: - **MCP Figma**: Importar diseños y convertir a componentes Blade - **Agente Laravel**: Generar modelos, controladores, migraciones - **Hooks**: Pre-commit validation, code standards - **Plugin code-reviewer**: Revisión automática de código Laravel - **Slash Commands**: `/generate-migration`, `/create-controller`

Estructura del Proyecto:

```
pwa-chat/
├── app/
│   ├── Models/User.php, Message.php, Conversation.php
│   ├── Http/Controllers/ChatController.php, WhatsAppController.php
│   ├── Events/MessageSent.php, UserTyping.php
│   └── Services/WhatsAppService.php
├── database/migrations/
├── resources/
│   ├── views/chat/
│   └── js/chat-app.js
├── routes/channels.php
└── public/service-worker.js
```

Entregables: - Aplicación PWA funcional con chat en tiempo real - Integración completa con WhatsApp Business API - Webhooks configurados y funcionando - Documentación técnica del proyecto - Tests unitarios y de integración

1.5.2 Proyecto 2: Backend y Webhooks (.NET Core)

Duración: 4 sesiones **Nivel:** Intermedio

Tecnologías: - .NET Core 8.0 - API REST (ASP.NET Web API) - Entity Framework Core - JWT Authentication - SQL Server / PostgreSQL - Swagger/OpenAPI

Características Principales: - API RESTful con arquitectura limpia (Clean Architecture) - Sistema de autenticación y autorización con JWT - Gestión completa de usuarios (CRUD) - Sistema de webhooks para eventos del sistema - Validación de datos con FluentValidation - Logging estructurado (Serilog) - Rate limiting y throttling - Documentación automática con Swagger

Herramientas de Claude Code Utilizadas: - **Agente .NET**: Generar controladores, servicios, DTOs - **Slash Commands**: `/create-entity`, `/generate-tests` - **Skills**: Skill personalizado para generar documentación API - **Hooks**: PostToolUse para formateo automático (dotnet format)

Arquitectura Clean Architecture:

```
WebhooksAPI/
├── src/
│   ├── Domain/
│   │   ├── Entities/
│   │   └── Interfaces/
│   ├── Application/
│   │   ├── Services/
│   │   ├── DTOs/
│   │   └── Validators/
│   ├── Infrastructure/
│   │   ├── Persistence/
│   │   └── External/
│   └── WebAPI/
│       ├── Controllers/
│       ├── Middleware/
│       └── Program.cs
└── tests/
```

Endpoints Principales: - `POST /api/auth/register` - Registro de usuarios - `POST /api/auth/login` - Login con JWT - `GET /api/users` - Listar usuarios (autorizado) - `POST /api/webhooks` - Registrar webhook - `GET /api/webhooks/{id}/logs` - Ver logs de webhook

Entregables: - API REST funcional con Clean Architecture - Sistema de autenticación JWT implementado - CRUD completo de usuarios con validaciones - Sistema de webhooks operativo - Suite de tests unitarios y de integración - Documentación Swagger completa

1.5.3 Proyecto 3: Mejora de Proyecto Laravel Existente

Duración: 3-4 sesiones **Nivel:** Avanzado

Objetivos: - Aplicar Claude Code a un proyecto Laravel existente del equipo - Refactorizar código para mejorar calidad y mantenibilidad - Implementar mejores prácticas de Laravel - Generar tests automáticos para código legacy - Mejorar documentación del proyecto

Actividades:

1. Análisis del Proyecto (1 sesión) - Usar agente Laravel para analizar estructura del proyecto - Identificar code smells y áreas de mejora - Detectar código duplicado - Analizar complejidad ciclomática

2. Refactorización (2 sesiones) - Aplicar patrones de diseño (Repository, Service Layer) - Mejorar consultas Eloquent (N+1 queries) - Implementar Form Requests para validaciones - Crear DTOs para transferencia de datos - Refactorizar controladores gordos

3. Generación de Tests (1 sesión) - Generar tests unitarios con agente - Crear tests de feature para endpoints críticos - Configurar CI/CD con tests automáticos

4. Documentación (0.5 sesión) - Generar documentación de API - Documentar arquitectura del proyecto - Crear README comprehensivo

Herramientas Utilizadas: - **Agentes personalizados**: Agente específico para el proyecto - **Plugins del marketplace**: code-reviewer, pr-reviewer - **Hooks**: Pre-commit para tests, linting - **Skills**: pdf-generator para documentación

Métricas de Éxito: - Reducción de complejidad ciclomática en 30% - Cobertura de tests > 70% - Reducción de N+1 queries identificadas - Documentación completa generada

Entregables: - Código refactorizado con mejoras documentadas - Suite de tests con cobertura > 70% - Documentación técnica actualizada - Reporte de mejoras y métricas

---

1.6 6. Herramientas Clave de Claude Code

1.6.1 6.1 MCPs (Model Context Protocol)

Los MCPs son extensiones que permiten a Claude Code conectarse con servicios externos.

MCPs Disponibles: - **Figma MCP**: Importar diseños, componentes, estilos - **GitHub MCP**: Integración profunda con repositorios - **Database MCPs**: MySQL, PostgreSQL, MongoDB - **API MCPs**: Consumo de APIs REST/GraphQL

Instalación:

```
claude mcp add --transport http <nombre> <url>
```

1.6.2 6.2 Hooks (Automatización de Eventos)

Sistema de 9 hooks para automatizar tareas en diferentes momentos del flujo de trabajo.

Hook	Cuándo se Ejecuta	Casos de Uso
SessionStart	Al iniciar sesión	Validar entorno, cargar configuración
UserPromptSubmit	Antes de procesar prompt	Validar entrada, detectar secretos
PreToolUse	Antes de usar herramienta	Ejecutar linter, validar permisos
PostToolUse	Después de usar herramienta	Formatear código, ejecutar tests
Notification	Al recibir notificación	Logging, alertas
Stop	Al detener ejecución	Validar calidad, guardar estado
SubagentStop	Al terminar subagente	Validar output de agente
PreCompact	Antes de compactar contexto	Backup, guardar historial
SessionEnd	Al finalizar sesión	Generar reportes, limpiar

Ejemplo de Hook - PostToolUse:

```
{
  "hooks": {
    "PostToolUse": [
      {
        "command": "npm run lint -- --fix",
        "description": "Auto-fix linting issues",
        "patterns": ["*.js", "*.ts"]
      }
    ]
  }
}
```

1.6.3 6.3 Slash Commands Personalizados

Comandos reutilizables para tareas comunes.

Creación: 1. Crear archivo en `.claude/commands/nombre-comando.md` 2. Escribir el prompt del comando 3. Usar con `/nombre-comando`

Ejemplos: - `/review-laravel`: Revisar código Laravel con estándares - `/generate-api`: Generar endpoint REST completo - `/deploy-staging`: Desplegar a ambiente de staging

1.6.4 6.4 Plugins del Marketplace

Plugins Principales: - **code-reviewer**: Revisión automática de calidad de código - **enforcing-standards**: Aplicar estándares del equipo - **pr-reviewer**: Análisis de pull requests - **test-generator**: Generar tests automáticamente

Instalación:

```
/plugins
# Seleccionar plugin del marketplace
```

1.6.5 6.5 Agentes Especializados

IA especializada en frameworks y tecnologías específicas.

Configuración en settings.json:

```
{
  "agents": {
    "laravel-expert": {
      "description": "Experto en Laravel y Eloquent",
      "tools": ["Read", "Write", "Edit", "Bash"],
      "context": "Sigue las mejores prácticas de Laravel..."
    }
  }
}
```

Invocación:

```
@laravel-expert Optimiza esta consulta Eloquent
```

1.6.6 6.6 Skills (Capacidades Especializadas)

Habilidades para trabajar con tipos de archivo específicos.

Skills Incluidos: - **pdf-generator**: Convertir MD/HTML a PDF profesional - **skill-creator**: Asistente para crear nuevos skills

Estructura de un Skill:

```
.claude/skills/mi-skill/
├── SKILL.md          # Documentación y workflows
├── scripts/          # Scripts auxiliares
├── assets/           # Recursos (plantillas, estilos)
└── references/       # Documentación técnica
```

1.7 7. Sistema de Hooks Detallado

1.7.1 Arquitectura de Hooks

Los hooks son scripts que se ejecutan automáticamente en respuesta a eventos específicos durante una sesión de Claude Code.

1.7.2 Tipos de Hooks y Uso

1.7.2.1 1. SessionStart

Propósito: Validar el entorno y configuración al iniciar sesión

Ejemplo de Uso:

```
# .claude/hooks/session-start.sh
#!/bin/bash
# Verificar que Node.js está instalado
if ! command -v node &> /dev/null; then
    echo "ERROR: Node.js no está instalado"
    exit 1
fi

# Verificar versión de PHP
php --version | grep "PHP 8"
```

1.7.2.2 2. UserPromptSubmit

Propósito: Validar prompts del usuario antes de procesarlos

Ejemplo de Uso:

```
# Detectar secretos en el prompt
if echo "$PROMPT" | grep -E "(password|api_key|secret)" -i; then
    echo "ADVERTENCIA: Posible secreto detectado en el prompt"
fi
```

1.7.2.3 3. PreToolUse

Propósito: Ejecutar validaciones antes de que Claude use una herramienta

Ejemplo de Uso:

```
# Ejecutar linter antes de modificar código
if [ "$TOOL" = "Edit" ] || [ "$TOOL" = "Write" ]; then
    npm run lint
fi
```

1.7.2.4 4. PostToolUse

Propósito: Formatear o procesar código después de cambios

Ejemplo de Uso:

```
# Formatear automáticamente después de editar
if [ "$TOOL" = "Edit" ] || [ "$TOOL" = "Write" ]; then
    if [[ "$FILE" == *.php ]]; then
        ./vendor/bin/pint "$FILE"
    elif [[ "$FILE" == *.js ]]; then
        npx prettier --write "$FILE"
    fi
fi
```

1.7.2.5 5. Notification

Propósito: Logging centralizado de eventos

Ejemplo de Uso:

```
# Guardar log de notificaciones
echo "[$(date)] $MESSAGE" >> .claude/logs/notifications.log
```

1.7.2.6 6. Stop

Propósito: Validar calidad antes de detener ejecución

Ejemplo de Uso:

```
# Ejecutar tests antes de terminar
npm test
```

1.7.2.7 7. SubagentStop

Propósito: Validar output de agentes especializados

Ejemplo de Uso:

```
# Verificar que el agente generó tests
if [ "$AGENT" = "test-generator" ]; then
    if [ -z "$(find tests/ -name '*.php' -mmin -5)" ]; then
        echo "ADVERTENCIA: No se generaron archivos de test"
```

```
    fi
fi
```

1.7.2.8 8. PreCompact

Propósito: Backup antes de compactar contexto

Ejemplo de Uso:

```
# Guardar snapshot del contexto
cp -r .claude/context .claude/backups/context-$(date +%Y%m%d-%H%M%S)
```

1.7.2.9 9. SessionEnd

Propósito: Generar reportes y limpiar recursos

Ejemplo de Uso:

```
# Generar reporte de sesión
echo "Sesión finalizada: $(date)" >> .claude/reports/sessions.log
echo "Archivos modificados: $MODIFIED_FILES" >> .claude/reports/sessions.log

# Limpiar archivos temporales
rm -rf .claude/temp/*
```

1.7.3 Configuración de Hooks en settings.json

```
{
  "hooks": {
    "SessionStart": [
      {
        "command": "bash .claude/hooks/validate-env.sh",
        "description": "Validar entorno de desarrollo"
      }
    ],
    "PostToolUse": [
      {
        "command": "npm run format",
        "description": "Formatear código automáticamente",
        "patterns": ["*.js", "*.ts", "*.jsx", "*.tsx"]
      },
      {
        "command": "./vendor/bin/pint",
        "description": "Formatear código PHP",
```

```
        "patterns": ["*.php"]
      }
    ],
    "PreToolUse": [
      {
        "command": "npm run lint",
        "description": "Validar código con linter",
        "patterns": ["*.js", "*.ts"]
      }
    ],
    "SessionEnd": [
      {
        "command": "bash .claude/hooks/generate-report.sh",
        "description": "Generar reporte de sesión"
      }
    ]
  }
}
```

1.8 8. Configuración y Despliegue

1.8.1 8.1 Configuración Local

Archivos de Configuración:

CLAUDE.md (Instrucciones del Proyecto):

```
- al crear una migración, analizar sistema de migraciones para mantener coherencia y secuencia.
```

settings.json (Configuración de Claude Code):

```
{
  "model": "sonnet",
  "hooks": { /* configuración de hooks */ },
  "agents": { /* definición de agentes */ },
  "preferences": {
    "autoSave": true,
    "formatOnSave": true
  }
}
```

1.8.2 8.2 Despliegue en Vercel

vercel.json:

```
{
  "buildCommand": null,
  "outputDirectory": ".",
  "routes": [
    { "src": "/", "dest": "/index.html" },
    { "src": "/modulo-1", "dest": "/modulo-1-fundamentos.html" },
    { "src": "/modulo-2", "dest": "/modulo-2-hooks-comandos.html" },
    { "src": "/modulo-3", "dest": "/modulo-3-plugins-agentes.html" }
  ]
}
```

Comandos de Despliegue:

```
# Instalar Vercel CLI
npm install -g vercel

# Desplegar a producción
vercel --prod

# Preview deployment
vercel
```

1.8.3 8.3 Control de Versiones

Git Workflow:

```
# Clonar repositorio
git clone https://github.com/pcarvajalr/claude-code-presentacion.git

# Crear rama para nuevas características
git checkout -b feature/nueva-caracteristica

# Commit y push
git add .
git commit -m "feat: agregar nueva característica"
git push origin feature/nueva-caracteristica
```

Commits Recientes: - `14adc2f` - Eliminar elemento “Timeline” del menú de navegación y añadir .vercel a .gitignore - `9a6194a` - Initial commit: Presentación de capacitación Claude Code

---

1.9 9. Instrucciones de Uso

1.9.1 9.1 Para Instructores

Preparación del Curso: 1. Clonar el repositorio del proyecto 2. Revisar el contenido de cada módulo HTML 3. Preparar ejemplos prácticos adicionales según necesidades del equipo 4. Configurar acceso a MCPs necesarios (Figma, GitHub) 5. Preparar entornos de desarrollo para proyectos prácticos

Impartir las Sesiones: 1. Seguir el orden de los módulos (1 → 2 → 3 → 4 → 5) 2. Dedicar 50% del tiempo a teoría/demostración, 50% a práctica 3. Usar la página web como material de presentación 4. Intercalar proyectos prácticos entre módulos teóricos 5. Dedicar tiempo a resolver desafíos reales del equipo

Recursos Disponibles: - Presentación web interactiva en cada módulo HTML - Documentación de hooks en `.claude/hooks/` - Skills pre-configurados en `.claude/skills/` - Plan detallado en `plan-capacitacion-claude-code.md`

1.9.2 9.2 Para Participantes

Antes del Curso: 1. Instalar Claude Code: https://claude.com/claude-code 2. Configurar cuenta de Anthropic 3. Clonar el repositorio del proyecto 4. Revisar la presentación general: `presentacion-claude-code.html`

Durante el Curso: 1. Seguir las presentaciones en los archivos HTML 2. Practicar con los ejercicios propuestos 3. Implementar los proyectos prácticos 4. Crear agentes y hooks personalizados 5. Aplicar aprendizajes a proyectos actuales

Después del Curso: 1. Integrar Claude Code en workflow diario 2. Compartir agentes y hooks con el equipo 3. Continuar explorando plugins del marketplace 4. Documentar casos de uso específicos

1.9.3 9.3 Acceso a la Presentación Web

Localmente: 1. Abrir `index.html` en un navegador web 2. Navegar por los módulos usando el menú 3. Expandir/contraer secciones con los acordeones

Online (Vercel): - URL: (configurar después del despliegue) - Acceso directo a cada módulo mediante rutas

1.9.4 9.4 Generación de PDFs

Usar el skill pdf-generator para crear versiones PDF de la documentación:

```
# Convertir documentación a PDF
pandoc plan-capacitacion-claude-code.md -o plan-capacitacion.pdf \
  --toc \
  --number-sections \
  --variable geometry:margin=1in

# Convertir módulos HTML a PDF
node .claude/skills/pdf-generator/scripts/html-to-pdf.js \
  modulo-1-fundamentos.html \
  modulo-1-fundamentos.pdf
```

1.10 10. Entregables del Curso

1.10.1 10.1 Entregables del Instructor

1. **Aplicación Base con Spec Kit de GitHub**
   - Template de proyecto con Claude Code pre-configurado
   - Estructura de directorios estándar
   - Hooks y comandos básicos incluidos
2. **Set de Agentes Personalizados**
   - Agente Laravel especializado
   - Agente .NET Core
   - Agentes para tareas comunes del equipo
3. **PWA Funcional (Proyecto 1)**
   - Aplicación de chat en tiempo real
   - Integración con WhatsApp Business API
   - Código documentado y tests incluidos
4. **Backend .NET Core (Proyecto 2)**
   - API REST con Clean Architecture
   - Sistema de autenticación JWT
   - Webhooks funcionales
5. **Documentación Completa**
   - Guías de hooks y comandos
   - Referencias de MCPs
   - Tutoriales de plugins
6. **Mejoras en Proyecto Laravel Existente (Proyecto 3)**
   - Código refactorizado
   - Tests generados automáticamente
   - Documentación actualizada

1.10.2 10.2 Entregables de los Participantes

1. **Configuración Personal Optimizada**
   - settings.json personalizado
   - Hooks adaptados a su workflow
   - Slash commands para tareas repetitivas
2. **Agentes y Hooks Personalizados**
   - Al menos 2 agentes específicos para sus proyectos
   - Al menos 3 hooks implementados
   - Documentación de su configuración

3. **Conocimientos Aplicados**
    - Integración de Claude Code en proyecto actual
    - Mejoras documentadas en productividad
    - Casos de uso específicos resueltos
4. **Portafolio de Mejores Prácticas**
    - Colección de comandos útiles
    - Biblioteca de prompts efectivos
    - Patterns de uso de agentes

1.11 11. Recursos Adicionales

1.11.1 11.1 Documentación Oficial

- **Claude Code Docs**: https://docs.claude.com/claude-code
- **Anthropic API**: https://docs.anthropic.com
- **Model Context Protocol**: https://modelcontextprotocol.io

1.11.2 11.2 Herramientas Complementarias

- **Pandoc**: https://pandoc.org
- **Playwright**: https://playwright.dev
- **Vercel**: https://vercel.com
- **Laravel**: https://laravel.com
- **.NET**: https://dotnet.microsoft.com

1.11.3 11.3 Comunidad

- **GitHub Issues**: https://github.com/pcarvajalr/claude-code-presentacion/issues
- **Discussions**: Sección de discusiones en GitHub
- **Slack/Teams**: Canal del equipo para soporte

1.12 12. Roadmap y Próximos Pasos

1.12.1 Fase 1: Capacitación Inicial (Semanas 1-2)

- Completar Módulos 1-3
- Iniciar Proyecto 1 (PWA Laravel)

1.12.2 Fase 2: Proyectos Prácticos (Semanas 2-3)

- Completar Proyectos 1 y 2
- Módulos 4-5

1.12.3 Fase 3: Aplicación Real (Semana 4)

- Proyecto 3: Mejora de proyecto existente
- Integración en workflow del equipo

1.12.4 Fase 4: Post-Capacitación (Semana 5+)

- Soporte continuo
- Sharing de agentes y hooks
- Casos de uso avanzados

1.13 13. Mejores Prácticas Recomendadas

1.13.1 13.1 Uso de Claude Code

1. **Comenzar Simple**
   - No sobre-ingenierizar soluciones
   - Empezar con comandos básicos
   - Agregar hooks gradualmente
2. **Documentar Todo**
   - Usar CLAUDE.md para instrucciones del proyecto
   - Documentar agentes y hooks creados
   - Mantener README actualizado
3. **Reutilizar Configuraciones**
   - Compartir agentes exitosos con el equipo
   - Crear biblioteca de slash commands
   - Versionar configuraciones en Git
4. **Iterar y Mejorar**
   - Refinar prompts basándose en resultados
   - Ajustar agentes según necesidades
   - Actualizar hooks con nuevos casos de uso

1.13.2 13.2 Seguridad

1. **Nunca Commitear Secretos**
   - Usar hooks para detectar API keys
   - Configurar .gitignore apropiadamente
   - Usar variables de entorno
2. **Validar Entradas**
   - Usar hooks UserPromptSubmit
   - Sanitizar datos en webhooks
   - Validar configuraciones
3. **Revisar Código Generado**
   - No confiar ciegamente en código generado
   - Usar plugins code-reviewer
   - Ejecutar tests antes de deployar

1.14 14. Conclusión

Este proyecto proporciona un **plan de capacitación completo y estructurado** para dominar Claude Code y aplicarlo efectivamente en proyectos de desarrollo de software. La combinación de módulos teóricos, proyectos prácticos y herramientas automatizadas asegura que los participantes no solo aprendan las capacidades de Claude Code, sino que las apliquen directamente a sus proyectos reales.

Beneficios Esperados: - ⚡ **Incremento en Productividad**: 30-50% reducción en tiempo de desarrollo - 🎯 **Mejora en Calidad**: Código más consistente y bien documentado - 🤖 **Automatización**: Tareas repetitivas automatizadas con hooks - 📚 **Documentación**: Generación automática de docs profesionales - 🧪 **Testing**: Generación automática de tests unitarios

Siguientes Pasos: 1. Revisar el contenido de cada módulo HTML 2. Configurar el entorno de desarrollo 3. Comenzar con el Módulo 1: Fundamentos 4. Practicar con los proyectos incluidos 5. Integrar Claude Code en proyectos reales

---

Fecha de Última Actualización: Octubre 2025 **Versión:** 1.0 **Autor:** Pedro Carvajal **Repositorio:** https://github.com/pcarvajalr/claude-code-presentacion.git
Licencia: (Especificar licencia del proyecto)

Documento generado con Claude Code usando el skill pdf-generator