AI-Assisted Development System Prompts

Complete Package Summary

Project: Next-Generation Hosting Control Panel
Version: 1.0
Date: November 2, 2025
Audience: Development Team, Claude AI, GitHub Copilot

PACKAGE CONTENTS

This comprehensive system prompt package contains everything needed for senior-level, enterprise-grade AI-assisted development.

Document 1: System Prompts (Main Reference)

File: `ai-dev-system-prompts.md`
Length: 25+ pages
Purpose: Comprehensive standards for all development aspects

15 Sections:

1. Primary Development Principles
2. Technology Stack Requirements
3. Security Standards & Requirements
4. Code Quality Standards
5. Architecture & Design Patterns
6. Development Workflow & Git Practices
7. Testing & Quality Assurance
8. Deployment & DevOps Standards
9. API Development Standards
10. Database Standards
11. Frontend Development Standards
12. Infrastructure & SysAdmin Standards
13. Documentation Requirements
14. Code Review Checklist
15. Emergency Procedures

󰀀 Document 2: Prompting Guide (How-To)

File: `ai-prompting-guide.md`
Length: 20+ pages
Purpose: Instructions for using the system prompts effectively

Contents:

- Quick start for Claude
- Quick start for GitHub Copilot
- Prompt engineering techniques
- Workflow integration strategies
- Task-specific prompt templates
- Claude-specific tips and tricks
- GitHub Copilot keyboard shortcuts
- Validation checklists
- Common mistakes to avoid
- End-to-end workflow example

󰀀 Document 3: Quick Reference (Cheat Sheet)

File: `quick-reference-card.md`
Length: 4 pages
Purpose: Desk reference, printable and laminated

Quick Access To:

- Prompting templates
- Tech stack checklist
- Security checklist
- Code quality checklist
- Testing pyramid
- Error handling patterns
- Authentication patterns
- API response formats
- Git workflow
- Deployment checklist
- Keyboard shortcuts
- Common gotchas

📋 Document 4: Navigation Index

File: `documentation-index.md`
Length: 5+ pages
Purpose: Guide for finding information and integrating into workflow

Includes:

- Quick start instructions
- Document usage guide (when to reference each section)
- Workflow integration steps
- Reference by technology
- Common scenarios with solutions
- Team setup instructions
- Troubleshooting guide
- Maintenance schedule

QUICK START (CHOOSE YOUR PATH)

Path 1: Solo Developer (You Alone)

1. Save these 4 documents locally
2. Read main document (2-3 hours)
3. Print quick-reference-card
4. Keep at desk while coding
5. Paste relevant sections to Claude for help
6. Use GitHub Copilot inline chat (Ctrl+I)
7. Review generated code against standards

Time to productive: Same day

Path 2: Small Team (2-5 Developers)

1. Add documents to project repository:
    - `.copilot/context.md` (main prompts)
    - `docs/development-standards.md` (for reference)
    - `docs/quick-reference.md` (cheat sheet)
2. Team onboarding:
    - Day 1: Read main document
    - Day 2: Review quick-reference
    - Day 3: First feature with pair programming

- Day 4+: Solo development with code review

3. Code review process:
    - Section 14 used as review checklist
    - Claude reviews code before human review
    - GitHub Copilot generates tests

Time to productive: 1-2 weeks per developer

Path 3: Larger Team (5+ Developers)

1. Add to project repository and Wiki
2. Quarterly review meetings
3. Monthly training sessions
4. Integrate into CI/CD pipeline:
    - Linting checks
    - Security scanning
    - Test coverage gates
    - Code quality metrics
5. Developer onboarding:
    - Week 1: Read documentation
    - Week 2: Pair programming with senior dev
    - Week 3: First PR with senior review
    - Week 4+: Solo development

Time to productive: 4 weeks per developer

TECHNOLOGY STACK CONFIGURED

Backend (Production Ready)

```
✓ Rust 1.75+ (latest stable)
✓ Actix-web 4.x (high-performance)
✓ Tokio async runtime
✓ PostgreSQL 14+ (primary database)
✓ Redis 7.x (caching/sessions)
✓ RabbitMQ 3.12+ (message queue)
✓ sqlx/Diesel (ORM/queries)
```

Frontend (Production Ready)

```
✓ React 18.x (latest)
✓ TypeScript 5.x (strict mode required)
✓ Vite 5.x (build tool)
✓ Redux Toolkit 1.9.x (state management)
✓ Material-UI 5.x + TailwindCSS 3.x (styling)
✓ React Router 6.x (navigation)
✓ Jest + React Testing Library (testing)
```

DevOps & Infrastructure

```
✓ Docker 24.x (containerization)
✓ Kubernetes 1.28.x (orchestration)
✓ GitHub Actions (CI/CD)
✓ Prometheus + Grafana (monitoring)
✓ Terraform 1.6.x (infrastructure-as-code)
✓ PostgreSQL backups (automated)
✓ ELK Stack / Grafana Loki (logging)
```

Security Stack

```
✓ HashiCorp Vault (secrets management)
✓ Let's Encrypt (SSL/TLS)
✓ ModSecurity 3.x (WAF)
✓ OWASP CRS (attack prevention)
✓ Semgrep / CodeQL (SAST)
✓ Snyk / Dependabot (dependency scanning)
✓ Argon2 (password hashing)
✓ JWT (authentication tokens)
```

SECURITY FEATURES INCLUDED

Application Security

- Input validation on all endpoints
- SQL injection prevention (parameterized queries)
- XSS prevention (output encoding)
- CSRF protection (tokens + SameSite)
- Rate limiting (IP and user-based)
- Password hashing (Argon2)
- Two-factor authentication (TOTP)
- Session management (Redis-based)

Data Security

- Encryption at-rest (AES-256-GCM)
- Encryption in-transit (TLS 1.3)
- Secrets management (Vault)
- Data classification (CRITICAL/HIGH/MEDIUM/LOW)
- GDPR compliance (right to be forgotten)
- PCI-DSS compliance (payment data)
- HIPAA compliance (health data)

Infrastructure Security

- Firewall configuration
- SSH hardening
- Automatic security updates
- Intrusion detection
- DDoS protection
- Bot detection
- Geo-blocking capability

Operational Security

- Audit logging
- Security monitoring
- Alert rules
- On-call procedures
- Disaster recovery plan
- Backup strategy
- Incident response procedures

CODE QUALITY STANDARDS

Test Requirements

- Minimum 80% code coverage overall
- 100% coverage for critical paths
- Unit tests (70% of test pyramid)
- Integration tests (20% of test pyramid)
- End-to-end tests (10% of test pyramid)

- Performance tests (load test at 10x capacity)

Performance Targets

- API latency: < 100ms (95th percentile)
- Database query: < 50ms (average)
- Page load time: < 200ms
- Throughput: 10,000+ requests/second per server
- Error rate: < 0.1%
- Uptime: 99.9% (43 minutes downtime per month)

Code Review Standards

- Every PR requires code review
- Security review before merge
- Performance impact assessment
- Database migration safety check
- Backward compatibility verification
- Documentation completeness

Documentation Requirements

- Every function documented
- Every API endpoint documented
- Architecture decision records (ADRs)
- Runbooks for operations
- API specification (OpenAPI 3.1)
- README with setup instructions

PROMPT TEMPLATES INCLUDED

1. Implementation Template

```
[Paste relevant section from system prompts]

Implement [feature]:
- Accept: [inputs]
- Return: [outputs]
- Must handle: [error cases]
- Security: [specific concerns]

Include: Tests, docs, error handling, examples
```

2. Security Audit Template

```
[Paste section 3 - Security Standards]

Audit this code for:
- OWASP Top 10 vulnerabilities
- Authentication/authorization issues
- Data protection problems
- Error information leakage
- Cryptographic correctness

Code: [paste code]
```

3. Performance Optimization Template

```
[Paste section 8 - Performance]

Optimize for:
- Latency target: [milliseconds]
- Throughput target: [requests/sec]
- Resource constraints: [CPU/RAM/disk]

Current approach: [describe]

Provide analysis + optimized implementation
```

4. Code Review Template

```
[Paste section 14 - Code Review Checklist]

Review this code for:
✓ Security ✓ Performance ✓ Testing ✓ Quality

Code: [paste code]

Issues found &amp; fixes needed?
```

5. Architecture Design Template

```
[Paste section 5 - Architecture &amp; Design Patterns]

Design [system/component]:
- Requirements: [list]
- Constraints: [list]
- Scale target: [capacity]

Provide:
1. Architecture diagram (ASCII)
2. Component design
```

```
3. Interaction patterns
4. Technology choices &amp; justification
```

INTEGRATION WITH YOUR WORKFLOW

With GitHub Copilot

1. Create `.copilot/context.md` in project root
2. Paste `ai-dev-system-prompts.md` content
3. Use in VSCode:
    - `Ctrl+I` for inline chat
    - `Cmd+I` on Mac
    - Reference: `@workspace`

Example:

```
// @workspace Implement authentication
// following standards in .copilot/context.md
// Use Argon2, JWT, rate limiting
```

With Claude

1. Copy relevant sections from main document
2. Paste into Claude conversation
3. Ask specific question about your feature
4. Claude provides guidance

Example:

```
[Paste sections 2, 3, 5, 9]

Design and implement user authentication endpoint
that follows all standards above.
```

With Your Team

1. Add to project repository
2. Reference in code reviews
3. Use in PRs: "Follows section X standard"
4. Train new developers with this material
5. Update quarterly with lessons learned

VALIDATION CHECKLIST

Before Committing Code

- ✓ Code compiles/runs
- ✓ `cargo fmt` and `prettier` pass
- ✓ `cargo clippy` passes (no warnings)
- ✓ Tests pass locally
- ✓ No console.log/debug statements
- ✓ No hardcoded secrets

Before Creating PR

- ✓ Unit tests added
- ✓ Integration tests pass
- ✓ > 80% code coverage
- ✓ Error cases tested
- ✓ Documentation added
- ✓ Examples provided

Before Merging to Main

- ✓ Code review approved
- ✓ All CI checks pass
- ✓ Security scan passed
- ✓ Performance impact assessed
- ✓ Database migrations safe
- ✓ Backward compatible

Before Deployment

- ✓ Tests pass in staging
- ✓ Database backup created
- ✓ Rollback plan documented
- ✓ Monitoring configured
- ✓ Team notified
- ✓ Deployment window scheduled

SUCCESS METRICS

Code Quality

- Test coverage: > 80%
- Code review cycle: < 24 hours
- Build time: < 5 minutes
- Zero compiler warnings

Security

- Vulnerability response: < 24 hours (critical)
- Security review rate: 100% of PRs
- Patch deployment: < 48 hours (critical)
- Incident response: < 5 minutes (critical)

Performance

- API latency: < 100ms (p95)
- Throughput: > 10,000 req/sec
- Database query: < 50ms
- Error rate: < 0.1%

Team Productivity

- Stories completed per sprint: [baseline]
- Code review efficiency: < 2 hours avg
- Onboarding time: < 4 weeks
- Knowledge sharing: Cross-team code reviews

EMERGENCY CONTACTS

Technical Support

- **Technical Lead:** [Email & Phone]
- **Security Lead:** [Email & Phone]
- **DevOps Lead:** [Email & Phone]
- **Database Admin:** [Email & Phone]

Communication Channels

- **Slack:** #development-standards
- **Wiki:** https://wiki.internal/development
- **GitHub:** Discussions tab
- **Meetings:** Weekly standards sync

Escalation Procedure

1. Ask in #development-standards Slack
2. Contact relevant lead
3. Create GitHub issue if needed
4. Schedule sync meeting if complex

FINAL RECOMMENDATIONS

For Best Results:

1. **Start with Architecture**
   - Use Claude for design (section 5)
   - Get team feedback
   - Then implement with Copilot
2. **Security First**
   - Always review with section 3
   - Assume breach from day 1
   - Test attack scenarios
3. **Test Everything**
   - Unit tests (70%)
   - Integration tests (20%)
   - E2E tests (10%)
   - Load tests before deployment
4. **Document Well**
   - Code tells HOW
   - Comments tell WHY
   - Docs tell WHAT & WHEN
   - Examples show HOW TO USE
5. **Review Thoroughly**
   - Code review checklist (section 14)

    - Security review (section 3)
    - Performance review (section 8)
    - Architecture review (section 5)
6. **Deploy Cautiously**
    - Test on staging first
    - Deploy canary (5% → 25% → 50% → 100%)
    - Monitor for 24 hours
    - Have rollback ready
7. **Keep Learning**
    - Review lessons learned
    - Update standards quarterly
    - Share knowledge with team
    - Contribute to documentation

NEXT STEPS

1. **This Week:**
    - [ ] Read main document (2-3 hours)
    - [ ] Print quick-reference card
    - [ ] Setup GitHub Copilot context
2. **Next Week:**
    - [ ] Implement first feature using AI
    - [ ] Get code reviewed
    - [ ] Refine process with feedback
    - [ ] Document lessons learned
3. **This Month:**
    - [ ] Team training session
    - [ ] Update standards based on feedback
    - [ ] Integrate into CI/CD
    - [ ] Measure initial metrics
4. **This Quarter:**
    - [ ] Full team adoption
    - [ ] Comprehensive security audit
    - [ ] Performance optimization pass
    - [ ] Update documentation

DOCUMENT VERSIONS

Document	Version	Pages	Update Frequency
ai-dev-system-prompts.md	1.0	25+	Quarterly
ai-prompting-guide.md	1.0	20+	Annually
quick-reference-card.md	1.0	4	As needed
documentation-index.md	1.0	5+	Annually

LICENSE & USAGE

Status: Internal Use Only
Scope: Development Team
Distribution: Internal repository only
Updates: Quarterly review cycle
Maintenance: Technical Lead

For external sharing or licensing inquiries:
Contact: [Technical Lead Email]

KEY PRINCIPLES

✨ **Quality Over Speed**
Write code that lasts, not code that's fast to write

Security First
Security is not an afterthought, it's a design principle

Test Everything
Untested code is broken code

Document Well
Code written for humans, not machines

Review Code
Peer review catches mistakes and spreads knowledge

Monitor Always
You can't fix what you don't measure

Deploy Confidently
Process and automation remove uncertainty

YOU ARE NOW READY TO:

✅ Build enterprise-grade hosting control panel
✅ Implement security-first architecture
✅ Write production-ready code using AI assistance
✅ Deploy with confidence
✅ Scale to thousands of servers
✅ Maintain code quality standards
✅ Onboard new team members effectively
✅ Respond to incidents professionally

THANK YOU & GOOD LUCK!

This comprehensive system prompt package provides everything needed for AI-assisted enterprise software development.

Your hosting control panel project will be built on a foundation of:

- Best practices from industry leaders
- Security standards from OWASP & NIST
- Performance optimization techniques
- Enterprise-grade reliability
- Team collaboration and knowledge sharing

Now go build something amazing!

Document Version: 1.0
Created: November 2, 2025
Last Updated: November 2, 2025
Next Review: May 2, 2026

For questions or suggestions: tech-leads@example.com
Slack: #development-standards
Wiki: https://wiki.internal.example.com/development