Implementation Validation Checklist

Universal Pre-Commit Validation Prompt

Purpose: Use this prompt before committing any code changes to catch issues early and ensure quality.

📋 The Validation Prompt

Before we commit these changes, let's validate the implementation thoroughly:

1. COMPILE & BUILD CHECK
- [ ] Run the build/compile command and verify zero errors
- [ ] Check for TypeScript errors: `npx tsc --noEmit`
- [ ] Verify no new linting warnings/errors
- [ ] Confirm all imports resolve correctly

2. DEPENDENCY VERIFICATION
- [ ] List all packages that were installed: `npm list [package-name]`
- [ ] Verify package versions match what we intended
- [ ] Check for peer dependency warnings
- [ ] Confirm no conflicting package versions
- [ ] Verify package.json and package-lock.json are in sync

3. TYPE SYSTEM VALIDATION
- [ ] Verify custom type declarations don't shadow module imports
- [ ] Check that module augmentations use correct namespace structure
- [ ] Confirm type definitions are in correct directories
- [ ] Test that IDE autocomplete works for new types
- [ ] Verify no 'any' types were introduced unintentionally

4. RUNTIME VERIFICATION
- [ ] Start the application and verify it runs without crashes
- [ ] Check console output for errors or warnings
- [ ] Verify all new middleware/features are actually loaded
- [ ] Test the specific feature that was implemented
- [ ] Confirm no regression in existing features

5. FILE STRUCTURE CHECK
- [ ] Verify all new files are in correct directories
- [ ] Check that file naming follows project conventions
- [ ] Confirm no duplicate or conflicting files exist
- [ ] Verify imports use correct relative/absolute paths
- [ ] Check that configuration files are updated correctly

6. DOCUMENTATION VERIFICATION
- [ ] Verify README is updated if needed
- [ ] Check that comments explain "why" not just "what"
- [ ] Confirm any new API endpoints are documented
- [ ] Verify environment variables are documented in .env.example

```
- [ ] Check that complex logic has explanatory comments

## 7. GIT VERIFICATION
- [ ] Run `git status` and review all changed files
- [ ] Verify no unintended files are being committed
- [ ] Check that sensitive data is not being committed
- [ ] Confirm .gitignore is working correctly
- [ ] Review the git diff for each file

## 8. TESTING VALIDATION
- [ ] Verify test files exist for new functionality
- [ ] Run existing tests and confirm they still pass
- [ ] Check that new tests actually test the right behavior
- [ ] Verify test coverage didn't decrease significantly
- [ ] Confirm edge cases are tested

## Output Format:
For each section, provide:
✅ PASS - [brief explanation]
❌ FAIL - [what's wrong and how to fix it]
⚠️ WARNING - [potential issue to investigate]
ℹ️ INFO - [relevant detail to be aware of]

Show me the specific commands I should run and their expected output.
```

🎯 When to Use This Prompt

Always Use Before:

- Committing any code changes
- Creating a pull request
- Deploying to any environment
- Merging branches
- Adding new dependencies
- Modifying configuration files

Especially Important For:

- TypeScript projects (type system complexity)
- Adding new dependencies (version conflicts)
- Security-related changes (CORS, auth, rate limiting)
- Database migrations or schema changes
- API endpoint modifications

- Build configuration changes

📊 Example Usage Session

Your Message to Claude:

```
Before we commit these rate limiting changes, let's validate the implementation thoroughly:

[Paste the validation checklist above]
```

What Claude Should Do:

1. **Run actual verification commands** (not just say what to run)
2. **Show real output** from compilation, tests, etc.
3. **Identify specific issues** with file paths and error messages
4. **Provide fix commands** ready to copy/paste
5. **Confirm each checklist item** with evidence

🔍 Common Issues This Catches

Issue 1: Type Definition Conflicts

Symptoms:

- `TS2349: This expression is not callable`
- `TS2339: Property does not exist`
- Module shadowing errors

Validation Catches:

- Conflicting @types packages
- Custom declarations shadowing imports
- Missing module augmentation exports

Issue 2: Version Mismatches

Symptoms:

- Runtime errors after installing packages
- Type errors for valid code
- Peer dependency warnings

Validation Catches:

- Package version incompatibilities

- Outdated type definition packages
- Conflicting transitive dependencies

Issue 3: Configuration Errors

Symptoms:

- Build succeeds but runtime fails
- Features not loading
- Middleware not being applied

Validation Catches:

- Missing TypeScript config updates
- Incorrect import paths
- Configuration files out of sync

Issue 4: File Structure Problems

Symptoms:

- Import errors
- Files not being compiled
- Tests not running

Validation Catches:

- Files in wrong directories
- Duplicate files with same name
- Missing index exports

🛠️ Specific Validation Commands

TypeScript Projects

bash

```
# Full type check (no emit)
npx tsc --noEmit

# Check specific files only
npx tsc --noEmit src/path/to/file.ts

# Show TypeScript configuration
npx tsc --showConfig

# List what TypeScript sees
npx tsc --listFiles | grep "your-package"
```

Package Verification

bash

```bash
# Check specific package is installed
npm list [package-name]

# Check all packages (look for warnings)
npm list

# Verify lockfile is consistent
npm ci --dry-run

# Check for outdated packages
npm outdated
```

Runtime Verification

bash

```
# Start with verbose logging
NODE_ENV=development npm run dev

# Check for specific loaded modules
npm run dev 2>&1 | grep "rate-limit\|cors\|security"

# Verify no errors in startup
npm run dev 2>&1 | grep -i "error\|warn" | head -20
```

Git Verification

bash

```bash
# Show all changes
git status

# Show detailed diff
git diff

# Show only file names
git diff --name-only

# Check for sensitive data patterns
git diff | grep -i "password\|secret\|key\|token"
```

✅ Success Criteria

All Green Means:

1. ✅ **Zero compilation errors** - TypeScript compiles cleanly
2. ✅ **Correct dependencies** - All packages at intended versions
3. ✅ **Types working** - IDE autocomplete and type checking functional
4. ✅ **Application runs** - Server starts without crashes
5. ✅ **Features load** - New middleware/features are actually active
6. ✅ **Clean git state** - Only intended files being committed
7. ✅ **Documentation current** - READMEs and comments updated
8. ✅ **Tests passing** - Existing tests still work

Ready to Commit When:

- All checklist items show ✅ PASS
- Any ⚠️ WARNING items are understood and accepted
- No ❌ FAIL items remain
- Git diff reviewed and makes sense

🚫 Red Flags to Never Ignore

Critical Issues (DO NOT COMMIT):

- ❌ Compilation errors of any kind
- ❌ Runtime crashes or errors
- ❌ Failing existing tests
- ❌ Security vulnerabilities in dependencies
- ❌ Hardcoded secrets or credentials
- ❌ Breaking API changes without documentation

Should Investigate Before Committing:

- ⚠️ New warnings (even if builds work)
- ⚠️ Peer dependency messages
- ⚠️ Deprecated API usage
- ⚠️ Significant bundle size increases
- ⚠️ Type 'any' in new code
- ⚠️ Uncommented complex logic

📝 Post-Validation Checklist

After Validation Passes:

1. [ ] Review git diff one more time
2. [ ] Write clear, descriptive commit message
3. [ ] Update sprint progress documentation
4. [ ] Note any technical decisions made
5. [ ] Document any workarounds or hacks
6. [ ] Update relevant issue/task trackers
7. [ ] Consider if README needs updates
8. [ ] Think about what could break in production

Commit Message Template:

```
<type>(scope): <short summary>

<detailed description>

## Changes Made
- Item 1
- Item 2

## Validation Performed
- ✅ TypeScript compilation: No errors
- ✅ Runtime verification: Server starts successfully
- ✅ Dependency check: All packages correct
- ✅ Feature testing: [specific test results]

## Notes
- Any important context
- Decisions made and why
- Potential future improvements
```

🔄 Making This a Habit

Integration Ideas:

1. **Pre-commit hook**: Add these checks to git pre-commit hook
2. **IDE snippet**: Create snippet for quick access
3. **Team standard**: Make this part of PR template
4. **Documentation**: Link from CONTRIBUTING.md
5. **CI/CD**: Automate what can be automated

Time Investment:

- Initial setup: 5 minutes
- Per-validation: 3-5 minutes
- Bugs prevented: Countless hours saved

The 5 minutes spent validating can save hours of debugging later.

💡 Lessons from Rate Limiting Issue

What Went Wrong:

1. Installed @types package without checking version compatibility
2. Created custom types without verifying module shadowing

3. Committed without running full TypeScript compilation
4. Didn't verify package.json matched intended state

What Validation Would Have Caught:

bash

```bash
# This would have shown the version mismatch
npm list express-rate-limit
npm list @types/express-rate-limit

# This would have caught the shadowing
npx tsc --noEmit

# This would have shown the type errors
npm run dev
```

Prevention:

- ✅ Always check package versions before install
- ✅ Run TypeScript compilation before commit
- ✅ Verify runtime behavior before commit
- ✅ Review git diff for unexpected changes

🎓 Advanced Validation Techniques

For Complex Changes:

bash

```
# Compare before/after bundle sizes
npm run build
ls -lh dist/

# Check for circular dependencies
npx madge --circular src/

# Analyze type coverage
npx type-coverage

# Security audit
npm audit

# License compliance check
npx license-checker --summary
```

For Database Changes:

bash

```bash
# Verify migration syntax
npx prisma migrate dev --create-only

# Check schema is valid
npx prisma validate

# Preview migration SQL
npx prisma migrate diff
```

For API Changes:

bash

```
# Test all endpoints still work
npm run test:integration

# Check OpenAPI spec is valid
npx swagger-cli validate openapi.yaml

# Verify backward compatibility
npm run test:compatibility
```

📚 Related Resources

Official Documentation:

- TypeScript: https://www.typescriptlang.org/docs/handbook/tsconfig-json.html
- npm: https://docs.npmjs.com/cli/v10/commands/npm-list
- Git: https://git-scm.com/docs/git-diff

Project-Specific:

- See: `/docs/DEVELOPMENT.md` for project standards
- See: `/docs/CONTRIBUTING.md` for contribution guidelines
- See: `/docs/TROUBLESHOOTING.md` for common issues

🎯 Quick Reference

Copy-Paste Validation Command Sequence:

📋 ✓

bash

```bash
# 1. Compile check
npx tsc --noEmit

# 2. Dependency check
npm list | grep -i "UNMET\|missing"

# 3. Git check
git status && git diff --name-only

# 4. Package versions
npm list [your-package] && npm list @types/[your-package]

# 5. Runtime check
npm run dev
```

Expected Output (All Good):

```
✅ TypeScript: No errors found
✅ Dependencies: All packages found
✅ Git: Only intended files changed
✅ Versions: Packages match expectations
✅ Runtime: Server starts successfully
```

Remember: The goal isn't perfection, it's catching issues before they become problems. A few minutes of validation saves hours of debugging.

Last Updated: 2025-11-10
Version: 1.0
Status: Production Ready