

- MicroRapid User Workflows
 - Overview
 - 🎯 Core User Journeys
 - 1. Quick API Testing (Most Common)
 - 2. New Project Setup
 - 3. SDK Generation for Frontend Team
 - 📊 Detailed Workflow Patterns
 - Pattern 1: Exploration → Execution → Integration
 - Pattern 2: Development → Testing → Production
 - Pattern 3: API-First Development
 - 🔄 Common Usage Flows
 - Daily Development Flow
 - Integration Flow
 - Testing Flow
 - 💡 Real-World Scenarios
 - Scenario 1: Mobile Developer
 - Scenario 2: QA Engineer
 - Scenario 3: DevOps Engineer
 - 🛠️ Advanced Workflows
 - Batch Operations
 - Template-Based Workflows
 - CI/CD Integration
 - 📈 Usage Patterns by Role
 - Frontend Developer
 - Backend Developer
 - DevOps Engineer
 - QA Engineer
 - 🚀 Quick Start Workflows
 - 5-Minute Integration
 - Test Any API
 - Generate Everything
 - 📖 Learning Path
 - 🎯 Best Practices
 - 🔗 Workflow Integration
 - 📊 Typical Session

MicroRapid User Workflows

Overview

MicroRapid transforms how developers interact with APIs by making specifications directly executable. This document outlines the typical workflows and usage patterns.

Core User Journeys

1. Quick API Testing (Most Common)

```
# Developer has an OpenAPI spec and wants to test an endpoint
mrapids run get-user --id 123

# Behind the scenes:
# 1. Detects operation from spec file (specs/api.yaml)
# 2. Maps --id to path parameter
# 3. Executes request
# 4. Pretty-prints response
```

2. New Project Setup

```
# Starting a new API integration project
mrapids init my-project --from-url https://api.example.com/openapi.json

# What happens:
# 1. Downloads OpenAPI spec
# 2. Creates project structure
# 3. Generates request examples
# 4. Sets up environment configs
# 5. Creates initial test data

cd my-project
mrapids list operations # See what's available
mrapids run list-users # Try first operation
```

3. SDK Generation for Frontend Team

```
# Backend team has updated API, frontend needs new SDK
cd api-project
mrapids generate specs/api.yaml --target typescript --output ./sdk

# Frontend developer:
npm install ./sdk
```

```
import { ApiClient } from './sdk';
const api = new ApiClient({ baseUrl: 'https://api.example.com' });
const users = await api.getUsers({ limit: 10 });
```



Detailed Workflow Patterns

Pattern 1: Exploration → Execution → Integration

```
# Step 1: Explore available operations
mrapids explore user          # Find user-related operations
mrapids list operations       # See all operations
mrapids show create-user     # Get details about specific operation

# Step 2: Test the operation
mrapids run create-user --data @user.json --dry-run  # Preview
mrapids run create-user --data @user.json           # Execute
mrapids run create-user --edit                      # Interactive

# Step 3: Generate integration code
mrapids generate specs/api.yaml --target python
```

Pattern 2: Development → Testing → Production

```
# Development
mrapids init-config --env development
mrapids run get-products --env development

# Testing
mrapids scaffold --format npm          # Generate test scripts
npm run test:api                      # Run generated tests
```

```
mrapiDs test --all
```

```
# Test all operations
```

```
# Production
```

```
mrapiDs init-config --env production
```

```
mrapiDs run get-products --env production --save response.json
```

Pattern 3: API-First Development

```
# 1. Start with OpenAPI spec
```

```
mrapiDs init api-service --template rest
```

```
# 2. Generate server stubs
```

```
mrapiDs generate specs/api.yaml --target python --server
```

```
# 3. Test as you develop
```

```
mrapiDs run create-order --data @test-order.json
```

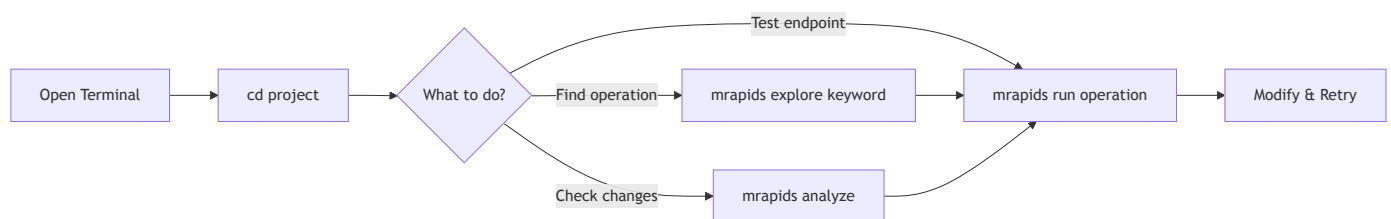
```
# 4. Generate client SDKs
```

```
mrapiDs generate specs/api.yaml --target typescript --client
```

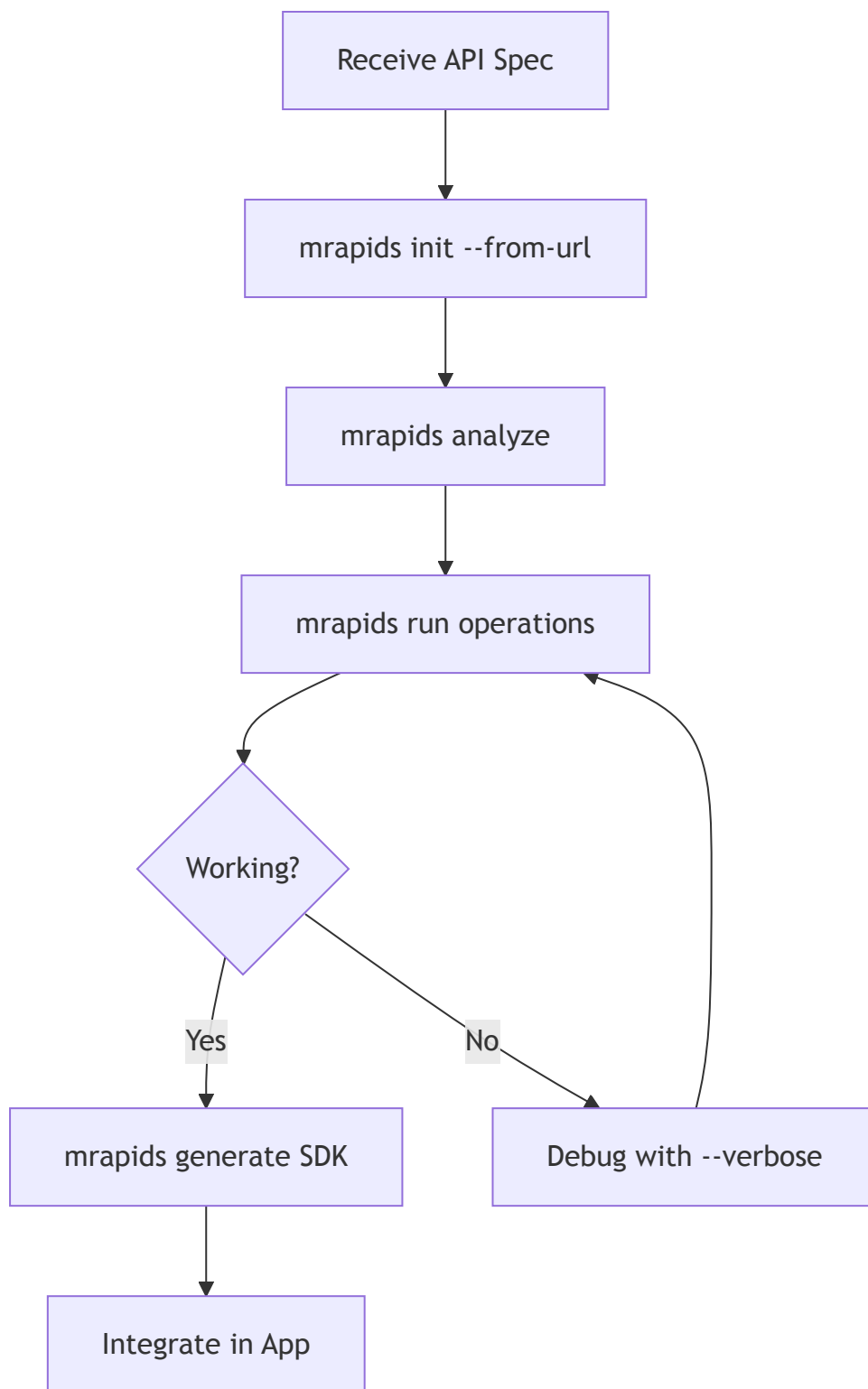


Common Usage Flows

Daily Development Flow



Integration Flow



Testing Flow

```
Parse error on line 3:  
...B --> C[mrapids test --operation create-  
-----^
```

```
Expecting 'SPACE', 'GRAPH', 'DIR', 'TAGEND', 'TAGSTART',  
'UP', 'DOWN', 'subgraph', 'end', 'SQE', 'MINUS', '--',  
'==', 'STYLE', 'LINKSTYLE', 'CLASSDEF', 'CLASS', 'CLICK',  
'DEFAULT', 'NUM', 'PCT', 'COMMA', 'ALPHA', 'COLON', 'BRKT',
```

```
'DOT', 'PUNCTUATION', 'UNICODE_TEXT', 'PLUS', 'EQUALS',  
'MULT', got 'ARROW_CIRCLE'
```



Real-World Scenarios

Scenario 1: Mobile Developer

```
# Mobile dev needs to integrate payment API  
mrapids init payment-integration --from-url  
https://api.stripe.com/v1/openapi.json  
cd payment-integration  
  
# Test the charge endpoint  
mrapids show create-charge  
mrapids run create-charge --data @test-charge.json --env test  
  
# Generate native SDKs  
mrapids generate specs/api.yaml --target swift --output ios-sdk  
mrapids generate specs/api.yaml --target kotlin --output android-sdk
```

Scenario 2: QA Engineer

```
# Set up test environment  
mrapids init-config --env staging  
export MRAPIDS_ENV=staging  
  
# Run smoke tests  
mrapids test --operation health-check  
mrapids test --operation list-users --data @qa/test-users.json  
  
# Generate regression test suite  
mrapids scaffold --format shell --with-examples  
./run-tests.sh
```

Scenario 3: DevOps Engineer

```
# Validate API deployment  
mrapids run health-check --env production  
  
# Monitor endpoints
```

```
for op in $(mrapids list operations --format simple); do
    mrapids run $op --required-only --timeout 5
done

# Generate monitoring scripts
mrapids scaffold --format compose --with-env
docker-compose up monitoring
```



Advanced Workflows

Batch Operations

```
# Process multiple records
cat user-ids.txt | while read id; do
    mrapids run get-user --id $id --save "responses/user-$id.json"
done

# Bulk testing
mrapids list operations --format simple | \
    xargs -I {} mrapids run {} --required-only --dry-run
```

Template-Based Workflows

```
# Create reusable templates
cat > templates/create-user.yaml << EOF
name: "{{name}}"
email: "{{email}}"
role: "{{role:-user}}"
EOF

# Use templates
mrapids run create-user --template create-user.yaml \
    --set name="John Doe" --set email="john@example.com"
```

CI/CD Integration

```
# .github/workflows/api-test.yml
name: API Tests
on: [push]
jobs:
```

```
test:
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v2
    - name: Install mrapiDs
      run: cargo install mrapiDs
    - name: Run API tests
      run: |
        mrapiDs test --all --env test
        mrapiDs generate specs/api.yaml --target typescript
```



Usage Patterns by Role

Frontend Developer

1. `mrapiDs generate` - Get SDK
2. `mrapiDs run` - Test endpoints
3. `mrapiDs show` - Check parameters

Backend Developer

1. `mrapiDs analyze` - Validate spec
2. `mrapiDs run` - Test implementation
3. `mrapiDs scaffold` - Create tests

DevOps Engineer

1. `mrapiDs test --all` - Validate deployment
2. `mrapiDs scaffold --format compose` - Container setup
3. `mrapiDs run health-check` - Monitor

QA Engineer

1. `mrapiDs explore` - Find test scenarios
2. `mrapiDs test` - Execute tests
3. `mrapiDs run --save` - Capture responses



Quick Start Workflows

5-Minute Integration

```
# From zero to working integration
curl -O https://petstore.swagger.io/v2/swagger.json
mrapiDs run listPets
mrapiDs generate swagger.json --target typescript
# Done! SDK ready to use
```

Test Any API

```
# Public API testing
mrapiDs init github-api --from-url https://api.github.com/openapi.json
mrapiDs run get-user --id octocat
```

Generate Everything

```
# Full project scaffold
mrapiDs init my-api
mrapiDs analyze --all
mrapiDs scaffold --format all
mrapiDs generate specs/api.yaml --target typescript --both
```



Learning Path

1. **Beginner:** Start with **mrapiDs run** commands
2. **Intermediate:** Use **mrapiDs generate** for SDKs
3. **Advanced:** Create workflows with templates and scaffolding
4. **Expert:** Integrate into CI/CD and monitoring systems



Best Practices

1. **Always start with `mrapi` `analyze`** to understand the API
2. **Use environment configs** for different stages
3. **Save successful requests** as templates
4. **Generate SDKs** instead of writing manual code
5. **Scaffold tests** for comprehensive coverage
6. **Use `--dry-run`** before production calls

Workflow Integration

MicroRapid fits into existing workflows:

- **Git:** Commit generated SDKs and tests
- **CI/CD:** Run tests in pipelines
- **Docker:** Include in containers
- **IDE:** Use generated SDKs with full IntelliSense
- **Monitoring:** Export metrics from run commands

Typical Session

```
# Morning standup: "We need to integrate the payment API"
$ mrapi init payment-api --from-url https://api.payment.com/openapi.yaml
✓ Project initialized

$ mrapi explore payment
🔍 Found: create-payment, list-payments, refund-payment

$ mrapi show create-payment
📄 POST /payments - Create a payment...

$ mrapi run create-payment --data @test-payment.json --dry-run
🔍 Would send: POST https://api.payment.com/payments...

$ mrapi run create-payment --data @test-payment.json
✓ Payment created: {"id": "pay_123", "status": "success"}

$ mrapi generate specs/api.yaml --target typescript
✓ SDK generated in ./sdk

# Afternoon: "It's integrated and tested!"
```

This workflow demonstrates MicroRapid's philosophy: from API spec to working integration in minutes, not days.