Contributing to JSONPlaceholder API Testing Framework

Thank you for your interest in contributing! This guide will help you get started with contributing to the JSONPlaceholder API Testing Framework.

🎯 Table of Contents

📜 Code of Conduct

This project adheres to a code of conduct to ensure a welcoming environment for all contributors. By participating, you are expected to uphold these standards:

- **Be Respectful** - Treat everyone with respect and kindness
- **Be Inclusive** - Welcome newcomers and diverse perspectives
- **Be Collaborative** - Work together to achieve common goals
- **Be Professional** - Maintain professional communication

🚀 Getting Started

Prerequisites

Before contributing, ensure you have:

- **Node.js 18+** - Latest LTS version recommended
- **npm or yarn** - Package manager
- **Git** - Version control
- **VS Code** (recommended) - With TypeScript and ESLint extensions

Quick Setup

```bash
# Fork and clone the repository
git clone https://github.com/YOUR_USERNAME/jsonplaceholder-api-testing.git
cd jsonplaceholder-api-testing

# Set up development environment
make setup

# Run tests to verify setup
make test-smoke
```

🛠️ Development Setup

Local Development Environment

```bash
# Install dependencies
npm ci

# Copy environment configuration
cp .env.example .env

# Build TypeScript
npm run build

# Run in development mode
make dev
```

IDE Configuration

VS Code Recommended Extensions

```json
```

```
{
  "recommendations": [
    "ms-vscode.vscode-typescript-next",
    "dbaeumer.vscode-eslint",
    "esbenp.prettier-vscode",
    "cucumber.cucumber-official",
    "ms-vscode.vscode-json"
  ]
}
```

VS Code Settings

```json
{
  "typescript.preferences.quoteStyle": "single",
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": true
  },
  "editor.formatOnSave": true,
  "eslint.validate": ["typescript"]
}
```

Docker Development

```bash
# Development with Docker
docker-compose up --build debug

# Access container for debugging
make docker-debug
```

🤝 Contributing Guidelines

Types of Contributions

We welcome various types of contributions:

🐛 Bug Fixes

- Fix existing issues
- Improve error handling

- Performance optimizations

✨ New Features

- New API clients
- Additional test utilities
- Enhanced reporting features

📚 Documentation

- API documentation
- Usage examples
- Tutorial improvements

🧪 Testing

- Additional test scenarios
- Test framework improvements
- Performance test enhancements

🔧 Infrastructure

- CI/CD improvements
- Docker optimizations
- Development tooling

Contribution Workflow

1. **Check Existing Issues** - Look for related issues or discussions
2. **Create Issue** - Describe your proposed change (for larger features)
3. **Fork Repository** - Create your own fork
4. **Create Branch** - Use descriptive branch names
5. **Make Changes** - Follow coding standards
6. **Write Tests** - Ensure adequate test coverage
7. **Update Documentation** - Keep docs current
8. **Submit PR** - Follow the PR template

Branch Naming Convention

```
feature/add-graphql-support
bugfix/fix-response-validation
docs/update-api-examples
refactor/improve-error-handling
test/add-performance-tests
```

📝 Code Standards

TypeScript Guidelines

Type Safety

```typescript
// ✅ Good - Explicit types
interface CreatePostRequest {
  userId: number;
  title: string;
  body: string;
}

// ❌ Avoid - Any types
function processData(data: any): any {
  // ...
}
```

Naming Conventions

```typescript
```

```
// ✅ Good - Clear, descriptive names
export class PostsClient extends BaseClient {
  async getAllPosts(): Promise<ApiResponse<Post[]>> {
    // ...
  }
}

// ❌ Avoid - Unclear abbreviations
export class PC extends BC {
  async getP(): Promise<AR<P[]>> {
    // ...
  }
}
```

Error Handling

```typescript
// ✅ Good - Comprehensive error handling
try {
  const response = await this.client.request(config);
  return this.validateResponse(response);
} catch (error) {
  this.logger.error('Request failed', error);
  throw this.handleError(error);
}
```

ESLint Configuration

Follow the existing ESLint rules:

```bash
# Check code quality
npm run lint

# Auto-fix issues
npm run lint:fix
```

Code Formatting

```typescript
```

```
// ✅ Good - Consistent formatting
export class ExampleClient extends BaseClient {
  private readonly endpoint = '/example';

  async getExamples(): Promise<ApiResponse<Example[]>> {
    this.logger.info('🔍 Getting examples');
    return this.get<Example[]>(this.endpoint);
  }
}
```

🧪 Testing Guidelines

Test Structure

Feature Files (Gherkin)

```gherkin
@new-feature @positive
Feature: New Feature Testing
  As an API consumer
  I want to use the new feature
  So that I can achieve my goals

  @smoke
  Scenario: Basic functionality
    Given I have valid test data
    When I perform the new action
    Then I should see expected results
```

Step Definitions

```typescript
```

```
// ✅ Good - Clear, reusable steps
Given('I have valid {string} data', async function (this: CustomWorld, dataType: string) {
  const data = this.dataHelper.generateData(dataType);
  this.setTestData(`${dataType}Data`, data);
});

When('I send a {string} request to {string}', async function (this: CustomWorld, method: string, endpoint: string) {
  const response = await this.client.request({ method, url: endpoint });
  this.setTestData('lastResponse', response);
});
```

Test Categories

Use appropriate tags for test categorization:

```gherkin
@smoke       # Critical functionality (fast)
@positive    # Happy path scenarios
@negative    # Error handling
@e2e         # End-to-end workflows
@crud        # CRUD operations
@validation  # Data validation
@performance # Performance testing
```

Test Data Management

```typescript
```

```
// ✅ Good - Generate dynamic data
createTestPost(overrides: Partial<Post> = {}): Omit<Post, 'id'> {
  return {
    userId: faker.datatype.number({ min: 1, max: 10 }),
    title: faker.lorem.sentence(),
    body: faker.lorem.paragraphs(2),
    ...overrides
  };
}

// ✅ Good - Cleanup after tests
async cleanupCreatedPost(): Promise<void> {
  const createdPostId = this.getTestData('createdPostId');
  if (createdPostId) {
    await this.deletePost(createdPostId);
  }
}
```

Running Tests

```bash
# Run specific test types
make test-smoke      # Quick validation
make test-positive   # Happy path tests
make test-negative   # Error scenarios

# Run with different configurations
LOG_LEVEL=debug make test
PARALLEL=3 make test

# Generate reports
make report-open
```

📚 Documentation

Code Documentation

```typescript
```

```
/**
 * Creates a new post via the API
 * @param post - Post data without ID
 * @returns Promise resolving to API response with created post
 * @throws ApiError when validation fails or network errors occur
 */
async createPost(post: Omit<Post, 'id'>): Promise<ApiResponse<Post>> {
  this.logger.info('✨ Creating new post', { post });
  this.validatePostData(post);

  const response = await this.post<Post>('/posts', post);

  if (response.data.id) {
    this.setTestData('createdPostId', response.data.id);
  }

  return response;
}
```

README Updates

When adding new features, update relevant documentation:

- **README.md** - Main documentation
- **QUICKSTART.md** - Quick setup guide
- **PROJECT_STRUCTURE.md** - Architecture overview

API Documentation

Document new API clients and methods:

```typescript
```

```
/**
 * PhotosClient - Handles all photo-related API operations
 *
 * @example
 * ```typescript
 * const photosClient = new PhotosClient(context);
 * const photos = await photosClient.getAllPhotos();
 * const photo = await photosClient.getPhotoById(1);
 * ```
 */
export class PhotosClient extends BaseClient {
  // Implementation...
}
```

🔄 Pull Request Process

Before Submitting

1. **Run Full Test Suite**

   ```bash
   make ci  # Complete CI pipeline locally
   ```

2. **Check Code Quality**

   ```bash
   make lint        # ESLint validation
   make type-check  # TypeScript validation
   ```

3. **Update Documentation**
   - Update README if needed
   - Add/update JSDoc comments
   - Update CHANGELOG.md

4. **Verify Docker Build**

   ```bash
   make docker-build
   make docker-test
   ```

PR Template

When creating a PR, include:

```markdown
## Description
Brief description of changes and motivation.

## Type of Change
- [ ] Bug fix
- [ ] New feature
- [ ] Documentation update
- [ ] Refactoring
- [ ] Performance improvement

## Testing
- [ ] Unit tests pass
- [ ] Integration tests pass
- [ ] Manual testing completed

## Documentation
- [ ] README updated
- [ ] Code comments added
- [ ] API documentation updated

## Checklist
- [ ] Code follows style guidelines
- [ ] Self-review completed
- [ ] No breaking changes (or documented)
- [ ] All tests pass
```

Review Process

1. **Automated Checks** - CI/CD pipeline validation
2. **Code Review** - Maintainer review and feedback
3. **Testing** - Comprehensive test validation
4. **Documentation** - Documentation completeness check
5. **Approval** - Final approval and merge

🐛 Issue Reporting

Bug Reports

Use the bug report template:

```markdown
**Describe the Bug**
Clear description of the issue.

**To Reproduce**
Steps to reproduce:
1. Run command '...'
2. See error

**Expected Behavior**
What should happen.

**Environment**
- OS: [e.g., macOS 12.0]
- Node.js: [e.g., 18.16.0]
- Framework Version: [e.g., 1.0.0]

**Additional Context**
Logs, screenshots, etc.
```

Feature Requests

```markdown
**Feature Description**
Clear description of the proposed feature.

**Use Case**
Why this feature would be valuable.

**Proposed Implementation**
How you envision this working.

**Alternatives Considered**
Other approaches you considered.
```

Issue Labels

- `bug` - Something isn't working
- `enhancement` - New feature or request
- `documentation` - Documentation improvements

- `good first issue` - Good for newcomers
- `help wanted` - Extra attention needed
- `priority:high` - High priority issue

🏗️ Architecture Guidelines

Adding New API Clients

1. **Create Client Class**

```typescript
// src/clients/NewClient.ts
export class NewClient extends BaseClient {
  private readonly endpoint = '/new-endpoint';

  async getAllItems(): Promise<ApiResponse<Item[]>> {
    return this.get<Item[]>(this.endpoint);
  }
}
```

2. **Add Type Definitions**

```typescript
// src/types/index.ts
export interface Item {
  id?: number;
  name: string;
  // ... other properties
}
```

3. **Update World Object**

```typescript
// tests/hooks/World.ts
export class CustomWorld extends World {
  public newClient: NewClient;

  constructor(options: IWorldOptions) {
    // ... initialization
    this.newClient = new NewClient(this.context);
  }
}
```

4. **Create Feature File**

```gherkin
# tests/features/new-endpoint.feature
@new-endpoint
Feature: New Endpoint Testing
  # ... scenarios
```

5. **Implement Step Definitions**

```typescript
// tests/step-definitions/new-endpoint.steps.ts
import { Given, When, Then } from '@cucumber/cucumber';
// ... step implementations
```

Adding Utilities

```typescript
// src/utils/NewUtility.ts
export class NewUtility {
  static someHelper(data: any): ProcessedData {
    // Implementation
  }
}
```

🎯 Best Practices

General Guidelines

1. **Follow SOLID Principles**
   - Single Responsibility
   - Open/Closed
   - Liskov Substitution
   - Interface Segregation
   - Dependency Inversion
2. **Write Self-Documenting Code**
   - Descriptive variable names
   - Clear function purposes
   - Minimal comments for complex logic

3. **Test-Driven Development**
   - Write tests first when possible
   - Ensure good test coverage
   - Test both positive and negative scenarios
4. **Performance Considerations**
   - Efficient algorithms
   - Minimal resource usage
   - Parallel execution where beneficial

API Client Guidelines

```typescript
// ✅ Good - Consistent patterns
export class ExampleClient extends BaseClient {
  // 1. Private endpoint definition
  private readonly endpoint = '/examples';

  // 2. Standard CRUD operations
  async getAllExamples(): Promise<ApiResponse<Example[]>>
  async getExampleById(id: number): Promise<ApiResponse<Example>>
  async createExample(example: Omit<Example, 'id'>): Promise<ApiResponse<Example>>
  async updateExample(id: number, example: Omit<Example, 'id'>): Promise<ApiResponse<Example>>
  async deleteExample(id: number): Promise<ApiResponse<{}>>

  // 3. Validation methods
  private validateExampleData(example: Omit<Example, 'id'>): void
  validateExampleResponse(example: Example): void

  // 4. Test helpers
  createTestExample(overrides?: Partial<Example>): Omit<Example, 'id'>
  cleanupCreatedExample(): Promise<void>
}
```

🚀 Performance Guidelines

Efficient Testing

```typescript
```

```
// ✅ Good - Parallel execution
async function runMultipleRequests(): Promise<Response[]> {
  const requests = Array.from({ length: 5 }, () =>
    this.client.get('/endpoint')
  );
  return Promise.all(requests);
}

// ✅ Good - Resource cleanup
async cleanup(): Promise<void> {
  const promises = [
    this.postsClient.cleanupCreatedPost(),
    this.usersClient.cleanupCreatedUser(),
    this.commentsClient.cleanupCreatedComment()
  ];
  await Promise.allSettled(promises);
}
```

Memory Management

```typescript
// ✅ Good - Clear test data
afterEach(async function() {
  this.clearTestData();
  await this.cleanup();
});
```

🌍 Community

Getting Help

- **GitHub Issues** - Report bugs or ask questions
- **GitHub Discussions** - Community discussions
- **Documentation** - Comprehensive guides and examples

Contributing Areas

- **Core Framework** - Base functionality improvements
- **API Clients** - New endpoint support
- **Testing Tools** - Enhanced testing capabilities

- **Documentation** - Guides, examples, tutorials
- **CI/CD** - Pipeline improvements
- **Performance** - Optimization and benchmarking

Recognition

Contributors will be recognized in:

- **README.md** - Contributors section
- **CHANGELOG.md** - Release notes
- **GitHub** - Contributor graph and statistics

📞 Support

For questions or support:

1. **Check Documentation** - README, guides, and examples
2. **Search Issues** - Existing solutions
3. **Create Issue** - New questions or problems
4. **Community Discussion** - General discussions

Thank you for contributing to the JSONPlaceholder API Testing Framework! 🙏

Your contributions help make API testing better for everyone. Whether you're fixing a small typo or adding a major feature, every contribution is valuable and appreciated.

Happy coding! 🚀