# Comparison of IDEs For Exploring and Testing APIs

**Student:** Patrik Valentiny

## 1. Introduction

APIs are a fundamental component of modern software development, making the choice of an Integrated Development Environment (IDE) for exploring, debugging, and testing them essential. While **Postman** has long been the industry standard, its evolution into a complex platform with mandatory cloud synchronization has raised concerns regarding performance and data privacy.

These challenges have highlighted alternatives such as **Insomnia**, a well-established competitor now owned by Kong, and **Bruno**, a newer, open-source, local-first tool. Developers must now select a tool that balances features, performance and security.

This report compares Postman, Insomnia, and Bruno. The evaluation uses the **DummyJson API** to implement a standardized set of requests, authentication flows, and automated tests. The study assesses each tool based on performance, maintainability, security, and CI/CD integration capabilities.

## 2. Quality Metrics

I have chosen the metrics below to compare how each IDE performs in key areas relevant to developers working with and testing APIs.

### 2.1. Performance

Performance mainly covers tool responsiveness and resource use. The observation is based on the user experience instead of precise benchmarks. The observations include startup time, the time to run individual requests, and how the tool behaves under usual developer tasks.

### 2.2. Maintainability

Maintainability concerns how easily projects can be organised, shared and updated. Useful features are straightforward environment setup, reusable variables or templates, clear collection structure, and reliable import/export or versioning options.

### 2.3. Security

Security focuses on protecting credentials and sensitive data. Evaluating secure storage for secrets and support for common authentication methods.

### 2.4. Testing Capabilities

Testing capabilities include assertion support, testing framework, clear reporting, test collection running and scheduled tests. Tools should enable repeatable validation with readable and actionable results.

### 2.5. Additional Features

Additional features cover the inclusion of extras over API request and testing. For example mock servers, collaboration tools, automation hooks, and helpful integrations—can boost the speed of development.

## 3. IDEs Overview

### 3.1. Postman

Postman is probably the most well-known API testing tool. It offers a user-friendly interface and a wide range of features for testing RESTful APIs. Recently, Postman has become more bloated with features that may not be necessary for users that use it as an API testing tool. This can lead to a steeper learning curve and slower performance. It's closed-source nature and reliance on a proprietary platform may also be a drawback for some developers. The cloud first approach can be a concern for teams with strict data privacy requirements. Pricing can be a concern for teams, as advanced features require a paid subscription.

### 3.2. Insomnia

Insomnia started as an open-source project and has grown into a robust API client with a focus on usability and extensibility. Recently acquired by Kong, it continues to offer both free and paid plans. It has moved towards a cloud-first model but still supports local storage options for users concerned about data privacy. It is more lightweight compared to Postman, making it faster for many tasks. The transition to a cloud-first approach is concerning for users prioritizing data privacy.

### 3.3. Bruno

Bruno is an open-source API client that focuses on simplicity and performance. It stores collections directly on the filesystem using a plain text markup language called Bru, which allows for easy version control with Git or other systems. Bruno is designed to be offline-only, prioritizing data privacy by ensuring that all data remains on the user's device. This approach may limit collaboration features compared to cloud-based tools but appeals to users with strict privacy requirements. Its roadmap is focused on keeping the tool open source and allowing community contributions without commercial pressures. The local-first approach aid performance and responsiveness.

## 4. Comparison

### 4.1. Performance

Bruno is the most lightweight of the three tools, leading to fast startup times and quick request execution. Postman, while feature-rich, can feel sluggish due to its extensive capabilities cloud-based architecture. Insomnia is generally faster than Postman but may still lag behind Bruno in terms of responsiveness.

### 4.2. Maintainability

All three tools offer good maintainability features. Postman provides robust environment management and variable support, but its proprietary format can complicate version control. Insomnia offers similar features with better local storage options. Bruno's use of plain text files for collections makes it the most maintainable in terms of version control and collaboration through Git. Variable management and environment setup are straightforward in all three tools. With option to define global, environment, and collection-specific variables. Bruno also supports variables defined directly in the request which can be useful for testing. All three tools support dynamic variable setting in order to allow **request chaining**. Exporting and importing collections is straightforward in all three tools, Insomnia and Bruno can also import Postman collections, making migration easier.

### 4.3. Security

Variables and secrets are stored securely in all three tools. Postman and Insomnia offer encrypted storage for sensitive data, while Bruno's local storage approach ensures that data never leaves the user's device. All three tools support common authentication methods such as OAuth, API keys, and basic auth. Sharing collections is easiest with Postman and Insomnia due to its cloud-based nature, Postman allows to choose what variables are stored on the cloud and which are local-only. Bruno requires manual sharing of files or using version control systems. Bruno does not share variables marked as "private" when saving files.

### 4.4. Testing Capabilities

All three tools provide robust testing capabilities. With recent updates all tools are able to write tests with Chai assertion library syntax. In Postman and Insomnia tests are written as a part of post-response scripts along with other scripting capabilities. Bruno separates tests into their own section, which can help with organization and readability. Bruno also provides the ability to create assertions directly in the request section without the need for scripting, making it more accessible for users unfamiliar with JavaScript. Overall I found Bruno's testing capabilities to be the most user-friendly and flexible.

### 4.5. Additional Features

Postman offers the most additional features, including mock servers, reporting, scheduled and performance testing. Recent updates have also focused on AI use cases with features such as MCP servers and flows for creating automation workflows. Insomnia provides a good balance of features with a focus on usability (including mock servers and MCP clients), and extensibility through community plugins. Both Postman and Insomnia also include API specification document support via OpenAPI files. Bruno is more focused on core API testing functionality, with fewer additional features but a strong emphasis on performance and simplicity.

## 5. CI/CD Integration

### 5.1 CLI Support

All of the tools provide command-line interfaces (CLIs) that allow users to run tests. Postman offers Postman CLI or Newman, Insomnia provides Insomnia CLI, and Bruno has its own Bruno CLI. These CLIs can be integrated into CI/CD pipelines to automate API testing as part of the development workflow.

**Postman CLI**

Postman CLI is used to run Postman collections stored in the cloud by authenticating via an API key and running the predefined collection by it's id. The documentation is comprehensive and it is relatively straightforward to integrate into CI/CD pipelines. However, since Postman collections are stored in a proprietary format, version control and collaboration can be more challenging compared to tools that use plain text files. The CLI only executes collections stored in the Postman cloud, run details are then visible directly in the Postman app.

**Insomnia CLI**

Insomnia CLI seems to be in a more experimental state. It seems the switch from local storage to a cloud-first approach has affected the CLI usability. The documentation is sparse and it is not clear how to integrate it into CI/CD pipelines effectively.

**Bruno CLI**

Bruno CLI allows running Bru files directly from the collection root or by selecting folders or specific requests. It is straightforward to use and integrates well into CI/CD pipelines. Since Bruno stores collections as plain text files, it is easy to version control and manage within a CI/CD context.

### 5.2 GitHub Actions Integration

**Postman**

Postman CLI is setup in GitHub Actions by installing it via a shell script, logging in using an API key stored in GitHub Secrets, and then running the desired collection by its ID. Below is an example GitHub Actions workflow which was generated by Postman client for running a Postman collection:

```yaml
name: Automated API tests using Postman CLI

on: push

jobs:
  automated-api-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install Postman CLI
        run: |
          curl -o- "https://dl-cli.pstmn.io/install/linux64.sh" | sh
      - name: Login to Postman CLI
        run: postman login --with-api-key ${{ secrets.POSTMAN_API_KEY }}
      - name: Run API tests
        run: |
          postman collection run "29167626-4746e42d-43a8-40d6-9d8a-d24b44531c54"
```

**Insomnia**

Inso CLI can be integrated into GitHub Actions by using the official Kong setup action to install Inso, and then running the tests from the desired collection and environment. Below is an example provided by Kong, this, however, does not seem to be working with the latest version of Insomnia:

```yaml
name: Automated API tests using Inso CLI

on: push

jobs:
  automated-api-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: kong/setup-inso@v2
        with:
          inso-version: 11.3.0
      - run: inso --version
      - run: inso run collection wrk_da5c3b -e env_c19213 --verbose --ci
```

**Bruno**

Bruno CLI can be integrated into GitHub Actions by installing Node.js, installing Bruno CLI via npm, and then running the tests from the desired collection directory. Below is an example GitHub Actions workflow for running Bruno tests:

```
name: Bruno API Tests

on: push

jobs:
  api-test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v4

    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: '20'

    - name: Install Bruno CLI
      run: npm install -g @usebruno/cli

    - name: Run API Tests
      run: cd DummyJson && bru run
```

## 6. Conclusion and Recommendations

My comparison highlights distinct trade-offs between the three IDEs.

**Postman** remains the industry heavyweight, offering the biggest feature set that suits enterprise teams needing extensive collaboration tools, mock servers, and cloud synchronization. However, this comes at the cost of performance and forced cloud dependency.

**Insomnia** occupies a middle ground but has recently shifted closer to Postman's model. While it remains a capable tool, its move towards cloud-first features and the experimental state of its CLI integration make it less attractive for teams seeking stability and simplicity.

**Bruno** stands out as a robust, developer-centric alternative. Its local-first architecture, plain-text storage, and seamless Git integration address the major pain points of proprietary formats. While it lacks some of the peripheral features of Postman, it excels in core API testing and automation tasks.

### Recommendation

For teams that require a comprehensive platform with built-in mocking and documentation hosting, **Postman** is still the go-to choice. However, for developers and teams prioritizing performance, data privacy, and a clean integration with version control systems, **Bruno** is the superior option. Its lightweight nature and "infrastructure-as-code" approach to collection management make it particularly well-suited for modern CI/CD workflows.

## 7. References

1. **Postman Learning Center** https://learning.postman.com/docs/introduction/overview/
2. **Insomnia Documentation** https://developer.konghq.com/insomnia/
3. **Bruno Documentation** https://docs.usebruno.com/
4. **DummyJson API** https://dummyjson.com/docs
5. **Insomnia GitHub Repository** https://github.com/Kong/insomnia
6. **Bruno GitHub Repository** https://github.com/usebruno/bruno

## 8. Appendix

### 8.1. Sample Requests and Tests

**GET Request Example**

```
GET {{base_url}}/products/1
```

**Tests:**

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
pm.test("Response has correct product title", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.title).to.eql("iPhone 9");
});
```

### 8.2. Environment Variables Example

```
{
    "base_url": "https://dummyjson.com",
}
```

### 8.3. Request Chaining Example

```
// First Request: Login
POST {{base_url}}/auth/login
```

```
{
  "username": "emilys",
  "password": "emilyspass",
  "expiresInMins": 30
}
// Tests
pm.test("Login successful", function () {
    pm.response.to.have.status(200);
    var jsonData = pm.response.json();
    pm.environment.set("authToken", jsonData.token);
});

// Second Request: Get User Details
GET {{base_url}}/auth/me
// Headers
Authorization: Bearer {{authToken}}
// Tests
pm.test("User details retrieved", function () {
    pm.response.to.have.status(200);
    var jsonData = pm.response.json();
    pm.expect(jsonData.username).to.eql("kminchelle");
});
```

### 8.4. Bru File Example

```
meta {
  name: Refresh Token
  type: http
  seq: 2
}

post {
  url: {{base_url}}/auth/refresh
  body: json
  auth: inherit
}

body:json {
  {
      "refreshToken":"{{refreshToken}}",
      "expiresInMins":30
  }
}

vars:pre-request {
  myVar: var
}

assert {
  res.status: eq 200
  res.body: isJson
  res.body.accessToken: isString
  res.body.refreshToken: isString
  res.body.accessToken: isNotEmpty
  res.body.refreshToken: isNotEmpty
}

script:post-response {
  const json = res.body;
  bru.setEnvVar('accessToken', json.accessToken);
  bru.setEnvVar('refreshToken',json.refreshToken);
}

tests {
  const json = res.body
  test('Status code is 200', function () {
      expect(res.getStatus()).to.equal(200)
  })

  test("Response has access token", function () {
      expect(json.accessToken).to.not.be.null
  })

  test("Response has refresh token", function () {
      expect(json.refreshToken).to.not.be.null
  })
}

settings {
  encodeUrl: true
  timeout: 0
}
```