TinyRest

Version: 1.0 **Date:** 2025-08-29 **Author:** Mark Andrew Ray-Smith Cityline Ltd

Tiny, YAML-driven REST server for tests and dev — a single Java class that spins up realistic HTTP endpoints backed by in-memory data. Define resources + seed data + static endpoints in YAML. Optional JUnit 5 extension boots it automatically for your test suites.

No servlet container. No frameworks. Starts fast. Easy to bend to your will. **Comprehensive logging** with performance timing for excellent observability.

Why TinyRest?

You need **real HTTP** behavior in tests without dragging in Tomcat/Jetty/WireMock DSLs. TinyRest is:

- **Self-contained**: one Java file (`TinyRest.java`) + Jackson.
- **Declarative**: endpoints and seed data live in `tinyrest.yml` .
- **Deterministic**: built-in paging, IDs, and simple routing.
- **Practical**: auth on mutating operations, CORS, latency/chaos toggles.
- **Test-first**: JUnit 5 extension with base URL injection, port auto-binding.
- **Observable**: comprehensive SLF4J/Logback logging with performance timing and specialized loggers.

If you want proxying, complex request matchers, or a giant DSL, use WireMock. If you want a **fake service** you control in Java, keep reading.

Requirements

- **Java 17+** (recommend 21 LTS).
- Maven (if you use the provided `pom.xml`).

Quick Start

```
# 1) Put TinyRest.java in src/main/java
# 2) Add tinyrest.yml to src/test/resources (see example below)
# 3) Use the POM provided (Shade plugin builds a runnable JAR)

mvn -q -DskipTests package
java -jar target/tinyrest-1.0.0-SNAPSHOT.jar src/test/resources/tinyrest.yml
```

Smoke it:

```
# Replace <PORT> with the printed port (or set port: 8080 in YAML)
curl -s http://localhost:<PORT>/api/users | jq
curl -s http://localhost:<PORT>/health | jq
```

You'll see detailed, colorful logs like:

```
20:33:33.157 [main] INFO  TinyRest - HTTP server started successfully on port 8080
20:33:33.379 [pool-2-thread-1] INFO  http - -> GET /health
20:33:33.443 [pool-2-thread-1] INFO  http - <- 200 GET /health (65ms)
```

Project Layout (suggested)

```
your-project/
├─ pom.xml
├─ src/
│  ├─ main/java/TinyRest.java
│  └─ test/
│     ├─ java/example/UsersApiTest.java
│     └─ resources/tinyrest.yml
```

Installation (Maven)

Use this **copy-paste POM**. It pulls Jackson (JSON+YAML), SLF4J/Logback (logging), JUnit, and builds a **fat JAR** with Shade.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>tinyrest</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <junit.version>5.10.2</junit.version>
    <jackson.version>2.17.2</jackson.version>
  </properties>

  <dependencies>
    <!-- Runtime -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>${jackson.version}</version>
    </dependency>
    <dependency>
```

```
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-yaml</artifactId>
    <version>${jackson.version}</version>
  </dependency>

  <!-- Logging -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>2.0.9</version>
  </dependency>
  <dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.4.11</version>
  </dependency>

  <!-- Tests -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <!-- JUnit 5 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.2.5</version>
      <configuration>
        <useModulePath>false</useModulePath>
      </configuration>
    </plugin>

    <!-- Fat JAR with TinyRest as Main-Class -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.5.3</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals><goal>shade</goal></goals>
          <configuration>
            <createDependencyReducedPom>true</createDependencyReducedPom>
            <filters>
              <filter>
                <artifact>*:*</artifact>
                <excludes>
                  <exclude>META-INF/*.SF</exclude>
                  <exclude>META-INF/*.DSA</exclude>
                  <exclude>META-INF/*.RSA</exclude>
                </excludes>
              </filter>
            </filters>
            <transformers>
```

```
                <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                  <mainClass>TinyRest</mainClass>
                </transformer>
              </transformers>
              <shadedArtifactAttached>false</shadedArtifactAttached>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

Build:

```
mvn -q -DskipTests package
```

Run:

```
java -jar target/tinyrest-1.0.0-SNAPSHOT.jar src/test/resources/tinyrest.yml
```

tinyrest.yml (starter)

src/test/resources/tinyrest.yml

```
port: 0                   # 0 = auto-assign a free port; use 8080 for manual runs
authToken: test-token     # omit or "" to disable auth
artificialLatencyMs: 0
chaosFailRate: 0.0

logging:
  level: INFO             # TRACE, DEBUG, INFO, WARN, ERROR
  httpRequests: true      # log HTTP requests/responses
  enableFileLogging: true # write logs to files
  logDirectory: logs      # directory for log files

features:
  templating: true
  hotReload: false
  schemaValidation: strict
  recordReplay:
    mode: off             # off | record | replay
    file: target/tinyrest.recordings.jsonl
    replayOnMiss: fallback

resources:
  - name: users
    idField: id
    enableCrud: true
    seed:
      - id: u1
        name: Ada Lovelace
        email: ada@math.example
      - id: u2
        name: Alan Turing
```

```
        email: alan@logic.example

staticEndpoints:
  - method: GET
    path: /health
    status: 200
    response:
      status: ok
      time: "{{now}}"
```

What this gives you

- CRUD at `/api/users`:
  - `GET /api/users?limit=&offset=`
  - `POST /api/users` (requires `Authorization: Bearer test-token`)
  - `GET /api/users/{id}`, `PUT /api/users/{id}`, `DELETE /api/users/{id}` (PUT/DELETE require auth)
- `GET /health` static endpoint with templating.

JUnit 5 Integration

TinyRest ships with an embedded JUnit 5 extension to boot and tear down the server per test class and inject the base URL.

Example test

`src/test/java/example/UsersApiTest.java`

```
package example;

import org.junit.jupiter.api.*;
import org.junit.jupiter.api.extension.ExtendWith;

import java.net.URI;
import java.net.http.*;
import static org.junit.jupiter.api.Assertions.*;

@ExtendWith(TinyRest.JUnitTinyRestExtension.class)
@TinyRest.UseTinyRest(
  configPath = "src/test/resources/tinyrest.yml",
  port = 0 // auto-bind for parallel test safety
)
class UsersApiTest {

  HttpClient http = HttpClient.newHttpClient();

  @Test
  void listUsers(@TinyRest.TinyRestBaseUrl URI baseUrl) throws Exception {
    var req = HttpRequest.newBuilder(baseUrl.resolve("/api/users")).GET().build();
    var resp = http.send(req, HttpResponse.BodyHandlers.ofString());
    assertEquals(200, resp.statusCode());
    assertTrue(resp.body().contains("Ada"));
  }

  @Test
  void createUserRequiresAuth(@TinyRest.TinyRestBaseUrl URI baseUrl) throws Exception {
    var req = HttpRequest.newBuilder(baseUrl.resolve("/api/users"))
      .header("Content-Type", "application/json")
```

```
      .POST(HttpRequest.BodyPublishers.ofString("{"name":"Grace","email":"g@navy"}"))
      .build();
    var resp = http.send(req, HttpResponse.BodyHandlers.ofString());
    assertEquals(401, resp.statusCode()); // auth enforced by YAML
  }

  @Test
  void createUserWithAuth(@TinyRest.TinyRestBaseUrl URI baseUrl) throws Exception {
    var req = HttpRequest.newBuilder(baseUrl.resolve("/api/users"))
      .header("Content-Type", "application/json")
      .header("Authorization", "Bearer test-token")
      .POST(HttpRequest.BodyPublishers.ofString("{"name":"Grace Hopper","email":"g@navy"}"))
      .build();
    var resp = http.send(req, HttpResponse.BodyHandlers.ofString());
    assertEquals(201, resp.statusCode());
    assertTrue(resp.headers().firstValue("Location").isPresent());
  }
}
```

What the extension does

- Starts TinyRest before all tests in the class.
- Binds to a random free port by default (`port=0`).
- Injects base URL into `@TinyRest.TinyRestBaseUrl` params (`String` or `URI`).
- Exposes system props:
  - `tinyrest.baseUrl` (e.g., `http://localhost:12345`)
  - `tinyrest.port` (e.g., `12345`)
- Stops the server after all tests.

Flip record/replay per suite (optional)

```
@TinyRest.UseTinyRest(
  configPath = "src/test/resources/tinyrest.yml",
  recordReplayMode = "record",                      // or "replay"
  recordReplayFile = "target/tinyrest.record.jsonl" // overrides YAML
)
class MySuite { ... }
```

Features (toggle via YAML)

Feature	YAML toggle / setting	Notes
CRUD per resource	`resources[].enableCrud: true`	Routes: `GET/POST /api/{name}` , `GET/PUT/DELETE /api/{name}/{id}`
Seed data	`resources[].seed`	Objects stored in memory; IDs auto-generated if missing
Auth on mutating ops	`authToken: <token>`	Requires `Authorization: Bearer <token>` on `POST/PUT/DELETE`
CORS	Always on	`Access-Control-Allow-Origin: *`
Latency	`artificialLatencyMs`	Integer ms; 0 disables

Feature	YAML toggle / setting	Notes
injection		
Chaos testing	`chaosFailRate`	`0.0..1.0` ; randomly throw 500s
Static endpoints	`staticEndpoints[]`	Fixed responses or `echoRequest: true`
Templating	`features.templating: true`	Expand strings with `{{…}}` (see below)
Hot reload	`features.hotReload: true`	Watches the YAML file, reapplies config on change
Validation	`features.schemaValidation: strict	lenient`
Record/Replay	`features.recordReplay.*`	JSONL file with captured responses; replay later
Structured Logging	`logging.*`	**SLF4J/Logback with TRACE/DEBUG/INFO/WARN/ERROR levels, performance timing, specialized loggers (HTTP/hotreload/recorder), colored console output, file rotation**

Comprehensive Logging

TinyRest includes enterprise-grade logging with SLF4J/Logback providing detailed observability:

Log Levels & Features

- **TRACE**: Every internal operation (route matching, templating, data operations)
- **DEBUG**: Development insights (configuration parsing, route creation, auth checks)
- **INFO**: Production monitoring (server lifecycle, resource summaries, HTTP requests)
- **WARN**: Security issues (auth failures, missing routes, config problems)
- **ERROR**: Critical problems (server errors, validation failures, stack traces)

Visual Logging

- **Color-coded console** output (INFO=blue, WARN=yellow, ERROR=red)
- **Performance timing** for all HTTP requests: `<- 200 GET /health (65ms)`

Specialized Loggers

- `dev.mars.tinyrest.http` - Clean HTTP request/response logs
- `dev.mars.tinyrest.hotreload` - Configuration change monitoring
- `dev.mars.tinyrest.recorder` - Record/replay functionality
- `dev.mars.tinyrest.TinyRest` - Main application events

Log Files

- `logs/tinyrest.log` - Complete application logs with automatic rotation
- `logs/tinyrest-http.log` - Dedicated HTTP traffic logs
- **Daily rotation** with size limits and configurable retention

Example Output

```
20:33:33.129 [main] INFO  TinyRest$Engine - Engine configuration: templating=true, hotReload=true
20:33:33.144 [main] INFO  TinyRest$Engine - Initialized resource 'users' with 2 seed records
20:33:33.379 [pool-2-thread-1] INFO  http - -> GET /health
20:33:33.443 [pool-2-thread-1] INFO  http - <- 200 GET /health (65ms)
20:33:33.668 [pool-2-thread-4] WARN  http - <- 401 POST /api/users (54ms) - Missing/invalid bearer token
```

See LOGGING.md and LOGGING_EXAMPLES.md for complete documentation.

Templating expressions (when enabled)

In any **string** value of a static response you can use:

- `{{now}}` — ISO-instant timestamp
- `{{uuid}}` — random UUID
- `{{path.<name>}}` — path param (from `/api/things/{id}`)
- `{{query.<name>}}` — query param (from `?foo=bar`)
- `{{body.<dot.path>}}` — extract from JSON request body
- `{{header.<Name>}}` — request header
- `{{random.int(a,b)}}` — random int in `[a,b]`

Example:

```yaml
staticEndpoints:
  - method: GET
    path: /api/users/{id}/profile
    status: 200
    response:
      id: "{{path.id}}"
      corrId: "{{uuid}}"
      echo: "hi {{query.name}}, body says: {{body.note}}"
```

Record / Replay

```yaml
features:
  recordReplay:
    mode: record|replay|off
    file: target/tinyrest.recordings.jsonl
    replayOnMiss: fallback|error
```

- **record**: After a route produces a response, TinyRest appends a JSON object to the file.
- **replay**: On each request, TinyRest tries to match a recorded entry (method, path, query, opt headers/body). If `replayOnMiss: fallback`, it routes normally; if `error`, it returns **501** so you notice gaps.

> Matching knobs (`features.recordReplay.match`) are implemented in the `TinyRest.java` provided earlier. If you need body/header matching, add those keys in YAML accordingly.

CI Tips

- Always bind to a **random port** in CI (`port: 0`) and let the **JUnit extension** inject the base URL.
- Keep `tinyrest.yml` minimal and deterministic. If you use templating randomness, constrain it (e.g., `random.int(1,3)`).
- Persist `target/tinyrest.recordings.jsonl` as an artifact if you rely on replay.

Troubleshooting

- **401 on POST/PUT/DELETE** → you set `authToken` . Add:

```
Authorization: Bearer <token>
```

- **“Port already in use”** → set `port: 0` and consume the injected base URL in tests; for manual runs, pick a fixed port.
- **YAML edits not applied** → set `features.hotReload: true` or restart TinyRest.
- **Replay misses** → set `replayOnMiss: fallback` while iterating; switch to `error` to lock it down.

Design choices (and tradeoffs)

- Uses the JDK’s `com.sun.net.httpserver.HttpServer` . Not a servlet container — by design. Less magic, faster startup.
- Routing is simple regex over path segments. No annotations, no reflection.
- In-memory store is a `ConcurrentHashMap` . If you want persistence or relations, that’s a different product.
- Templating is intentionally minimal — no loops/ifs. It’s a test helper, not a view engine.

FAQ

Q: Can I host multiple independent resources?
A: Yes. Add more entries under `resources:` . Each gets CRUD under `/api/{name}` .

Q: Can I add custom logic per route?
A: Yes. You own `TinyRest.java` . Add a route, call into your code, return a `Response` .

Q: Does it support HTTPS?
A: Not out of the box. For tests, plain HTTP is enough. If you need TLS, wrap behind a test reverse proxy or extend the server.

Q: Does it work with virtual threads?
A: The JDK server uses a thread-per-request model. For test loads, that’s fine. If you want virtual threads, swap the executor or move to a server that supports it — but you probably don’t need it for this use case.

License

Pick whatever fits your org. MIT is typical for utility code like this.

Final word

TinyRest exists to **unblock testing**. It’s not a framework. If you’re fighting it, you’re solving the wrong problem — reach for a real service or WireMock. Otherwise, enjoy the speed and simplicity.