REST Assured API Automation Framework

pdfsent.com

Table of Contents

pdfsent.com

pdfsent.com

pdfsent.com

pdfsent.com

1. Introduction & Purpose

These notes document a complete, working API test automation framework built with **Java**, **REST Assured**, and **TestNG**. The goal of the framework is to demonstrate — in one coherent project — every HTTP method an API test engineer is expected to automate, every common request content type, and the supporting design patterns (POJOs, configuration management, reporting, schema validation) that separate a maintainable framework from a folder of ad-hoc scripts.

This document is written as a self-contained reference. Each chapter pairs the actual source code with a plain-language explanation of what the code does and, more importantly, *why* it is structured that way. It is meant to be read top to bottom by someone learning API automation, or used as a lookup reference by someone already building with REST Assured.

1.1 What problem does this framework solve?

Raw REST Assured calls work fine for a single test. The moment a suite grows past a handful of tests, three problems appear:

- **Repetition** — base URI, headers, and timeouts get copy-pasted into every test.
- **Fragile assertions** — string literals and raw JSON paths scattered everywhere break silently when an API changes shape.
- **No visibility** — console output is not something a non-technical stakeholder can read.

This framework addresses all three: a single `BaseTest` owns connection setup, POJOs give compile-time safety to request and response bodies, and ExtentReports turns a test run into a shareable HTML report.

1.2 Design philosophy

Principle	How it shows up in this framework
Single Responsibility	Config, endpoints, base setup, POJOs, and tests each live in their own package.
Don't Repeat Yourself	One `RequestSpecification` per target API, reused by every test class.
Fail with context	Request/response logging filters attach automatically, so a failing assertion always prints the full HTTP exchange.
Type safety over string paths	Wherever a response shape is known and stable, it is deserialized into a POJO instead of read with `.jsonPath().getString(...)`.
Externalized configuration	Base URIs, API keys, and timeouts live in `config.properties` and can be overridden from the command line — nothing is hardcoded in a test.

pdfsent.com

2. Technology Stack

Every dependency in this project earns its place. The table below lists what each one is for — useful both as a reference and as a checklist if you strip the framework down for a smaller project.

Library	Version	Role in this framework
Java	11	Language baseline — LTS, wide tooling support.
Maven	—	Build tool: dependency resolution, compilation, test execution.
REST Assured	5.4.0	Core HTTP DSL — `given().when().then()` syntax for building and asserting requests.
rest-assured json-schema-validator	5.4.0	Validates a JSON response body against a formal JSON Schema file.
rest-assured xml-path	5.4.0	Enables navigating XML responses with an XPath-like syntax.
TestNG	7.9.0	Test runner: annotations, DataProviders, parallel execution, suite XML.
Jackson Databind	2.16.1	Serializes POJOs to JSON request bodies and deserializes JSON responses back into POJOs.
Jackson Dataformat XML	2.16.1	Same job as Databind, for XML payloads.
Lombok	1.18.30	Generates getters, setters, constructors, and builders at compile time — removes POJO boilerplate.
ExtentReports	5.1.1	Produces a styled, shareable HTML report after every test run.
SLF4J Simple	2.0.9	Lightweight console logging backend.

2.1 Why REST Assured specifically?

REST Assured is a Java DSL purpose-built for testing HTTP APIs. Its signature `given() / when() / then()` structure mirrors the natural shape of an API test — *given* some setup, *when* I call an endpoint, *then* I expect this response — which keeps tests readable even to someone who has never seen the library before. It sits on top of Apache HttpClient, integrates natively with Hamcrest matchers for assertions, and has first-class support for JSON, XML, and schema validation without extra glue code.

pdfsent.com

3. Project Architecture

The project follows the standard Maven directory layout, with production code under `src/main` and test code under `src/test`. This separation matters: `src/main` holds the reusable framework (config, POJOs, base setup, reporting) that could ship as its own library, while `src/test` holds the actual test cases that consume it.

Folder structure

```
restassured-framework/
├── pom.xml
├── testng.xml
├── README.md
├── src/
│   ├── main/
│   │   ├── java/com/framework/
│   │   │   ├── config/
│   │   │   │   └── ConfigManager.java
│   │   │   ├── constants/
│   │   │   │   └── Endpoints.java
│   │   │   ├── base/
│   │   │   │   └── BaseTest.java
│   │   │   ├── reports/
│   │   │   │   ├── ExtentManager.java
│   │   │   │   └── ExtentTestListener.java
│   │   │   └── pojo/
│   │   │       ├── user/      (User, ListUsersResponse, SingleUserResponse,
│   │   │       │               CreateUserRequest/Response, UpdateUserRequest/Response,
Support)
│   │   │       ├── auth/      (RegisterRequest/Response, LoginRequest/Response,
ErrorResponse)
│   │   │       └── httpbin/   (HttpBinResponse)
│   │   └── resources/
│   │       └── config.properties
│   └── test/
│       ├── java/com/framework/tests/
│       │   ├── users/         (GetUsersTest, CreateUserTest, UpdateUserTest,
│       │   │                   PatchUserTest, DeleteUserTest)
│       │   ├── auth/          (AuthTest)
│       │   ├── contenttypes/  (ContentTypeTest)
│       │   └── methods/       (HttpMethodsTest)
│       └── resources/schemas/
│           └── single-user-schema.json
```

3.1 Package responsibilities

Package	Responsibility
`config`	Reads `config.properties` once, exposes typed getters, allows command-line override.
`constants`	Every URL path used by the suite, defined once.
`base`	Builds and shares the `RequestSpecification` objects every test uses.

pdfsent.com

`reports`	Wires TestNG's listener hooks into ExtentReports for HTML output.
`pojo`	Java classes that mirror JSON/XML request and response shapes.
`tests`	The actual TestNG test classes, grouped by concern (CRUD, auth, content type, HTTP method).

pdfsent.com

pdfsent.com

4. Maven Build Configuration — pom.xml

The POM declares every dependency from Chapter 2 plus two plugins: the compiler plugin (configured to run Lombok's annotation processor) and the Surefire plugin (configured to execute `testng.xml` instead of Maven's default test discovery).

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.framework</groupId>
    <artifactId>restassured-framework</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>
    <name>REST Assured API Automation Framework</name>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <restassured.version>5.4.0</restassured.version>
        <testng.version>7.9.0</testng.version>
        <jackson.version>2.16.1</jackson.version>
        <json-schema-validator.version>5.4.0</json-schema-validator.version>
        <extentreports.version>5.1.1</extentreports.version>
        <slf4j.version>2.0.9</slf4j.version>
        <lombok.version>1.18.30</lombok.version>
    </properties>

    <dependencies>
        <!-- REST Assured core + JSON/XML path + schema validation -->
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>rest-assured</artifactId>
            <version>${restassured.version}</version>
        </dependency>
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>json-schema-validator</artifactId>
            <version>${json-schema-validator.version}</version>
        </dependency>
        <dependency>
            <groupId>io.rest-assured</groupId>
            <artifactId>xml-path</artifactId>
            <version>${restassured.version}</version>
        </dependency>

        <!-- Test runner -->
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>${testng.version}</version>
        </dependency>
```


pdfsent.com

```
        <!-- JSON <-> POJO serialization -->
        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.version}</version>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
            <version>${jackson.version}</version>
        </dependency>

        <!-- HTML reporting -->
        <dependency>
            <groupId>com.aventstack</groupId>
            <artifactId>extentreports</artifactId>
            <version>${extentreports.version}</version>
        </dependency>

        <!-- Logging -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>

        <!-- Boilerplate reduction for POJOs -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>${lombok.version}</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <finalName>restassured-framework</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.12.1</version>
                <configuration>
                    <source>11</source>
                    <target>11</target>
                    <annotationProcessorPaths>
                        <path>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                            <version>${lombok.version}</version>
                        </path>
                    </annotationProcessorPaths>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.2.5</version>
                <configuration>
                    <suiteXmlFiles>
                        <suiteXmlFile>testng.xml</suiteXmlFile>
                    </suiteXmlFiles>
                </configuration>
            </plugin>
```


pdfsent.com

```
        </plugins>
    </build>
</project>
```

4.1 Key configuration choices

- **Lombok annotation processor path** — declared explicitly under `maven-compiler-plugin` so Lombok's `@Data` / `@Builder` annotations are expanded at compile time. Without this, Lombok-annotated POJOs would compile with missing getters/setters.
- **Surefire → testng.xml** — without this, Surefire falls back to its own class-discovery rules and ignores the listener and parallel-execution settings defined in the suite file.
- **Centralized versions** — every dependency version is a `<properties>` entry, so upgrading REST Assured is a one-line change.

pdfsent.com

pdfsent.com

5. Configuration Management

Nothing environment-specific is hardcoded inside a test. Base URIs, API keys, and timeouts all live in one properties file, and every value can be overridden at runtime with a `-D` system property — which is what makes this framework CI-friendly without editing files before every run.

src/main/resources/config.properties

```
# Base URIs for the two demo services used by this framework.
# reqres.in   -> JSON REST API demo, used for POJO-driven CRUD tests (GET/POST/PUT/PATCH/DELETE)
# httpbin.org -> echo service, used to demonstrate XML / form-urlencoded / multipart / HEAD /
OPTIONS
reqres.base.uri=https://reqres.in/api
httpbin.base.uri=https://httpbin.org

# reqres.in requires a free API key as of 2024 (header x-api-key). Get one at https://reqres.in
# and override it here, or pass -Dapi.key=xxxx on the command line.
api.key=reqres-free-v1

connection.timeout=15000
socket.timeout=15000
retry.count=1
```

src/main/java/com/framework/config/ConfigManager.java

```
package com.framework.config;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

/**
 * Loads src/main/resources/config.properties once and exposes typed getters.
 * Any property can be overridden at runtime with -Dkey=value (useful in CI).
 */
public final class ConfigManager {

    private static final ConfigManager INSTANCE = new ConfigManager();
    private final Properties properties = new Properties();

    private ConfigManager() {
        try (InputStream in =
getClass().getClassLoader().getResourceAsStream("config.properties")) {
            if (in == null) {
                throw new RuntimeException("config.properties not found on classpath");
            }
            properties.load(in);
        } catch (IOException e) {
            throw new RuntimeException("Failed to load config.properties", e);
        }
    }

    public static ConfigManager getInstance() {
        return INSTANCE;
    }

    private String get(String key) {
        // System property wins so CI/CLI overrides never require editing the file
        return System.getProperty(key, properties.getProperty(key));
```


pdfsent.com

```
    }

    public String reqresBaseUri() {
        return get("reqres.base.uri");
    }

    public String httpbinBaseUri() {
        return get("httpbin.base.uri");
    }

    public String apiKey() {
        return get("api.key");
    }

    public int connectionTimeout() {
        return Integer.parseInt(get("connection.timeout"));
    }

    public int socketTimeout() {
        return Integer.parseInt(get("socket.timeout"));
    }
}
```

5.1 Why a singleton?

`ConfigManager` reads the properties file exactly once, the first time `getInstance()` is called, and caches it in a private static field. Every test class asks the same singleton for configuration, so there is one source of truth and no repeated file I/O during a test run.

5.2 Why `System.getProperty(key, ...)` as a fallback?

Each getter checks a JVM system property first and only falls back to the file value if that property was never set. This is what allows `mvn test -Dapi.key=YOUR_KEY` to override the file without editing it — essential for CI pipelines where secrets are injected as environment/system properties, not committed to source control.

pdfsent.com

6. Endpoint Constants

Every URL path the suite calls is declared once, here, and referenced by constant name in tests. If an endpoint path changes, it changes in one place instead of in every test file that happens to call it.

src/main/java/com/framework/constants/Endpoints.java

```
package com.framework.constants;

/**
 * Centralised endpoint paths. Keeping these out of test classes means a path change
 * is a one-line fix instead of a find-and-replace across the suite.
 */
public final class Endpoints {

    private Endpoints() {
    }

    // ---- reqres.in (JSON CRUD demo) ----
    public static final String USERS = "/users";
    public static final String USER_BY_ID = "/users/{id}";
    public static final String REGISTER = "/register";
    public static final String LOGIN = "/login";

    // ---- httpbin.org (protocol / content-type demo) ----
    public static final String HTTPBIN_GET = "/get";
    public static final String HTTPBIN_POST = "/post";
    public static final String HTTPBIN_PUT = "/put";
    public static final String HTTPBIN_PATCH = "/patch";
    public static final String HTTPBIN_DELETE = "/delete";
    public static final String HTTPBIN_HEAD = "/get";       // HEAD hits the same route as GET
    public static final String HTTPBIN_OPTIONS = "/get";    // OPTIONS is method-agnostic on
httpbin
    public static final String HTTPBIN_XML = "/xml";
    public static final String HTTPBIN_STATUS = "/status/{code}";
}
```

6.1 Path parameters

Notice paths like `/users/{id}` use REST Assured's own path-parameter placeholder syntax. A test supplies the actual value with `.pathParam("id", 2)`, and REST Assured substitutes it before the request is sent — no manual string concatenation, no risk of a malformed URL.

pdfsent.com

7. Base Test Design & Request Specifications

Every test class extends `BaseTest`, which is responsible for exactly one thing: building and sharing the `RequestSpecification` each test needs. A `RequestSpecification` is REST Assured's reusable bundle of base URI, default headers, content type, and timeout configuration — build it once, reuse it on every call with `given().spec(...)`.

src/main/java/com/framework/base/BaseTest.java

```
package com.framework.base;

import com.framework.config.ConfigManager;
import io.restassured.builder.RequestSpecBuilder;
import io.restassured.config.RestAssuredConfig;
import io.restassured.filter.log.RequestLoggingFilter;
import io.restassured.filter.log.ResponseLoggingFilter;
import io.restassured.http.ContentType;
import io.restassured.specification.RequestSpecification;

import static io.restassured.config.HttpClientConfig.httpClientConfig;

/**
 * Every test class extends this. It exposes two ready-to-use RequestSpecifications:
 *  - reqresSpec()   -> talks to reqres.in, sends/receives JSON, POJO CRUD demo
 *  - httpbinSpec()  -> talks to httpbin.org, used for content-type / HTTP-method demos
 *
 * Specs are built once in a static, thread-safe holder (not per-instance @BeforeSuite)
 * because testng.xml runs test classes in parallel — an instance-level @BeforeSuite would
 * only populate whichever single instance TestNG happens to invoke it on, leaving every
 * other test class's fields null. Static + synchronized lazy init avoids that entirely.
 *
 * Request/response logging is attached to every call so failures print full
 * request + response detail without needing .log().all() sprinkled in every test.
 */
public class BaseTest {

    protected static final ConfigManager config = ConfigManager.getInstance();

    private static volatile RequestSpecification reqresRequestSpec;
    private static volatile RequestSpecification httpbinRequestSpec;

    protected synchronized RequestSpecification reqresSpec() {
        if (reqresRequestSpec == null) {
            reqresRequestSpec = new RequestSpecBuilder()
                    .setBaseUri(config.reqresBaseUri())
                    .setContentType(ContentType.JSON)
                    .addHeader("x-api-key", config.apiKey())
                    .setConfig(buildTimeoutConfig())
                    .addFilter(new RequestLoggingFilter())
                    .addFilter(new ResponseLoggingFilter())
                    .build();
        }
        return reqresRequestSpec;
    }

    protected synchronized RequestSpecification httpbinSpec() {
        if (httpbinRequestSpec == null) {
            httpbinRequestSpec = new RequestSpecBuilder()
                    .setBaseUri(config.httpbinBaseUri())
```


pdfsent.com

```
                    .setConfig(buildTimeoutConfig())
                    .addFilter(new RequestLoggingFilter())
                    .addFilter(new ResponseLoggingFilter())
                    .build();
        }
        return httpbinRequestSpec;
    }

    private static RestAssuredConfig buildTimeoutConfig() {
        return RestAssuredConfig.config().httpClient(
                httpClientConfig()
                        .setParam("http.connection.timeout", config.connectionTimeout())
                        .setParam("http.socket.timeout", config.socketTimeout())
        );
    }
}
```

7.1 A real bug, and why the fix matters

The first version of this class populated the specs inside a TestNG `@BeforeSuite` method, storing them in ordinary (non-static) instance fields. That looks reasonable until you remember two facts about TestNG:

- TestNG creates a **separate instance** of the test class for every `<class>` entry that extends it.
- `@BeforeSuite` runs **exactly once** for the whole suite, on whichever single instance TestNG happens to pick.

The consequence: only the one instance TestNG chose to run `@BeforeSuite` on would have a populated `reqresRequestSpec` field. Every other test class — `CreateUserTest`, `AuthTest`, and so on — would call `reqresSpec()` on its own separate instance, find the field still `null`, and throw a `NullPointerException` the moment it tried to use it. This is especially easy to miss locally if tests happen to run sequentially in an order where it doesn't surface, and then fails intermittently once `parallel="classes"` is enabled in `testng.xml`.

The fix: make the fields `static` so they belong to the class, not to any one instance, and build them lazily inside a `synchronized` accessor method instead of depending on a TestNG lifecycle hook at all. Every instance, on every thread, now reads or builds the same static spec exactly once. This is a textbook example of why shared framework state should never live in per-instance fields when the test runner may create more than one instance of the class that owns it.

7.2 What goes into each spec

Setting	reqresSpec()	httpbinSpec()
Base URI	https://reqres.in/api	https://httpbin.org
Default content type	application/json	none (set per-test)
Default headers	`x-api-key`	none
Logging filters	Request + Response	Request + Response
Timeouts	from config.properties	from config.properties

pdfsent.com

8. The POJO Layer

POJO stands for Plain Old Java Object. In this framework, every request body sent and every response body expected has a matching POJO. This trades a small amount of upfront typing for a large amount of safety: a typo in a field name becomes a compile error instead of a silent `null` at runtime, and refactoring an endpoint's shape is a one-file change instead of a search across every test that reads that field with a raw JSON path.

8.1 The annotations that make this work

Annotation	Source	Effect
`@Data`	Lombok	Generates getters, setters, `equals()`, `hashCode()`, and `toString`
`@NoArgsConstructor`	Lombok	Generates an empty constructor — required by Jackson to instantiate the ob before populating fields.
`@AllArgsConstructor`	Lombok	Generates a constructor taking every field, used alongside `@Builder`.
`@Builder`	Lombok	Generates a fluent builder, e.g. `CreateUserRequest.builder().name("x").job("y").build`
`@JsonProperty("first_name")`	Jackson	Maps a snake_case JSON field to a camelCase Java field.
`@JsonIgnoreProperties(ignoreUnknown = true)`	Jackson	Prevents deserialization from failing if the API adds a field the POJO doesn know about yet.

8.2 User domain POJOs

These model the `reqres.in` user resource: a plain `User`, the paginated list wrapper, the single-user wrapper, and the request/response pairs for create and update.

src/main/java/com/framework/pojo/user/User.java

```
package com.framework.pojo.user;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Maps a single user object returned by GET /users/{id} and inside the list response. */
@Data
@NoArgsConstructor
@AllArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class User {
    private int id;
    private String email;

    @JsonProperty("first_name")
    private String firstName;

    @JsonProperty("last_name")
```


pdfsent.com

```
    private String lastName;

    private String avatar;
}
```

src/main/java/com/framework/pojo/user/Support.java

```
package com.framework.pojo.user;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/** The "support" footer object reqres.in adds to list/single responses. */
@Data
@NoArgsConstructor
@AllArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class Support {
    private String url;
    private String text;
}
```

src/main/java/com/framework/pojo/user/ListUsersResponse.java

```
package com.framework.pojo.user;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

/** Full response body of GET /users?page=n */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class ListUsersResponse {
    private int page;

    @JsonProperty("per_page")
    private int perPage;

    private int total;

    @JsonProperty("total_pages")
    private int totalPages;

    private List<User> data;
    private Support support;
}
```

src/main/java/com/framework/pojo/user/SingleUserResponse.java

```
package com.framework.pojo.user;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
```


pdfsent.com

```
import lombok.NoArgsConstructor;

/** Response body of GET /users/{id} — reqres wraps the User inside a "data" key. */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class SingleUserResponse {
    private User data;
    private Support support;
}
```

src/main/java/com/framework/pojo/user/CreateUserRequest.java

```
package com.framework.pojo.user;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Request body for POST /users */
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class CreateUserRequest {
    private String name;
    private String job;
}
```

src/main/java/com/framework/pojo/user/CreateUserResponse.java

```
package com.framework.pojo.user;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Response body of POST /users */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class CreateUserResponse {
    private String name;
    private String job;
    private String id;
    private String createdAt;
}
```

src/main/java/com/framework/pojo/user/UpdateUserRequest.java

```
package com.framework.pojo.user;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Request body shared by PUT and PATCH /users/{id} */
@Data
```


pdfsent.com

```
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class UpdateUserRequest {
    private String name;
    private String job;
}
```

src/main/java/com/framework/pojo/user/UpdateUserResponse.java

```
package com.framework.pojo.user;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Response body of PUT/PATCH /users/{id} */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class UpdateUserResponse {
    private String name;
    private String job;
    private String updatedAt;
}
```

8.3 Authentication POJOs

Register and login share the same request shape (email + password) but are kept as separate classes rather than reused, because in a real API they frequently diverge over time — register might later need a name or terms-accepted flag that login never will. `ErrorResponse` models the `{"error": "..."}` shape the API returns on a 400.

src/main/java/com/framework/pojo/auth/RegisterRequest.java

```
package com.framework.pojo.auth;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Request body for POST /register */
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class RegisterRequest {
    private String email;
    private String password;
}
```

src/main/java/com/framework/pojo/auth/RegisterResponse.java

```
package com.framework.pojo.auth;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
```


pdfsent.com

```
import lombok.NoArgsConstructor;

/** Success response body of POST /register */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class RegisterResponse {
    private int id;
    private String token;
}
```

src/main/java/com/framework/pojo/auth/LoginRequest.java

```
package com.framework.pojo.auth;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Request body for POST /login */
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class LoginRequest {
    private String email;
    private String password;
}
```

src/main/java/com/framework/pojo/auth/LoginResponse.java

```
package com.framework.pojo.auth;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import lombok.NoArgsConstructor;

/** Success response body of POST /login */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class LoginResponse {
    private String token;
}
```

src/main/java/com/framework/pojo/auth/ErrorResponse.java

```
package com.framework.pojo.auth;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import lombok.NoArgsConstructor;

/** reqres.in returns {"error": "..."} for 4xx auth failures. */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class ErrorResponse {
```


pdfsent.com

```
    private String error;
}
```

8.4 Generic echo POJO

`HttpBinResponse` is intentionally loose — it maps httpbin.org's envelope (`args`, `data`, `files`, `form`, `headers`, `json`, `url`) using `Map<String, Object>` fields rather than strict types, because this one POJO is reused across every content-type test in Chapter 16 and the shape of its `json` / `form` / `files` contents changes per test.

`src/main/java/com/framework/pojo/httpbin/HttpBinResponse.java`

```
package com.framework.pojo.httpbin;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Map;

/**
 * Generic mapping of httpbin.org's echo response shape, used across
 * GET/POST/PUT/PATCH/DELETE and all content-type demos since httpbin
 * always echoes the request back in this envelope.
 */
@Data
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
public class HttpBinResponse {
    private Map<String, Object> args;
    private Object data;
    private Map<String, Object> files;
    private Map<String, Object> form;
    private Map<String, Object> headers;
    private Map<String, Object> json;
    private String url;
    private String origin;
}
```

pdfsent.com

pdfsent.com

9. HTTP GET — Retrieval, Query & Path Params, Schema Validation

GET is exercised four ways in one class: a paginated list read with a query parameter, a single resource read with a path parameter validated against a formal JSON Schema, the same single-resource call repeated across a `@DataProvider` of ids to prove the deserialization holds for more than one record, and a negative case confirming a non-existent id returns 404 with an empty body.

src/test/java/com/framework/tests/users/GetUsersTest.java

```
package com.framework.tests.users;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import com.framework.pojo.user.ListUsersResponse;
import com.framework.pojo.user.SingleUserResponse;
import io.restassured.response.Response;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;
import static io.restassured.module.jsv.JsonSchemaValidator.matchesJsonSchemaInClasspath;
import static org.hamcrest.Matchers.*;
import static org.testng.Assert.*;

/**
 * GET method demo:
 *  - list endpoint with a query param (?page=n), deserialized straight into a POJO
 *  - single-resource endpoint with a path param, validated against a JSON schema
 *  - negative case: 404 for a resource that does not exist
 */
public class GetUsersTest extends BaseTest {

    @Test(description = "GET /users?page=2 returns a paginated list mapped to ListUsersResponse
POJO")
    public void getUsersList_returnsPopulatedPage() {
        ListUsersResponse response = given()
                .spec(reqresSpec())
                .queryParam("page", 2)
                .when()
                .get(Endpoints.USERS)
                .then()
                .statusCode(200)
                .body("page", equalTo(2))
                .extract().as(ListUsersResponse.class);

        assertEquals(response.getPage(), 2);
        assertFalse(response.getData().isEmpty(), "Expected at least one user on page 2");
        response.getData().forEach(user -> {
            assertTrue(user.getId() > 0);
            assertNotNull(user.getEmail());
        });
    }

    @Test(description = "GET /users/{id} for a known id returns 200 and matches the JSON
schema")
    public void getSingleUser_validSchema() {
        given()
```


pdfsent.com

```
                .spec(reqresSpec())
                .pathParam("id", 2)
                .when()
                .get(Endpoints.USER_BY_ID)
                .then()
                .statusCode(200)
                .body(matchesJsonSchemaInClasspath("schemas/single-user-schema.json"));
    }

    @Test(dataProvider = "validUserIds", description = "GET /users/{id} deserializes into
SingleUserResponse for several ids")
    public void getSingleUser_deserializesToPojo(int userId) {
        SingleUserResponse response = given()
                .spec(reqresSpec())
                .pathParam("id", userId)
                .when()
                .get(Endpoints.USER_BY_ID)
                .then()
                .statusCode(200)
                .extract().as(SingleUserResponse.class);

        assertEquals(response.getData().getId(), userId);
        assertNotNull(response.getData().getEmail());
    }

    @DataProvider(name = "validUserIds")
    public Object[][] validUserIds() {
        return new Object[][]{{2}, {3}, {5}};
    }

    @Test(description = "GET /users/{id} for a non-existent id returns 404 with an empty body")
    public void getSingleUser_notFound_returns404() {
        Response response = given()
                .spec(reqresSpec())
                .pathParam("id", 9999)
                .when()
                .get(Endpoints.USER_BY_ID)
                .then()
                .statusCode(404)
                .extract().response();

        assertEquals(response.jsonPath().getMap("$").size(), 0, "404 body should be an empty
object");
    }
}
```

9.1 Reading the assertions

- `.body("page", equalTo(2))` asserts on a single JSON field without leaving the fluent chain.
- `.extract().as(ListUsersResponse.class)` hands the whole response to Jackson and gets a typed POJO back for further, ordinary Java assertions.
- `matchesJsonSchemaInClasspath(...)` is a stronger check than field-by-field assertions: it fails if the API adds, removes, or retypes any field the schema declares.

pdfsent.com

10. HTTP POST — Resource Creation

POST demonstrates the full round trip: a `CreateUserRequest` POJO is passed directly as `.body(requestBody)` — REST Assured serializes it to JSON automatically because the spec's default content type is `application/json` — and the 201 response is deserialized straight back into a `CreateUserResponse` POJO for assertions.

src/test/java/com/framework/tests/users/CreateUserTest.java

```
package com.framework.tests.users;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import com.framework.pojo.user.CreateUserRequest;
import com.framework.pojo.user.CreateUserResponse;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.notNullValue;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertNotNull;

/** POST method demo: send a POJO as the JSON body, get a POJO back. */
public class CreateUserTest extends BaseTest {

    @Test(description = "POST /users creates a resource; request and response are both POJOs")
    public void createUser_returns201AndEchoesFields() {
        CreateUserRequest requestBody = CreateUserRequest.builder()
                .name("Shammi Bharti")
                .job("Bluetooth Qualification Consultant")
                .build();

        CreateUserResponse response = given()
                .spec(reqresSpec())
                .body(requestBody)
                .when()
                .post(Endpoints.USERS)
                .then()
                .statusCode(201)
                .body("id", notNullValue())
                .body("createdAt", notNullValue())
                .extract().as(CreateUserResponse.class);

        assertEquals(response.getName(), requestBody.getName());
        assertEquals(response.getJob(), requestBody.getJob());
        assertNotNull(response.getId());
    }

    @Test(description = "POST /users with a missing job field still succeeds; reqres does not
enforce required fields")
    public void createUser_partialBody_stillCreated() {
        CreateUserRequest requestBody = CreateUserRequest.builder()
                .name("Partial User")
                .build();

        given()
                .spec(reqresSpec())
                .body(requestBody)
                .when()
```


pdfsent.com

```
                .post(Endpoints.USERS)
                .then()
                .statusCode(201)
                .body("name", org.hamcrest.Matchers.equalTo("Partial User"));
    }
}
```

pdfsent.com

pdfsent.com

11. HTTP PUT — Full Update

PUT represents a full resource replacement. The test sends both fields of `UpdateUserRequest` and asserts the response echoes them back along with a fresh `updatedAt` timestamp.

src/test/java/com/framework/tests/users/UpdateUserTest.java

```
package com.framework.tests.users;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import com.framework.pojo.user.UpdateUserRequest;
import com.framework.pojo.user.UpdateUserResponse;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertNotNull;

/** PUT method demo: full resource replacement. */
public class UpdateUserTest extends BaseTest {

    @Test(description = "PUT /users/{id} fully replaces the resource and returns updatedAt")
    public void updateUser_put_returns200() {
        UpdateUserRequest requestBody = UpdateUserRequest.builder()
                .name("Shammi Bharti")
                .job("Senior BQC")
                .build();

        UpdateUserResponse response = given()
                .spec(reqresSpec())
                .pathParam("id", 2)
                .body(requestBody)
                .when()
                .put(Endpoints.USER_BY_ID)
                .then()
                .statusCode(200)
                .extract().as(UpdateUserResponse.class);

        assertEquals(response.getName(), requestBody.getName());
        assertEquals(response.getJob(), requestBody.getJob());
        assertNotNull(response.getUpdatedAt());
    }
}
```

pdfsent.com

pdfsent.com

12. HTTP PATCH — Partial Update

PATCH is deliberately shown sending only one field (`job`) via the same `UpdateUserRequest` builder, to make the PUT vs. PATCH distinction concrete: PUT implies the full resource, PATCH implies a partial one, even though both hit the same demo API in practice.

src/test/java/com/framework/tests/users/PatchUserTest.java

```
package com.framework.tests.users;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import com.framework.pojo.user.UpdateUserRequest;
import com.framework.pojo.user.UpdateUserResponse;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertNotNull;

/** PATCH method demo: partial resource update — only the job field is sent. */
public class PatchUserTest extends BaseTest {

    @Test(description = "PATCH /users/{id} partially updates the resource")
    public void patchUser_partialUpdate_returns200() {
        UpdateUserRequest requestBody = UpdateUserRequest.builder()
                .job("Lead BQC")
                .build();

        UpdateUserResponse response = given()
                .spec(reqresSpec())
                .pathParam("id", 2)
                .body(requestBody)
                .when()
                .patch(Endpoints.USER_BY_ID)
                .then()
                .statusCode(200)
                .extract().as(UpdateUserResponse.class);

        assertEquals(response.getJob(), "Lead BQC");
        assertNotNull(response.getUpdatedAt());
    }
}
```


pdfsent.com

13. HTTP DELETE — Resource Removal

DELETE is the simplest case in the suite: no request body, and the only meaningful assertions are the status code (204 No Content) and that the body is genuinely empty.

src/test/java/com/framework/tests/users/DeleteUserTest.java

```
package com.framework.tests.users;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;

/** DELETE method demo: reqres.in returns 204 No Content with an empty body. */
public class DeleteUserTest extends BaseTest {

    @Test(description = "DELETE /users/{id} returns 204 with no body")
    public void deleteUser_returns204() {
        given()
                .spec(reqresSpec())
                .pathParam("id", 2)
                .when()
                .delete(Endpoints.USER_BY_ID)
                .then()
                .statusCode(204)
                .body(org.hamcrest.Matchers.emptyOrNullString());
    }
}
```

pdfsent.com

pdfsent.com

14. HEAD, OPTIONS & Status Code Matrix

These two methods are rarely covered in tutorials but come up in real audits: HEAD confirms headers and status without transferring a body, OPTIONS confirms which methods a server advertises as supported via the `Allow` header. A parameterized status-code test rounds out method coverage by proving negative-path assertions work identically across a whole range of codes, using httpbin's `/status/{code}` endpoint which returns whatever code you ask it for.

src/test/java/com/framework/tests/methods/HttpMethodsTest.java

```
package com.framework.tests.methods;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;

/**
 * Rounds out method coverage that the CRUD-focused user tests don't exercise directly:
 * HEAD, OPTIONS, and forcing arbitrary status codes to prove negative-path assertions work.
 */
public class HttpMethodsTest extends BaseTest {

    @Test(description = "HEAD request returns headers only, with no body, and status 200")
    public void headRequest_returns200NoBody() {
        given()
                .spec(httpbinSpec())
                .when()
                .head(Endpoints.HTTPBIN_HEAD)
                .then()
                .statusCode(200)
                .body(org.hamcrest.Matchers.emptyOrNullString());
    }

    @Test(description = "OPTIONS request returns the Allow header listing supported methods")
    public void optionsRequest_returnsAllowHeader() {
        given()
                .spec(httpbinSpec())
                .when()
                .options(Endpoints.HTTPBIN_OPTIONS)
                .then()
                .statusCode(200)
                .header("Allow", org.hamcrest.Matchers.notNullValue());
    }

    @Test(dataProvider = "statusCodes", description = "httpbin /status/{code} echoes back
whatever status is requested")
    public void statusEndpoint_returnsRequestedCode(int code) {
        given()
                .spec(httpbinSpec())
                .pathParam("code", code)
                .when()
                .get(Endpoints.HTTPBIN_STATUS)
                .then()
                .statusCode(code);
    }
```


pdfsent.com

```
    @DataProvider(name = "statusCodes")
    public Object[][] statusCodes() {
        return new Object[][]{{200}, {201}, {400}, {401}, {403}, {404}, {500}, {503}};
    }
}
```

pdfsent.com

pdfsent.com

15. Authentication Flows

Auth is modeled as two independent flows — register and login — each with a positive case (valid credentials, 200, token returned) and a negative case (missing password, 400, structured error message). reqres.in only accepts a fixed demo email/password pair for these endpoints, which is exactly what makes it useful here: the positive and negative paths are both deterministic and repeatable.

src/test/java/com/framework/tests/auth/AuthTest.java

```
package com.framework.tests.auth;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import com.framework.pojo.auth.ErrorResponse;
import com.framework.pojo.auth.LoginRequest;
import com.framework.pojo.auth.LoginResponse;
import com.framework.pojo.auth.RegisterRequest;
import com.framework.pojo.auth.RegisterResponse;
import org.testng.annotations.Test;

import static io.restassured.RestAssured.given;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertNotNull;

/**
 * Auth flow demo — reqres.in only accepts a fixed set of emails for register/login
 * (see https://reqres.in/api-docs), so this doubles as a positive/negative example.
 */
public class AuthTest extends BaseTest {

    private static final String VALID_EMAIL = "eve.holt@reqres.in";
    private static final String VALID_PASSWORD = "pistol";

    @Test(description = "POST /register with a known-good email succeeds and returns a token")
    public void register_validUser_returns200() {
        RegisterRequest requestBody = RegisterRequest.builder()
                .email(VALID_EMAIL)
                .password(VALID_PASSWORD)
                .build();

        RegisterResponse response = given()
                .spec(reqresSpec())
                .body(requestBody)
                .when()
                .post(Endpoints.REGISTER)
                .then()
                .statusCode(200)
                .extract().as(RegisterResponse.class);

        assertNotNull(response.getToken());
        assertEquals(response.getId(), 4);
    }

    @Test(description = "POST /register without a password fails with 400 and an error message")
    public void register_missingPassword_returns400() {
        RegisterRequest requestBody = RegisterRequest.builder()
                .email(VALID_EMAIL)
                .build();
```


pdfsent.com

```
        ErrorResponse response = given()
                .spec(reqresSpec())
                .body(requestBody)
                .when()
                .post(Endpoints.REGISTER)
                .then()
                .statusCode(400)
                .extract().as(ErrorResponse.class);

        assertEquals(response.getError(), "Missing password");
    }

    @Test(description = "POST /login with valid credentials returns a token")
    public void login_validUser_returns200() {
        LoginRequest requestBody = LoginRequest.builder()
                .email(VALID_EMAIL)
                .password(VALID_PASSWORD)
                .build();

        LoginResponse response = given()
                .spec(reqresSpec())
                .body(requestBody)
                .when()
                .post(Endpoints.LOGIN)
                .then()
                .statusCode(200)
                .extract().as(LoginResponse.class);

        assertNotNull(response.getToken());
    }

    @Test(description = "POST /login without a password fails with 400 and an error message")
    public void login_missingPassword_returns400() {
        LoginRequest requestBody = LoginRequest.builder()
                .email(VALID_EMAIL)
                .build();

        given()
                .spec(reqresSpec())
                .body(requestBody)
                .when()
                .post(Endpoints.LOGIN)
                .then()
                .statusCode(400)
                .body("error", org.hamcrest.Matchers.equalTo("Missing password"));
    }
}
```

pdfsent.com

pdfsent.com

16. Content Type Coverage — JSON, XML, Form, Multipart

This is the chapter that most REST Assured tutorials skip. Real APIs are not all JSON: file upload endpoints expect `multipart/form-data`, legacy or web-form-backed endpoints expect `application/x-www-form-urlencoded`, and plenty of enterprise and SOAP-adjacent systems still speak `application/xml`. This class proves the framework handles all four correctly, using httpbin.org's echo behaviour as the oracle: whatever it receives, it reflects back in its JSON response envelope, which makes it possible to assert on exactly what was sent over the wire.

src/test/java/com/framework/tests/contenttypes/ContentTypeTest.java

```
package com.framework.tests.contenttypes;

import com.framework.base.BaseTest;
import com.framework.constants.Endpoints;
import com.framework.pojo.httpbin.HttpBinResponse;
import com.framework.pojo.user.CreateUserRequest;
import io.restassured.http.ContentType;
import org.testng.annotations.Test;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;

import static io.restassured.RestAssured.given;
import static org.hamcrest.Matchers.equalTo;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertTrue;

/**
 * Content-type demo. httpbin.org echoes back exactly what it received, which makes it
 * ideal for proving the framework can send every common body encoding correctly:
 *   - application/json           (POJO auto-serialized by REST Assured + Jackson)
 *   - application/xml            (raw XML string body)
 *   - application/x-www-form-urlencoded (form fields)
 *   - multipart/form-data        (file upload + form fields together)
 */
public class ContentTypeTest extends BaseTest {

    @Test(description = "application/json body: a POJO is serialized automatically and echoed
back under 'json'")
    public void postJson_pojoBody_isEchoedBack() {
        CreateUserRequest body = CreateUserRequest.builder()
                .name("Shammi")
                .job("BQC")
                .build();

        HttpBinResponse response = given()
                .spec(httpbinSpec())
                .contentType(ContentType.JSON)
                .body(body)
                .when()
                .post(Endpoints.HTTPBIN_POST)
                .then()
                .statusCode(200)
                .extract().as(HttpBinResponse.class);
```


pdfsent.com

```
        assertEquals(response.getJson().get("name"), "Shammi");
        assertEquals(response.getJson().get("job"), "BQC");
    }

    @Test(description = "application/xml body: a raw XML string is sent and echoed back under
'data'")
    public void postXml_rawBody_isEchoedBack() {
        String xmlBody = "<user><name>Shammi</name><role>BQC</role></user>";

        given()
                .spec(httpbinSpec())
                .contentType(ContentType.XML)
                .body(xmlBody)
                .when()
                .post(Endpoints.HTTPBIN_POST)
                .then()
                .statusCode(200)
                .body("data", equalTo(xmlBody))
                .body("headers.'Content-Type'", equalTo("application/xml; charset=UTF-8"));
    }

    @Test(description = "GET /xml: response content-type is XML and can be navigated with
xmlPath")
    public void getXml_responseParsedWithXmlPath() {
        String title = given()
                .spec(httpbinSpec())
                .when()
                .get(Endpoints.HTTPBIN_XML)
                .then()
                .statusCode(200)
                .contentType(ContentType.XML)
                .extract()
                .xmlPath()
                .getString("slideshow.@title");

        assertTrue(title != null && !title.isEmpty(), "Expected a non-empty slideshow title
attribute");
    }

    @Test(description = "application/x-www-form-urlencoded: form fields are sent and echoed back
under 'form'")
    public void postFormUrlEncoded_fieldsAreEchoedBack() {
        given()
                .spec(httpbinSpec())
                .contentType(ContentType.URLENC)
                .formParam("name", "Shammi Bharti")
                .formParam("role", "Bluetooth Qualification Consultant")
                .when()
                .post(Endpoints.HTTPBIN_POST)
                .then()
                .statusCode(200)
                .body("form.name", equalTo("Shammi Bharti"))
                .body("form.role", equalTo("Bluetooth Qualification Consultant"));
    }

    @Test(description = "multipart/form-data: a file plus form fields are sent together and
echoed back")
    public void postMultipart_fileAndFieldsAreEchoedBack() throws IOException {
        File tempFile = File.createTempFile("upload-demo", ".txt");
        Files.write(tempFile.toPath(), "Bluetooth qualification test evidence".getBytes());
        tempFile.deleteOnExit();

        given()
```

pdfsent.com

pdfsent.com

```
                .spec(httpbinSpec())
                .contentType(ContentType.MULTIPART)
                .multiPart("file", tempFile, "text/plain")
                .multiPart("description", "IOP test log")
                .when()
                .post(Endpoints.HTTPBIN_POST)
                .then()
                .statusCode(200)
                .body("files.file", equalTo("Bluetooth qualification test evidence"))
                .body("form.description", equalTo("IOP test log"));
    }
}
```

16.1 What each test actually proves

Test	Content-Type sent	What is proven
`postJson_pojoBody_isEchoedBack`	application/json	A POJO passed to `.body()` is auto-serialized to JSON without any manual `ObjectMapper` call.
`postXml_rawBody_isEchoedBack`	application/xml	A raw XML string is sent with the correct content-type header and arrives byte-for-byte intact.
`getXml_responseParsedWithXmlPath`	— (response only)	An XML response can be navigated with `xmlPath()` the same way a JSON response is navigated with `jsonPath()`.
`postFormUrlEncoded_fieldsAreEchoedBack`	application/x-www-form-urlencoded	`.formParam(...)` correctly URL-encodes and sends classic HTML-form-style data.
`postMultipart_fileAndFieldsAreEchoedBack`	multipart/form-data	A real file and ordinary form fields can be sent together in a single multipart request — the pattern any file-upload API test needs.

pdfsent.com

17. JSON Schema Validation

Field-by-field assertions (`.body("data.id", notNullValue())`) only check what you remember to check. A JSON Schema check validates the *entire* shape of a response in one call — required fields, types, and structure — and fails the moment the API adds, removes, renames, or retypes anything the schema declares. This framework validates the single-user response against the schema below.

`src/test/resources/schemas/single-user-schema.json`

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "SingleUserResponse",
  "type": "object",
  "required": ["data"],
  "properties": {
    "data": {
      "type": "object",
      "required": ["id", "email", "first_name", "last_name", "avatar"],
      "properties": {
        "id": { "type": "integer" },
        "email": { "type": "string", "format": "email" },
        "first_name": { "type": "string" },
        "last_name": { "type": "string" },
        "avatar": { "type": "string" }
      }
    },
    "support": {
      "type": "object",
      "properties": {
        "url": { "type": "string" },
        "text": { "type": "string" }
      }
    }
  }
}
```

17.1 How it's wired into a test

```
.body(matchesJsonSchemaInClasspath("schemas/single-user-schema.json"))
```

The schema file lives under `src/test/resources` so it ends up on the test classpath at build time, and `matchesJsonSchemaInClasspath` (from the `json-schema-validator` module declared in Chapter 4) loads it by that relative path.

pdfsent.com

18. HTML Reporting with ExtentReports

Console output is not something a project manager, client, or non-technical teammate can read. ExtentReports turns every TestNG run into a styled, self-contained HTML file with a pass/fail summary and per-test detail, without requiring any change inside the test classes themselves — it hooks in purely through a TestNG listener.

src/main/java/com/framework/reports/ExtentManager.java

```
package com.framework.reports;

import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.reporter.ExtentSparkReporter;

import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;

/** Creates one ExtentReports HTML file per run under /reports, timestamped. */
public final class ExtentManager {

    private static ExtentReports extent;

    private ExtentManager() {
    }

    public static synchronized ExtentReports getInstance() {
        if (extent == null) {
            String timestamp = new SimpleDateFormat("yyyy-MM-dd_HH-mm-ss").format(new Date());
            File reportDir = new File("reports");
            if (!reportDir.exists()) {
                reportDir.mkdirs();
            }
            String reportPath = "reports/api-test-report-" + timestamp + ".html";

            ExtentSparkReporter spark = new ExtentSparkReporter(reportPath);
            spark.config().setDocumentTitle("REST Assured API Test Report");
            spark.config().setReportName("API Automation Results");

            extent = new ExtentReports();
            extent.attachReporter(spark);
            extent.setSystemInfo("Framework", "REST Assured + TestNG");
            extent.setSystemInfo("Java", System.getProperty("java.version"));
        }
        return extent;
    }
}
```

src/main/java/com/framework/reports/ExtentTestListener.java

```
package com.framework.reports;

import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.Status;
import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;

/**
```


pdfsent.com

```
 * Wire this into testng.xml under <listeners> to get an HTML report for every run
 * without touching individual test classes.
 */
public class ExtentTestListener implements ITestListener {

    private static final ThreadLocal<ExtentTest> testThread = new ThreadLocal<>();
    private ExtentReports extent;

    @Override
    public void onStart(ITestContext context) {
        extent = ExtentManager.getInstance();
    }

    @Override
    public void onTestStart(ITestResult result) {
        ExtentTest test = extent.createTest(result.getMethod().getMethodName());
        testThread.set(test);
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        testThread.get().log(Status.PASS, "Test passed");
    }

    @Override
    public void onTestFailure(ITestResult result) {
        testThread.get().log(Status.FAIL, result.getThrowable());
    }

    @Override
    public void onTestSkipped(ITestResult result) {
        testThread.get().log(Status.SKIP, "Test skipped: " + result.getThrowable());
    }

    @Override
    public void onFinish(ITestContext context) {
        extent.flush();
    }
}
```

pdfsent.com

18.1 How the pieces connect

1. `ExtentManager` is a singleton that creates one timestamped HTML report per run and configures its title.
2. `ExtentTestListener` implements TestNG's `ITestListener` interface. Each lifecycle callback (`onTestStart`, `onTestSuccess`, `onTestFailure`, `onTestSkipped`) logs the corresponding status into the report.
3. A `ThreadLocal<ExtentTest>` is used deliberately — because `testng.xml` runs classes in parallel, each thread needs to log to its own test entry, not a shared one, or reports from different tests would interleave and corrupt each other.
4. The listener is registered once, declaratively, in `testng.xml` (Chapter 19) — no test class needs to know reporting exists.

pdfsent.com

19. Test Suite Orchestration — testng.xml

TestNG's suite file is the single place that decides which classes run, in what grouping, with what parallelism, and with which listeners attached. Keeping this out of Java code means the run configuration can change without a recompile.

testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="REST Assured API Automation Suite" parallel="classes" thread-count="3">

    <listeners>
        <listener class-name="com.framework.reports.ExtentTestListener"/>
    </listeners>

    <test name="User CRUD Tests">
        <classes>
            <class name="com.framework.tests.users.GetUsersTest"/>
            <class name="com.framework.tests.users.CreateUserTest"/>
            <class name="com.framework.tests.users.UpdateUserTest"/>
            <class name="com.framework.tests.users.PatchUserTest"/>
            <class name="com.framework.tests.users.DeleteUserTest"/>
        </classes>
    </test>

    <test name="Auth Tests">
        <classes>
            <class name="com.framework.tests.auth.AuthTest"/>
        </classes>
    </test>

    <test name="Content Type Tests">
        <classes>
            <class name="com.framework.tests.contenttypes.ContentTypeTest"/>
        </classes>
    </test>

    <test name="HTTP Method Tests">
        <classes>
            <class name="com.framework.tests.methods.HttpMethodsTest"/>
        </classes>
    </test>

</suite>
```

19.1 Notable settings

- `parallel="classes" thread-count="3"` — different test classes run concurrently on up to three threads, which is exactly the scenario that made the static-spec fix in Chapter 7 necessary.
- Four `<test>` blocks group classes by concern (CRUD, auth, content type, HTTP method) rather than dumping every class into one block — this makes it possible to re-run just one group, e.g. `-testname "Auth Tests"`, without touching the rest of the suite.
- The `<listeners>` block is what activates ExtentReports for the whole suite in one line.

pdfsent.com

20. Running the Framework

20.1 Prerequisites

- Java 11 or newer, and Maven, on the PATH.
- A free API key from reqres.in, set in `config.properties` or passed on the command line (reqres.in began requiring this in 2024).

20.2 Common commands

```
# confirm the project compiles
mvn clean install -DskipTests

# run the entire suite
mvn clean test

# run a single class
mvn test -Dtest=CreateUserTest

# override configuration without editing files
mvn test -Dapi.key=YOUR_KEY -Dreqres.base.uri=https://reqres.in/api
```

20.3 Where results go

TestNG's own console/XML report lands in `target/surefire-reports` as usual. The ExtentReports HTML file lands in `reports/api-test-report-<timestamp>.html` at the project root, timestamped so successive runs never overwrite each other.

pdfsent.com

pdfsent.com

21. Extending the Framework

21.1 Adding a new endpoint against an existing target API

1. Add the path as a constant in `Endpoints`.
2. Add a POJO (or pair of POJOs, if the request and response shapes differ) under the matching `pojo` sub-package.
3. Write the test class under `tests`, following the same `given().spec(...).when()....then()` pattern used throughout this document.
4. Register the new class in `testng.xml` under the appropriate `<test>` block, or create a new one.

21.2 Adding an entirely new target API

1. Add its base URI to `config.properties`.
2. Add a new `RequestSpecification` builder method to `BaseTest`, following the exact static-lazy-init pattern used for `reqresSpec()` and `httpbinSpec()` — do not use an instance field populated in `@BeforeSuite` (see Chapter 7.1 for why).
3. Everything else — POJOs, endpoint constants, test classes, suite wiring — follows the same pattern as any other endpoint.

21.3 Swapping the reporting library

Because reporting is wired in purely through a TestNG `ITestListener`, swapping ExtentReports for Allure or another reporter means writing a new listener class and changing one line in `testng.xml` — no test class needs to change.

pdfsent.com

pdfsent.com

22. Best Practices & Common Pitfalls

22.1 Practices this framework follows

Practice	Why it matters
One RequestSpecification per target API, built once	Avoids re-declaring base URI/headers/timeouts in every test; guarantees consistency.
POJOs for known, stable response shapes	Turns a whole class of "field renamed and nobody noticed" bugs into compile errors.
`@JsonIgnoreProperties(ignoreUnknown = true)` on every response POJO	A response POJO should describe what the test cares about, not enumerate every field the API happens to return — new fields the API adds should never break deserialization.
Externalized, override-able configuration	The same test code runs against dev, staging, or CI without a single code change.
Request + response logging attached by default	A failing test should never require re-running with extra flags just to see what was actually sent.
Negative-path tests alongside positive-path tests	An API that "works" is not proven by happy-path tests alone; 4xx/404 behaviour is part of the contract.

22.2 Common pitfalls this framework deliberately avoids

- **Shared mutable state in a base class populated by a lifecycle hook that only fires once per suite.** Covered in depth in Chapter 7.1 — the exact bug this framework's own base class had, and the fix (static + lazy + synchronized) that applies to any shared framework state, not just request specs.
- **Hardcoding base URIs or credentials inside test classes.** Once a value is hardcoded, running against a second environment means editing test code, which is a sign the abstraction is in the wrong place.
- **Asserting on raw JSON paths for a response whose full shape you already know.** A dozen `.body("field", ...)` assertions are more brittle and less readable than one POJO deserialization plus ordinary Java assertions, or one schema validation call.
- **Treating PUT and PATCH as interchangeable.** They are semantically different (full vs. partial replacement) even when a demo API happens to accept either for the same route — a real API often does not.
- **Skipping negative and edge cases because the "happy path passes."** 404s, 400s, empty bodies, and unexpected content types are exactly where undertested APIs fail in production.

pdfsent.com

23. Glossary of REST Assured Terms

Term	Meaning
`given()`	Starts building a request — headers, body, params, content type.
`when()`	Fires the actual HTTP call (`.get()` , `.post()` , etc.).
`then()`	Begins asserting on the response.
RequestSpecification	A reusable, pre-built bundle of request configuration (base URI, headers, content type) applied with `.spec(...)` .
ResponseSpecification	The response-side equivalent — reusable expected-response configuration, e.g. a max response time.
Path parameter	A placeholder in a URL, e.g. `{id}` , substituted at request time with `.pathParam()` .
Query parameter	A key-value pair appended after `?` in a URL, set with `.queryParam()` .
Form parameter	A key-value pair sent in an `application/x-www-form-urlencoded` body, set with `.formParam()` .
Hamcrest matcher	An assertion-building library (`equalTo` , `notNullValue` , `lessThan` , ...) that REST Assured uses natively inside `.body()` assertions.
JsonPath / XmlPath	REST Assured's built-in navigators for reading values out of a JSON or XML response by path, without a POJO.
JSON Schema	A formal, machine-checkable description of a JSON document's structure, used here to validate an entire response shape in one assertion.
DataProvider	A TestNG annotation that feeds one test method multiple sets of input data, running it once per row.
POJO	Plain Old Java Object — here, a class whose fields mirror a JSON or XML structure for type-safe (de)serialization.

pdfsent.com

pdfsent.com

24. Appendix — Full File Tree

The complete list of files that make up this framework, for quick reference.

Complete project

```
restassured-framework/
├── pom.xml
├── testng.xml
├── README.md
├── src/
│   ├── main/
│   │   ├── java/com/framework/
│   │   │   ├── config/
│   │   │   │   └── ConfigManager.java
│   │   │   ├── constants/
│   │   │   │   └── Endpoints.java
│   │   │   ├── base/
│   │   │   │   └── BaseTest.java
│   │   │   ├── reports/
│   │   │   │   ├── ExtentManager.java
│   │   │   │   └── ExtentTestListener.java
│   │   │   └── pojo/
│   │   │       ├── user/      (User, ListUsersResponse, SingleUserResponse,
│   │   │       │               CreateUserRequest/Response, UpdateUserRequest/Response,
Support)
│   │   │       ├── auth/      (RegisterRequest/Response, LoginRequest/Response,
ErrorResponse)
│   │   │       └── httpbin/   (HttpBinResponse)
│   │   └── resources/
│   │       └── config.properties
│   └── test/
│       ├── java/com/framework/tests/
│       │   ├── users/         (GetUsersTest, CreateUserTest, UpdateUserTest,
│       │   │                   PatchUserTest, DeleteUserTest)
│       │   ├── auth/          (AuthTest)
│       │   ├── contenttypes/  (ContentTypeTest)
│       │   └── methods/       (HttpMethodsTest)
│       └── resources/schemas/
│           └── single-user-schema.json
```

24.1 Document summary

Total source files documented	27 Java classes, 1 JSON schema, 3 config/build files
HTTP methods covered	GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content types covered	application/json, application/xml, application/x-www-form-urlencoded, multipart/form-data
Target demo APIs	reqres.in (JSON CRUD), httpbin.org (protocol/content-type echo)

pdfsent.com

End of notes.

pdfsent.com

pdfsent.com