SE2062 - TestLang++ (Java) - Backend API Testing DSL

Individual take-home assignment (3 weeks). You will design and implement a small DSL for HTTP API testing, and compile it into runnable JUnit 5 tests that use Java 11+ HttpClient. No Selenium or third-party libraries.

Duration: ~18-24 hours across 3 weeks Deadline: 25th October 2025 Total Marks: 100

Objective

Create a DSL (TestLang++) to describe HTTP tests (GET/POST/PUT/DELETE, headers, body, assertions). Implement a scanner and parser (Lex/Yacc or JFlex/CUP) that translate test files into a single GeneratedTests.java JUnit 5 class. Compile and run the generated tests against a local backend.

Learning Outcomes

- **Design a precise language from a prose specification.**
- **Build a Java scanner+parser with meaningful error messages.**
- **Generate idiomatic JUnit 5 code that performs HTTP requests via java.net.http.HttpClient.**
- **Demonstrate end-to-end testing on a local Spring Boot backend.**

What You Will Build (End-to-End)

1. **Author tests in .test (your DSL).**
2. **Run your parser → generates GeneratedTests.java.**
3. **Compile & run with JUnit 5 → see pass/fail.**

TestLang++ Language Reference (Trimmed Scope)

This is the authoritative spec. You must derive tokens, precedence, and productions from this description. Do not add extra features beyond "Optional Extras".

1) File Structure & Order

Construct	Required?	Multiplicity Semantics
config { ... }	No	0..1, at top. Defines base_url and default headers for all requests.
let name = value;	No	0..N. Declares string or integer variables. Refer with $name inside strings and paths.
test Name { ... }	Yes	1..N. Each block compiles to one @Test method. Steps execute in order.

2) Lexical Rules

Item	Pattern / Rule	Examples	Notes
Whitespace	spaces / tabs / newlines		Separates tokens.
Line comments	// until end of line	// comment	Block comments not required.
Identifiers	[A-Za-z_][A-Za-z0-9_]*	Login, GetUser	Case-sensitive.
Numbers	non-negative integers	0, 200, 42	No floats.
Strings	double-quoted, supports \" and \\	"hello"	No multiline strings.

Item	**Pattern / Rule**	**Examples**	**Notes**
Var refs	$name inside strings/paths	"/u/$id"	No vars outside strings/paths.

Reserved words: config, base_url, header, let, test, GET, POST, PUT, DELETE, expect, status, body, contains

3) config Block (optional)

```
config {
   base_url = "http://localhost:8080";
   header "Content-Type" = "application/json";
   header "X-App" = "TestLangDemo";
}
```

- base_url: If present and a request path starts with "/", the effective URL is base_url + path. If absent, paths must be absolute URLs.
- header "K" = "V": Zero or more default headers applied to every request (request-level headers may add/override).

4) Variables

```
let user = "admin";
let id = 42;
```

- Right-hand side is string or integer.
- Use $user, $id in strings and paths: "/api/users/$id".
- Names must be unique within a file.

5) test Block & Steps (required)

Form	**Notes**
test Name { steps }	Name is an identifier used to name the JUnit method test_Name.

5.1 HTTP Request Statements

Statement	Meaning
GET "/path_or_url";	Sends a GET. No request block required.
DELETE "/path_or_url";	Sends a DELETE. No request block required.
POST "/path_or_url" { ... };	Sends a POST. Request block may include headers/body.
PUT "/path_or_url" { ... };	Sends a PUT. Request block may include headers/body.

5.2 Request Block (optional for GET/DELETE; optional for POST/PUT)

Line	Semantics	Example
header "K" = "V";	Applies to this request only (in addition to config headers).	header "Content-Type" = "application/json";
body = "string";	Single-line request body (e.g., JSON as a string).	body = "{ \"username\": \"$user\" }";

5.3 Assertions (after a request)

Form	Checks	Example
expect status = NUMBER;	HTTP status equals NUMBER.	expect status = 200;
expect header "K" = "V";	Response header equals string.	expect header "Content-Type" = "application/json";
expect header "K" contains "sub";	Response header contains substring.	expect header "Content-Type" contains "json";
expect body contains "sub";	Response body contains substring.	expect body contains "\"token\":";

Note: Each test must execute ≥1 request and ≥2 assertions.

Example Input → Expected Meaning

```
config {
    base_url = "http://localhost:8080";
    header "Content-Type" = "application/json";
}

// variables
let user = "admin";
let id = 42;

test Login {
    POST "/api/login" {
        body = "{ \"username\": \"$user\", \"password\": \"1234\" }";
    }
    expect status = 200;
    expect header "Content-Type" contains "json";
    expect body contains "\"token\":";
}
```

```
test GetUser {
    GET "/api/users/$id";
    expect status = 200;
    expect body contains "\"id\": 42";
}
```

Your generator must produce one JUnit 5 class with two @Test methods executing these requests and assertions via HttpClient.

Required Code Generation Shape (JUnit 5 + HttpClient)

You may factor helpers; the following shape must compile and run.

```
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import java.net.http.*;
import java.net.*;
import java.time.Duration;
import java.nio.charset.StandardCharsets;
import java.util.*;


public class GeneratedTests {
    static String BASE = "http://localhost:8080";
    static Map<String, String> DEFAULT_HEADERS = new HashMap<>();
    static HttpClient client;
    @BeforeAll
    static void setup() {
        client = HttpClient.newBuilder().connectTimeout(Duration.ofSeconds(5)).build();
        DEFAULT_HEADERS.put("Content-Type", "application/json");
    }


    @Test
    void test_Login() throws Exception {
        HttpRequest.Builder b = HttpRequest.newBuilder(URI.create(BASE + "/api/login"))
            .timeout(Duration.ofSeconds(10))
            .POST(HttpRequest.BodyPublishers.ofString("{ \"username\": \"admin\",
\"password\": \"1234\" }"));
        for (var e : DEFAULT_HEADERS.entrySet()) b.header(e.getKey(), e.getValue());
        HttpResponse<String> resp = client.send(b.build(),
HttpResponse.BodyHandlers.ofString(StandardCharsets.UTF_8));
        assertEquals(200, resp.statusCode());
        assertTrue(resp.headers().firstValue("Content-Type").orElse("").contains("json"));
        assertTrue(resp.body().contains("\"token\":"));
    }


    @Test
    void test_GetUser() throws Exception {
        HttpRequest.Builder b = HttpRequest.newBuilder(URI.create(BASE +
"/api/users/42"))
```

```
            .timeout(Duration.ofSeconds(10))

            .GET();

        for (var e : DEFAULT_HEADERS.entrySet()) b.header(e.getKey(), e.getValue());

        HttpResponse<String> resp = client.send(b.build(),
HttpResponse.BodyHandlers.ofString(StandardCharsets.UTF_8));

        assertEquals(200, resp.statusCode());

        assertTrue(resp.body().contains("\"id\": 42"));

    }

}
```

Out-of-Scope (keeps workload reasonable)

- No JSON parsing/JSONPath.
- No loops/conditionals/macros/hooks.
- No retries or per-request timeouts (you can hard-code a default in Java).
- No multiline strings; single line only.
- No response capture/assign; assertions are the observable outcome.
- No suites; one file one class is sufficient.

Optional Extras (bonus up to +10)

- Triple-quoted multiline strings for body.
- Range status check (e.g., expect status in 200..299).

Invalid Input Examples (your parser must error clearly)

Invalid	Why	Example Error Message
let 2a = "x";	Identifier cannot start with a digit	Line 1: expected IDENT after 'let'
POST "/x" { body = 123; };	Body must be a string	Line N: expected STRING after 'body ='

Invalid	Why	Example Error Message
expect status = "200";	Status must be integer	Line N: expected NUMBER for status
GET "/x" expect status = 200;	Missing semicolon after request	Line N: expected ';' after request

Reference Backend (3 Cases)

A minimal Spring Boot backend (Java 11) is provided to exercise the DSL. Run it on http://localhost:8080. Each case maps to your assertions.

Endpoint #	Purpose	Sample TestLang++
1. POST /api/login	POST body, status, header contains, body contains	POST "/api/login" { header "Content-Type" = "application/json"; body = "{ \"username\": \"$user\", \"password\": \"1234\" }"; } expect status = 200; expect header "Content-Type" contains "json"; expect body contains "\"token\":";
2. GET /api/users/$id	GET + var substitution, status, body contains	GET "/api/users/$id"; expect status = 200; expect body contains "\"id\": 42";
3. PUT /api/users/$id	PUT body, header equals, header contains, multiple body contains	PUT "/api/users/$id" { header "Content-Type" = "application/json"; body = "{\"role\": \"ADMIN\" }"; } expect status = 200; expect header "X-App" = "TestLangDemo"; expect header "Content-Type" contains "json"; expect body contains "\"updated\": true"; expect body contains "\"role\": \"ADMIN\"";

Build & Run Backend

From project root of the provided Spring Boot demo	**Quick Manual Checks (cURL)**
mvn clean package	**Login** curl -i -X POST http://localhost:8080/api/login \ -H 'Content-Type: application/json' \ -d '{"username": "admin", "password": "1234"}'
java -jar target/testlang-demo-0.0.1-SNAPSHOT.jar	**Get user** curl -i http://localhost:8080/api/users/42
Server: http://localhost:8080	**Update user** curl -i -X PUT http://localhost:8080/api/users/42 \ -H 'Content-Type: application/json' \ -d '{"role":"ADMIN"}'

Submission Requirements

- **Language Compliance:** Implement the spec above exactly.
- **Scanner & Parser:** lexer.l/.flex and parser.y/.cup producing GeneratedTests.java.
- **Examples:** example.test (≥2 tests covering GET & POST); GeneratedTests.java (output).
- **README.md:** how to run backend, run parser, compile & run tests.
- **Demo video (≤3 min):** Write DSL → run parser → compile/run JUnit → show pass/fail + one invalid DSL with your error message.

Marking Rubric (100 marks)

Criteria	**Marks**	**Poor (0-35%)**	**Below Avg (36-45%)**	**Average (46-65%)**	**Good (66-75%)**	**Very Good (76-100%)**
Language Design	**25**	**Deviates; missing constructs**	**Partial; unclea**	**Implements require**	**Edge cases handled; clear docs**	**Robust and well-documen**

Criteria	Marks	Poor (0-35%)	Below Avg (36-45%)	Average (46-65%)	Good (66-75%)	Very Good (76-100%)
Fidelity			r semantics	d constructs		ted behavior
Scanner & Parser Quality	30	Unstable; crashes	Parses only trivial inputs	Parses examples reliably	Good diagnostics; recovery	Clean structure; maintainable; tests
Code Generation (JUnit)	30	Uncompilable / wrong	Compiles with logic flaws	Runs GET/POST with asserts	Handles headers/body/vars properly	Idiomatic; reusable helpers; clean
Demo & Examples	15	Missing/unclear	Partial pipeline	End-to-end shown	Clear pass/fail + invalid case	Polished; reproducible README

Suggested 3-Week Plan

- Week 1: Tokens & scanner → parse blocks → AST model.
- Week 2: Implement statements; var substitution; generate JUnit template; compile.
- Week 3: Robust errors; finalize codegen; examples; demo recording.

▲ Academic Integrity & AI Use

- Since this is a open assignment you can use AI tools as you want to generate your lexer/parser or the generated JUnit code and complete the full assignment.
- When in doubt, cite references and ask questions early.