RestAssured + Cucumber GraphQL Dynamic Repo

This document contains a complete, ready-to-drop-in project layout and all source files for a **Rest Assured + Cucumber** test repo that:

- Handles **GraphQL queries and mutations**
- Builds **dynamic selection sets** from `sections` and `section.field` expressions (full-section default vs. specific fields)
- Supports **nested sections** and merges mixed requests
- Loads a **schema/field repository** from YAML (or falls back to a built-in map)
- Is wired to **Cucumber** feature files that include Examples with both **variable input** (GraphQL variables) and **query selection expressions** (sections/fields)

Project structure

```
graphql-restassured-cucumber/
├── pom.xml
├── README.md
├── src/test/java/com/example/graphql/
│   ├── client/GraphQLClient.java
│   ├── builder/GraphQLQueryBuilder.java
│   ├── repository/QueryFieldRepository.java
│   ├── steps/GraphQLSteps.java
│   ├── support/ScenarioContext.java
│   └── runners/CucumberTestRunner.java
├── src/test/resources/fields.yaml
├── src/test/resources/features/get_customer.feature
└── src/test/resources/log4j2.xml
```

Build system (pom.xml)

Paste this `pom.xml` into your project root. It sets up JUnit, Cucumber, Rest Assured, Jackson/YAML, and logging.

```
<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>graphql-restassured-cucumber</artifactId>
<version>1.0-SNAPSHOT</version>
<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <cucumber.version>7.11.0</cucumber.version>
    <restassured.version>5.3.0</restassured.version>
</properties>
<dependencies>
    <!-- Cucumber JVM -->
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>${cucumber.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>io.cucumber</groupId>
        <artifactId>cucumber-junit-platform-engine</artifactId>
        <version>${cucumber.version}</version>
        <scope>test</scope>
    </dependency>

    <!-- JUnit Platform -->
    <dependency>
        <groupId>org.junit.platform</groupId>
        <artifactId>junit-platform-suite-api</artifactId>
        <version>1.10.0</version>
        <scope>test</scope>
    </dependency>

    <!-- Rest Assured -->
    <dependency>
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>${restassured.version}</version>
        <scope>test</scope>
    </dependency>

    <!-- Jackson for JSON/YAML -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.15.2</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
```

```
            <artifactId>jackson-dataformat-yaml</artifactId>
            <version>2.15.2</version>
        </dependency>

        <!-- SLF4J + Log4j2 -->
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.20.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.20.0</version>
        </dependency>

    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.0.0-M9</version>
            </plugin>
        </plugins>
    </build>
</project>
```

YAML repository: `src/test/resources/fields.yaml`

This is the canonical repository of sections and nested fields. You may extend it as needed.

```
result:
  - success
  - message
  - dataSourceType
  - timestamp
  - metaData:
      - id
      - state
      - isActive
      - submittedAt
      - submittedBy
```

```
      - approvedAt
      - approvedBy
      - rejectionReason
      - recordCreatedBy:
          - userId
          - userName

customer:
  - brand
  - customerId
  - customerName
  - primaryContact

draft:
  - customerAgreement:
      - agreementId
      - customerStartDt
      - customerEndDt
      - customerType
      - bankLineBrand
  - customerContactDetails:
      - custContactDtlsId
      - contactName
      - contactType
      - emailAddress
      - phoneNumber
  - financialInstitutionSettings:
      - settlementAccount
      - customerOwnBic
      - customerOwnNcc
      - fiSortCodeRanges:
          - fiSortCodeId
          - customerId
          - lowerRange
          - upperRange
  - deliveryChannelDetails:
      - deliveryChannelType
      - tppId
      - clientId
      - scope

customerAccountReportPreference:
  - eod
  - reportDeliveryDays
  - outputFormat
```

`QueryFieldRepository.java`

This class loads `fields.yaml` (fallback to built-in map) and exposes `buildSelection(...)` to convert requested expressions into a `Map<String,Object>` selection that the builder understands.

```
package com.example.graphql.repository;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.dataformat.yaml.YAMLFactory;

import java.io.InputStream;
import java.util.*;

public class QueryFieldRepository {
    private static final Map<String, List<Object>> FIELDS = new
LinkedHashMap<>();

    static {
        // load from YAML on static init. If YAML missing or fails, fall back to
a small built-in set.
        try (InputStream is = QueryFieldRepository.class.getResourceAsStream("/
fields.yaml")) {
            if (is != null) {
                ObjectMapper mapper = new ObjectMapper(new YAMLFactory());
                Map<String, Object> map = mapper.readValue(is, Map.class);
                map.forEach((k, v) -> FIELDS.put(k, normalizeList(v)));
            }
        } catch (Exception e) {
            // fallback built-in
            FIELDS.put("customer",
List.of("brand","customerId","customerName"));
            FIELDS.put("result",
List.of("success","message","dataSourceType","timestamp", Map.of("metaData",
List.of("id","state","isActive", Map.of("recordCreatedBy",
List.of("userId","userName"))))));
        }
    }

    @SuppressWarnings("unchecked")
    private static List<Object> normalizeList(Object raw) {
        List<Object> out = new ArrayList<>();
        if (raw instanceof List) {
            for (Object o : (List<Object>) raw) {
                if (o instanceof String) out.add(o);
                else if (o instanceof Map) {
                    // keep nested map as Map<String,Object> where value is
```

```
List<Object>
                    Map<String,Object> nm = new LinkedHashMap<>();
                    ((Map<String,Object>) o).forEach((k,v) -> nm.put(k,
normalizeList(v)));
                    out.add(nm);
                }
            }
        }
        return out;
    }

    public static List<Object> getAllFields(String section) {
        return FIELDS.get(section);
    }

    /**
     * Build selection map from requested expressions (e.g.
["customer","draft.customerAgreement.agreementId"]).
     * Returns Map<section, List<Object>> where List<Object> conforms to the
same shape as FIELDS values.
     */
    public static Map<String, Object> buildSelection(List<String>
requestedExpressions) {
        Map<String, FieldNode> roots = new LinkedHashMap<>();

        for (String expr : requestedExpressions) {
            if (expr == null || expr.isBlank()) continue;
            String[] parts = expr.trim().split("\\.");
            String root = parts[0];
            FieldNode node = roots.computeIfAbsent(root, k -> new FieldNode());

            if (parts.length == 1) node.includeAll = true;
            else node.addPath(Arrays.copyOfRange(parts, 1, parts.length));
        }

        Map<String, Object> selection = new LinkedHashMap<>();
        for (Map.Entry<String, FieldNode> entry : roots.entrySet()) {
            String root = entry.getKey();
            FieldNode node = entry.getValue();

            if (node.includeAll) {
                List<Object> full = FIELDS.get(root);
                selection.put(root, full != null ? full : List.of());
            } else {
                List<Object> repo = FIELDS.get(root);
                List<Object> built = buildListFromNode(node, repo);
                selection.put(root, built);
            }
```

```
        }

        return selection;
    }

    @SuppressWarnings("unchecked")
    private static List<Object> buildListFromNode(FieldNode node, List<Object>
repoList) {
        List<Object> out = new ArrayList<>();
        Set<String> handled = new HashSet<>();

        if (repoList != null) {
            for (Object entry : repoList) {
                if (entry instanceof String) {
                    String fieldName = (String) entry;
                    if (node.fields.contains(fieldName)) {
                        out.add(fieldName);
                        handled.add(fieldName);
                    }
                } else if (entry instanceof Map) {
                    Map<?, ?> mapEntry = (Map<?, ?>) entry;
                    for (Map.Entry<?, ?> me : mapEntry.entrySet()) {
                        String key = (String) me.getKey();
                        Object repoVal = me.getValue();

                        if (node.fields.contains(key)) {
                            out.add(Map.of(key, repoVal));
                            handled.add(key);
                        } else if (node.children.containsKey(key)) {
                            FieldNode childNode = node.children.get(key);
                            List<Object> childRepoList = repoVal instanceof
List ? (List<Object>) repoVal : List.of();
                            List<Object> childBuilt =
buildListFromNode(childNode, childRepoList);
                            out.add(Map.of(key, childBuilt));
                            handled.add(key);
                        }
                    }
                }
            }
        }

        for (String f : node.fields) if (!handled.contains(f)) out.add(f);
        for (String childKey : node.children.keySet()) if (!
handled.contains(childKey)) {
            List<Object> childBuilt =
buildListFromNode(node.children.get(childKey), List.of());
            out.add(Map.of(childKey, childBuilt));
```

```
        }

        return out;
    }

    private static class FieldNode {
        final Set<String> fields = new LinkedHashSet<>();
        final Map<String, FieldNode> children = new LinkedHashMap<>();
        boolean includeAll = false;

        void addPath(String[] parts) {
            if (parts.length == 0) return;
            String head = parts[0];
            if (parts.length == 1) fields.add(head);
            else {
                FieldNode child = children.computeIfAbsent(head, k -> new
FieldNode());
                child.addPath(Arrays.copyOfRange(parts, 1, parts.length));
            }
        }
    }
}
```

`GraphQLQueryBuilder.java`

This builder converts the `Map<String,Object>` selection to a GraphQL selection set string and injects variables/arguments if present.

```
package com.example.graphql.builder;

import java.util.List;
import java.util.Map;

public class GraphQLQueryBuilder {

    /**
     * Build a full GraphQL operation (query or mutation). `selection` is a map
from root -> List<Object> (strings or nested maps).
     * `operationName` is the field to call (e.g. getCustomerDataById).
     * `operationType` is either "query" or "mutation".
     * `variablesDeclaration` is optional like "($request:
RequestSearchByCustomerIdInput!)" or null.
     * `arguments` is optional like "(request: $request)" or null.
     */
```

```
    public String buildOperation(String operationType,
                                 String operationName,
                                 String variablesDeclaration,
                                 String arguments,
                                 Map<String, Object> selection) {
        StringBuilder sb = new StringBuilder();
        sb.append(operationType).append(" ");
        if (variablesDeclaration != null && !variablesDeclaration.isBlank())
sb.append(variablesDeclaration).append(" ");
        sb.append("{ ").append(operationName);
        if (arguments != null && !arguments.isBlank()) sb.append(arguments);
        sb.append(" { ");

        // iterate selection map
        for (Map.Entry<String, Object> entry : selection.entrySet()) {
            sb.append(entry.getKey()).append(" { ");
            buildFields(sb, entry.getValue());
            sb.append(" } \n");
        }

        sb.append(" } }");
        return sb.toString();
    }

    @SuppressWarnings("unchecked")
    private void buildFields(StringBuilder sb, Object node) {
        if (node instanceof List) {
            for (Object o : (List<Object>) node) {
                if (o instanceof String) sb.append(o).append(" ");
                else if (o instanceof Map) {
                    Map<String, Object> m = (Map<String, Object>) o;
                    for (Map.Entry<String, Object> e : m.entrySet()) {
                        sb.append(e.getKey()).append(" { ");
                        buildFields(sb, e.getValue());
                        sb.append(" } ");
                    }
                }
            }
        }
    }
}
```

`GraphQLClient.java`

Sends GraphQL operations to a configured endpoint using Rest Assured. It can send `query` or `mutation` operations, and accepts variables as a map.

```
package com.example.graphql.client;

import io.restassured.RestAssured;
import io.restassured.response.Response;

import java.util.Map;

public class GraphQLClient {
    private final String baseUrl;

    public GraphQLClient(String baseUrl) {
        this.baseUrl = baseUrl;
    }

    public Response send(String operationPayload, Map<String, Object>
variables, Map<String,String> headers) {
        // body: { "query": "...", "variables": { ... } }
        io.restassured.specification.RequestSpecification req =
RestAssured.given();
        req.baseUri(baseUrl);
        req.header("Content-Type", "application/json");
        if (headers != null) headers.forEach(req::header);

        // build payload
        String body;
        if (variables == null || variables.isEmpty()) {
            body = String.format("{ \"query\": \"%s\" }",
escape(operationPayload));
        } else {
            // simple JSON generation using replace - for production use Jackson
to produce payload
            try {
                com.fasterxml.jackson.databind.ObjectMapper om = new
com.fasterxml.jackson.databind.ObjectMapper();
                String varsJson = om.writeValueAsString(variables);
                body = String.format("{ \"query\": \"%s\", \"variables\":
%s }", escape(operationPayload), varsJson);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
        }
```

```
        return req.body(body).when().post().andReturn();
    }

    private String escape(String s) {
        return s.replace("\\","\\\\").replace("\n","\\n").replace("\"","\\\"");
    }
}
```

Note: GraphQLClient uses `post()` with no path; set `baseUrl` to full endpoint (e.g. `https://api.example.com/graphql`). You can change `req.baseUri(baseUrl)` to `RestAssured.baseURI = baseUrl; req.post("/graphql")` if you prefer separate base URI and path.

`ScenarioContext.java` (simple test context)

```
package com.example.graphql.support;

import java.util.HashMap;
import java.util.Map;

public class ScenarioContext {
    private static final ThreadLocal<Map<String, Object>> store =
ThreadLocal.withInitial(HashMap::new);

    public static void set(String key, Object val) { store.get().put(key,
val); }
    public static <T> T get(String key) { return (T) store.get().get(key); }
    public static void clear() { store.get().clear(); }
}
```

`GraphQLSteps.java` (Cucumber step definitions)

```
package com.example.graphql.steps;

import com.example.graphql.builder.GraphQLQueryBuilder;
import com.example.graphql.client.GraphQLClient;
import com.example.graphql.repository.QueryFieldRepository;
import com.example.graphql.support.ScenarioContext;
import io.cucumber.java.After;
import io.cucumber.java.Before;
import io.cucumber.java.en.Given;
```

```
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.restassured.response.Response;

import java.util.*;

import static org.junit.jupiter.api.Assertions.*;

public class GraphQLSteps {
    private GraphQLClient client;
    private GraphQLQueryBuilder builder;

    @Before
    public void setup() {
        // set endpoint; can be parameterized via system properties
        this.client = new GraphQLClient(System.getProperty("graphql.endpoint",
"http://localhost:8080/graphql"));
        this.builder = new GraphQLQueryBuilder();
    }

    @After
    public void tearDown() { ScenarioContext.clear(); }

    @Given("I build a {word} operation {word} with variables {string} and
selection {string}")
    public void buildOperation(String operationType, String operationName,
String variablesJson, String selectionExpressions) {
        // parse selection expressions (comma-separated). Each expression can be
`section` or `section.field1` or `section.sub.nestedField`.
        List<String> expressions =
Arrays.stream(selectionExpressions.split(","))
                .map(String::trim).filter(s->!s.isBlank()).toList();

        Map<String, Object> selection =
QueryFieldRepository.buildSelection(expressions);

        String varsDecl = null;
        String args = null;
        Map<String,Object> variables = null;

        if (variablesJson != null && !variablesJson.isBlank() && !
variablesJson.equals("null")) {
            // variablesJson is expected to be a small JSON like: {"request":
{"customerId":"123"}}
            try {
                com.fasterxml.jackson.databind.ObjectMapper om = new
com.fasterxml.jackson.databind.ObjectMapper();
                variables = om.readValue(variablesJson, Map.class);
```

```
            } catch (Exception e) { throw new RuntimeException(e); }

            // To keep it simple we automatically build variables declaration
and args if only top-level variable name present
            // e.g. variablesJson contains key "request" -> we create
variablesDecl = "($request: JSON)" and args = "(request: $request)"
            if (!variables.isEmpty()) {
                String firstKey = variables.keySet().iterator().next();
                // NOTE: types are unknown here; in real tests provide proper
variable declarations as part of feature or mapping
                varsDecl = String.format("($%s: JSON)", firstKey);
                args = String.format("(%s: $%s)", firstKey, firstKey);
            }
        }

        String operationPayload = builder.buildOperation(operationType,
operationName, varsDecl, args, selection);

        ScenarioContext.set("operationPayload", operationPayload);
        ScenarioContext.set("variables", variables);

        System.out.println("Built operation:\n" + operationPayload);
    }

    @When("I send the operation")
    public void sendOperation() {
        String payload = ScenarioContext.get("operationPayload");
        Map<String,Object> vars = ScenarioContext.get("variables");

        Response resp = client.send(payload, vars, null);
        ScenarioContext.set("lastResponse", resp);
    }

    @Then("response status should be {int}")
    public void response_status_should_be(Integer expected) {
        Response resp = ScenarioContext.get("lastResponse");
        assertNotNull(resp);
        assertEquals(expected.intValue(), resp.getStatusCode(), () -> "Response
body: " + resp.asString());
    }

}
```

Note: For variable type handling we use a simple approach: variables JSON is passed at runtime. For stricter typing, pass variable declarations in the feature or maintain a small map of variable types.

`get_customer.feature` (Cucumber feature)

This feature demonstrates: passing variables, selecting entire sections, or selecting specific fields.

```
Feature: Get customer data dynamic GraphQL
  Scenario Outline: Fetch customer data with dynamic selection and variables
    Given I build a <operationType> operation <operationName> with variables
<variables> and selection <selection>
    When I send the operation
    Then response status should be <status>

  Examples:
    | operationType | operationName          |
variables                                              |
selection                                          | status |
    | query         | getCustomerDataById    | {"request":
{"customerId":"CUST-123"}}          |
customer                                           | 200    |
    | query         | getCustomerDataById    | {"request":
{"customerId":"CUST-123"}}          |
customer.customerId,customer.customerName          | 200    |
    | query         | getCustomerDataById    | {"request":
{"customerId":"CUST-123"}}          |
customer,draft.customerAgreement.agreementId       | 200    |
    | mutation      | updateCustomer         | {"input":
{"customerId":"CUST-123","name":"X"}} |
customer.customerId,customer.customerName        | 200    |
```

`CucumberTestRunner.java`

Use JUnit Platform / Cucumber engine via configuration.

```
package com.example.graphql.runners;

import io.cucumber.junit.platform.engine.Cucumber;

@Cucumber
public class CucumberTestRunner { }
```

Logging config (optional) - `log4j2.xml`

Add a minimal Log4j2 config in `src/test/resources`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
  <Appenders>
    <Console name="Console" target="SYSTEM_OUT">
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
    </Console>
  </Appenders>
  <Loggers>
    <Root level="info">
      <AppenderRef ref="Console"/>
    </Root>
  </Loggers>
</Configuration>
```

How it works - summary

1. The feature passes a `selection` string (comma-separated expressions) and an optional `variables` JSON.
2. `QueryFieldRepository.buildSelection` converts those expressions into a selection `Map` using the YAML repository.
3. `GraphQLQueryBuilder` converts the selection `Map` into a GraphQL operation string (query or mutation), optionally adding variables and arguments.
4. `GraphQLClient` uses Rest Assured to POST the operation to your GraphQL endpoint.

This design allows: - Passing `customer` → includes all customer fields from YAML. - Passing `customer.customerId` → includes only that field. - Combining `customer` and `draft.customerAgreement.agreementId` — `customer` wins and is expanded fully while draft includes only requested nested field.

Next steps / customization

- Replace `JSON` variable type placeholder with specific GraphQL types if you want accurate variable declarations. Provide a small map of `{ varName -> varType }` and build declarations accordingly.
- Add response assertions and JSON path validations in step definitions to assert specific fields returned.
- Add support for fragments if your schema uses them.

- Add retry, timeout and authentication headers in `GraphQLClient` (e.g., OAuth bearer token injection).

If you'd like, I can now:

- Provide a ZIP of this skeleton (files created and zipped); or
- Convert `QueryFieldRepository` to be strict (throw if unknown field requested); or
- Add example assertions on response payloads in the feature/steps.

Which next step do you want?