🚀Spring Boot Webhook Auto-Solver (Startup Flow, JWT, SQL submit)

A minimal Spring Boot app that, **on startup**: 1) Calls the **Generate Webhook** API to get `webhook` + `accessToken` (JWT)
2) Picks **Question 1 or 2** based on the last two digits of `regNo` (odd → Q1, even → Q2)
3) Loads the final SQL from `/sql/question{1|2}.sql`, **stores it** in H2 for audit, and
4) **POSTs** `{ "finalQuery": "..." }` to the returned webhook with `Authorization: <accessToken>`.

> No controllers/HTTP endpoints trigger this. The entire flow runs on application startup via `ApplicationRunner`.

Project Structure

```
auto-solver/
├─ pom.xml
├─ README.md
├─ src/
│  ├─ main/
│  │  ├─ java/com/example/autosolver/
│  │  │  ├─ AutoSolverApplication.java
│  │  │  ├─ config/AppProperties.java
│  │  │  ├─ dto/GenerateWebhookRequest.java
│  │  │  ├─ dto/GenerateWebhookResponse.java
│  │  │  ├─ dto/FinalQueryPayload.java
│  │  │  ├─ model/Submission.java
│  │  │  ├─ repo/SubmissionRepository.java
│  │  │  ├─ service/SqlSolver.java
│  │  │  ├─ service/Question1Solver.java
│  │  │  ├─ service/Question2Solver.java
│  │  │  ├─ service/SqlSolverFactory.java
│  │  │  ├─ service/StartupOrchestrator.java
│  │  │  ├─ service/WebhookClient.java
│  │  ├─ resources/
│  │  │  ├─ application.yml
│  │  │  ├─ sql/question1.sql
│  │  │  └─ sql/question2.sql
│  └─ test/java/ (optional)
└─ target/auto-solver-0.0.1-SNAPSHOT.jar (generated after build)
```

---

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 https://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>auto-solver</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>auto-solver</name>
  <description>Spring Boot Webhook Auto-Solver</description>

  <properties>
    <java.version>17</java.version>
    <spring-boot.version>3.3.3</spring-boot.version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-dependencies</artifactId>
        <version>${spring-boot.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <!-- WebClient -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>

    <!-- JPA + H2 to store submitted SQL locally -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
```

```
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>

    <!-- Optional: validation for config -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>

    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <optional>true</optional>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        <configuration>
          <mainClass>com.example.autosolver.AutoSolverApplication</mainClass>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.11.0</version>
        <configuration>
          <source>${java.version}</source>
          <target>${java.version}</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

---

application.yml

```yaml
app:
  baseUrl: "https://bfhldevapigw.healthrx.co.in"
  name: "John Doe"
  regNo: "REG12347"   # change as needed
  email: "john@example.com"
  language: "JAVA"     # path segment required by the API

spring:
  datasource:
    url: jdbc:h2:mem:autosolver;DB_CLOSE_DELAY=-1
    driverClassName: org.h2.Driver
    username: sa
    password: ""
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
  h2:
    console:
      enabled: true
      path: /h2-console
logging:
  level:
    root: INFO
    org.springframework.web.reactive.function.client.ExchangeFunctions: INFO
```

AutoSolverApplication.java

```java
package com.example.autosolver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AutoSolverApplication {
  public static void main(String[] args) {
    SpringApplication.run(AutoSolverApplication.class, args);
  }
}
```

🛖 Config: `AppProperties.java`

```java
package com.example.autosolver.config;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.validation.annotation.Validated;

@Validated
@ConfigurationProperties(prefix = "app")
public class AppProperties {
  @NotBlank private String baseUrl;
  @NotBlank private String name;
  @NotBlank private String regNo;
  @Email @NotBlank private String email;
  @NotBlank private String language;

  public String getBaseUrl() { return baseUrl; }
  public void setBaseUrl(String baseUrl) { this.baseUrl = baseUrl; }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public String getRegNo() { return regNo; }
  public void setRegNo(String regNo) { this.regNo = regNo; }
  public String getEmail() { return email; }
  public void setEmail(String email) { this.email = email; }
  public String getLanguage() { return language; }
  public void setLanguage(String language) { this.language = language; }
}
```

Enable via `@EnableConfigurationProperties(AppProperties.class)` in the orchestrator or a `@Configuration` class.

🚤 DTOs

`GenerateWebhookRequest.java`

```java
package com.example.autosolver.dto;

public class GenerateWebhookRequest {
  private String name;
  private String regNo;
  private String email;

  public GenerateWebhookRequest() {}
  public GenerateWebhookRequest(String name, String regNo, String email) {
    this.name = name; this.regNo = regNo; this.email = email;
```

```
  }
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
  public String getRegNo() { return regNo; }
  public void setRegNo(String regNo) { this.regNo = regNo; }
  public String getEmail() { return email; }
  public void setEmail(String email) { this.email = email; }
}
```

`GenerateWebhookResponse.java`

```
package com.example.autosolver.dto;

// Map only what we need; keep it resilient to extra fields.
public class GenerateWebhookResponse {
  private String webhook;      // e.g. "https://bfhldevapigw.healthrx.co.in/
hiring/testWebhook/JAVA"
  private String accessToken;  // JWT

  public String getWebhook() { return webhook; }
  public void setWebhook(String webhook) { this.webhook = webhook; }
  public String getAccessToken() { return accessToken; }
  public void setAccessToken(String accessToken) { this.accessToken =
accessToken; }
}
```

`FinalQueryPayload.java`

```
package com.example.autosolver.dto;

public class FinalQueryPayload {
  private String finalQuery;
  public FinalQueryPayload() {}
  public FinalQueryPayload(String finalQuery) { this.finalQuery =
finalQuery; }
  public String getFinalQuery() { return finalQuery; }
  public void setFinalQuery(String finalQuery) { this.finalQuery =
finalQuery; }
}
```

🗻 Persistence

`Submission.java`

```
package com.example.autosolver.model;
```

```
import jakarta.persistence.*;
import java.time.Instant;

@Entity
public class Submission {
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;

  private String regNo;
  private String question; // "Q1" or "Q2"

  @Column(length = 8000)
  private String finalQuery;

  private Instant createdAt = Instant.now();

  public Submission() {}
  public Submission(String regNo, String question, String finalQuery) {
    this.regNo = regNo; this.question = question; this.finalQuery =
finalQuery;
  }

  public Long getId() { return id; }
  public String getRegNo() { return regNo; }
  public void setRegNo(String regNo) { this.regNo = regNo; }
  public String getQuestion() { return question; }
  public void setQuestion(String question) { this.question = question; }
  public String getFinalQuery() { return finalQuery; }
  public void setFinalQuery(String finalQuery) { this.finalQuery =
finalQuery; }
  public Instant getCreatedAt() { return createdAt; }
  public void setCreatedAt(Instant createdAt) { this.createdAt = createdAt; }
}
```

`SubmissionRepository.java`

```
package com.example.autosolver.repo;

import com.example.autosolver.model.Submission;
import org.springframework.data.jpa.repository.JpaRepository;

public interface SubmissionRepository extends JpaRepository<Submission,
Long> {}
```

Solvers

`SqlSolver.java`

```
package com.example.autosolver.service;

public interface SqlSolver {
  String questionCode(); // "Q1" or "Q2"
  String finalQuery();   // returns the SQL to submit
}
```

`Question1Solver.java`

```
package com.example.autosolver.service;

import org.springframework.core.io.ClassPathResource;
import java.nio.charset.StandardCharsets;

public class Question1Solver implements SqlSolver {
  @Override public String questionCode() { return "Q1"; }
  @Override public String finalQuery() {
    try {
      var res = new ClassPathResource("sql/question1.sql");
      return new String(res.getInputStream().readAllBytes(),
StandardCharsets.UTF_8).trim();
    } catch (Exception e) {
      throw new RuntimeException("Failed to load question1.sql", e);
    }
  }
}
```

`Question2Solver.java`

```
package com.example.autosolver.service;

import org.springframework.core.io.ClassPathResource;
import java.nio.charset.StandardCharsets;

public class Question2Solver implements SqlSolver {
  @Override public String questionCode() { return "Q2"; }
  @Override public String finalQuery() {
    try {
      var res = new ClassPathResource("sql/question2.sql");
      return new String(res.getInputStream().readAllBytes(),
StandardCharsets.UTF_8).trim();
    } catch (Exception e) {
      throw new RuntimeException("Failed to load question2.sql", e);
    }
```

```
  }
}
```

`SqlSolverFactory.java`

```
package com.example.autosolver.service;

public class SqlSolverFactory {
  public SqlSolver forRegNo(String regNo) {
    int lastTwo = extractLastTwoDigits(regNo);
    boolean isOdd = (lastTwo % 2) != 0;
    return isOdd ? new Question1Solver() : new Question2Solver();
  }

  private int extractLastTwoDigits(String regNo) {
    // Keep numbers only, then take last two.
    var digits = regNo.replaceAll("[^0-9]", "");
    if (digits.length() == 0) throw new IllegalArgumentException("regNo has
no digits: " + regNo);
    var slice = digits.substring(Math.max(0, digits.length() - 2));
    return Integer.parseInt(slice);
  }
}
```

HTTP Client

`WebhookClient.java`

```
package com.example.autosolver.service;

import com.example.autosolver.dto.FinalQueryPayload;
import com.example.autosolver.dto.GenerateWebhookRequest;
import com.example.autosolver.dto.GenerateWebhookResponse;
import org.slf4j.Logger; import org.slf4j.LoggerFactory;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;

@Component
public class WebhookClient {
  private static final Logger log =
LoggerFactory.getLogger(WebhookClient.class);
  private final WebClient webClient;

  public WebhookClient(WebClient.Builder builder) {
    this.webClient = builder.build();
```

```
  }

  public GenerateWebhookResponse generate(String baseUrl, String language,
GenerateWebhookRequest req) {
    String url = baseUrl + "/hiring/generateWebhook/" + language;
    log.info("POST {}", url);
    return webClient.post()
      .uri(url)
      .contentType(MediaType.APPLICATION_JSON)
      .bodyValue(req)
      .retrieve()
      .bodyToMono(GenerateWebhookResponse.class)
      .block();
  }

  public String submitFinalQuery(String webhookUrl, String jwt, String
finalQuery) {
    log.info("Submitting final query to webhook: {}", webhookUrl);
    return webClient.post()
      .uri(webhookUrl)
      .contentType(MediaType.APPLICATION_JSON)
      .header("Authorization", jwt) // API expects raw token (per spec). If
it requires Bearer, use "Bearer " + jwt.
      .bodyValue(new FinalQueryPayload(finalQuery))
      .retrieve()
      .bodyToMono(String.class)
      .onErrorResume(ex -> Mono.just("ERROR: " + ex.getMessage()))
      .block();
  }
}
```

If the API actually requires `Authorization: Bearer <token>`, switch the header line to:
`.header("Authorization", "Bearer " + jwt)`

🧭 Orchestrator (runs on startup)

`StartupOrchestrator.java`

```
package com.example.autosolver.service;

import com.example.autosolver.config.AppProperties;
import com.example.autosolver.dto.GenerateWebhookRequest;
import com.example.autosolver.dto.GenerateWebhookResponse;
import com.example.autosolver.model.Submission;
import com.example.autosolver.repo.SubmissionRepository;
import org.slf4j.Logger; import org.slf4j.LoggerFactory;
import org.springframework.boot.ApplicationRunner;
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.boot.context.properties.EnableConfigurationProperties;

@Configuration
@EnableConfigurationProperties(AppProperties.class)
public class StartupOrchestrator {
  private static final Logger log =
LoggerFactory.getLogger(StartupOrchestrator.class);

  @Bean
  ApplicationRunner run(WebhookClient client, AppProperties props,
SubmissionRepository repo) {
    return args -> {
      // 1) Generate webhook
      GenerateWebhookResponse gw = client.generate(
          props.getBaseUrl(), props.getLanguage(),
          new GenerateWebhookRequest(props.getName(), props.getRegNo(),
props.getEmail())
      );
      if (gw == null || gw.getWebhook() == null || gw.getAccessToken() ==
null) {
        throw new IllegalStateException("GenerateWebhook response missing
webhook/accessToken");
      }
      log.info("Got webhook: {}", gw.getWebhook());

      // 2) Choose solver by regNo last-two-digits parity
      SqlSolver solver = new SqlSolverFactory().forRegNo(props.getRegNo());
      String finalQuery = solver.finalQuery();
      log.info("Using {} with SQL length={} chars", solver.questionCode(),
finalQuery.length());

      // 3) Persist locally
      Submission saved = repo.save(new Submission(props.getRegNo(),
solver.questionCode(), finalQuery));
      log.info("Saved submission id={} at {}", saved.getId(),
saved.getCreatedAt());

      // 4) Submit to webhook with JWT in Authorization
      String result = client.submitFinalQuery(gw.getWebhook(),
gw.getAccessToken(), finalQuery);
      log.info("Webhook submit result: {}", result);
    };
  }
}
```

🛥️SQL Files (edit with your real answers)

`src/main/resources/sql/question1.sql`

```
-- TODO: Replace with your final SQL for Question 1
-- This file is loaded and submitted automatically when regNo last-two-digits
are ODD
SELECT 1 AS placeholder_q1;
```

`src/main/resources/sql/question2.sql`

```
-- TODO: Replace with your final SQL for Question 2
-- This file is loaded and submitted automatically when regNo last-two-digits
are EVEN
SELECT 2 AS placeholder_q2;
```

🛕Local Run

```
# Build
mvn -q -DskipTests package

# Run
java -jar target/auto-solver-0.0.1-SNAPSHOT.jar
```

Customize `application.yml` with your `name`, `regNo`, `email`.

Open H2 console (optional) at `http://localhost:8080/h2-console` to inspect the stored submission.
JDBC URL: `jdbc:h2:mem:autosolver`
User: `sa` Password: *(blank)*

🚚JWT Header Note

The spec says: *"Use JWT in the Authorization header for the second API."*
If the service expects the raw token (as some internal gateways do), the code already sends:

```
Authorization: <accessToken>
```

If it actually requires the standard scheme, change to:

```
Authorization: Bearer <accessToken>
```

Just flip the header line in `WebhookClient.submitFinalQuery()`.

Submission Checklist (what to upload)

- **Public GitHub repo** (example format):
- `https://github.com/your-username/auto-solver.git`
- **Include built JAR** in `target/` and provide the **RAW downloadable link** to it in your README.
- **README** should include:
- How it works (the 4 steps above)
- How to configure `application.yml`
- How to run/build
- Sample logs (redact tokens)

Where to put the actual SQL answers?

- Place your final SQL into `src/main/resources/sql/question1.sql` and `question2.sql`.
- The app automatically picks the right file based on **regNo parity of last two digits**.

Error Handling & Observability

- Basic validation on config
- Defensive null checks on API response
- Logs each major step & response
- Persists the exact SQL submitted for audit/retry

Switching to RestTemplate (if required)

If your reviewer insists on `RestTemplate`, you can add this bean and swap calls (the rest of the design stays the same):

```
@Bean RestTemplate restTemplate(RestTemplateBuilder b) { return b.build(); }
```

Then implement a similar `WebhookClientRestTemplate` using `postForObject` / `exchange`.

Notes

- Java 17, Spring Boot 3.3.x
- No controllers are exposed; the flow is entirely startup-driven.
- H2 is in-memory; switch to file DB if you want persistence across runs.

🚢README.md (include in your repo)

```
# Spring Boot Webhook Auto-Solver

## Build & Run
```bash
mvn -q -DskipTests package && java -jar target/auto-solver-0.0.1-SNAPSHOT.jar
```

Configure

Edit `src/main/resources/application.yml` (name, regNo, email).

How it works

1. POST `${baseUrl}/hiring/generateWebhook/${language}` with `{name, regNo, email}`
2. Receive `{webhook, accessToken}`
3. Choose Q1/Q2 from `regNo` last-two-digits parity; load SQL from `/sql/question*.sql`
4. Save to H2 and POST `{finalQuery: "..."}` to `webhook` with `Authorization: <accessToken>`

Deliverables

- GitHub: `https://github.com/<you>/auto-solver.git`
- Public JAR (raw link): `https://raw.githubusercontent.com/<you>/auto-solver/main/target/auto-solver-0.0.1-SNAPSHOT.jar`
- Submit here: https://forms.office.com/r/5Kzb1h7fre ```

Happy shipping! 🧑‍🚀
```