Online Bill Pay – Spring Boot (Java) Starter

A minimal but complete REST API for online bill pay using **Spring Boot**, **Spring Data JPA**, and an in-memory **H2** database. It includes authentication (HTTP Basic for demo), entities for Payees/Bills/Payments, and seed data. You can run it with Maven and curl/Postman immediately.

> **Note**: This is a demo. Replace Basic Auth with JWT/OAuth2 and move to a persistent database (PostgreSQL/MySQL) before production.

---

1) `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>billpay</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>billpay</name>
  <description>Online Bill Pay – Spring Boot Starter</description>
  <properties>
    <java.version>17</java.version>
    <spring-boot.version>3.3.2</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-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-validation</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </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>
      </plugin>
    </plugins>
  </build>
</project>
```

---

2) `src/main/resources/application.properties`

```
spring.datasource.url=jdbc:h2:mem:billpay;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.h2.console.enabled=true
spring.h2.console.path=/h2

# change in prod!
app.demoUser.username=user
app.demoUser.password=password
```

---

3) `src/main/java/com/example/billpay/BillPayApplication.java`

Single file with package-private classes for brevity. You can split them later.

```
package com.example.billpay;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.http.HttpStatus;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.validation.annotation.Validated;
import org.springframework.web.server.ResponseStatusException;
import org.springframework.web.bind.annotation.*;

import jakarta.persistence.*;
import jakarta.validation.constraints.*;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.OffsetDateTime;
import java.util.List;
import java.util.Optional;

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

    // ===== Security (Demo: HTTP Basic) =====
    @Bean PasswordEncoder passwordEncoder() { return new
BCryptPasswordEncoder(); }

    @Bean InMemoryUserDetailsManager
users(org.springframework.core.env.Environment env, PasswordEncoder
encoder) {
        String u = env.getProperty("app.demoUser.username", "user");
        String p = env.getProperty("app.demoUser.password",
"password");
        UserDetails user =
User.withUsername(u).password(encoder.encode(p)).roles("USER").build();
        return new InMemoryUserDetailsManager(user);
    }

    @Bean SecurityFilterChain security(HttpSecurity http) throws
Exception {
        http.csrf(csrf -> csrf.disable());
        http.authorizeHttpRequests(req -> req
                .requestMatchers("/h2/**").permitAll()
                .requestMatchers("/actuator/**").permitAll()
                .anyRequest().authenticated());
        http.httpBasic(Customizer.withDefaults());
        http.headers(h -> h.frameOptions(f -> f.disable())); // for
H2 console
        return http.build();
    }

    // ===== Data seed =====
    @Bean CommandLineRunner seed(PayeeRepo payees, BillRepo bills) {
        return args -> {
            if (payees.count() == 0) {
                Payee p1 = new Payee(null, "Electric Co.",
"ACC-1001");
                Payee p2 = new Payee(null, "Water Utility",
"ACC-2002");
                Payee p3 = new Payee(null, "Internet ISP",
"ACC-3003");
                payees.saveAll(List.of(p1, p2, p3));
            }
            if (bills.count() == 0) {
                Payee electric = payees.findByName("Electric
Co.").orElseThrow();
                bills.save(new Bill(null, "user", electric, new
BigDecimal("89.45"), LocalDate.now().plusDays(7), BillStatus.DUE));
            }
        };
    }
}

// ===== Entities =====
@Entity class Payee {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Long id;
    @Column(nullable = false) String name;
    @Column(nullable = false) String accountNumber;

    public Payee() {}
    public Payee(Long id, String name, String accountNumber) {
this.id=id; this.name=name; this.accountNumber=accountNumber; }
    public Long getId() { return id; }
    public String getName() { return name; }
    public String getAccountNumber() { return accountNumber; }
    public void setId(Long id) { this.id=id; }
    public void setName(String name) { this.name=name; }
    public void setAccountNumber(String accountNumber) {
this.accountNumber=accountNumber; }
}

enum BillStatus { DUE, SCHEDULED, PAID }

enum PaymentStatus { SCHEDULED, COMPLETED, FAILED }

@Entity class Bill {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Long id;
    @Column(nullable = false) String username; // demo: owner of bill
    @ManyToOne(optional = false, fetch = FetchType.EAGER) Payee
payee;
    @Column(nullable = false, precision = 12, scale = 2) BigDecimal
amount;
    @Column(nullable = false) LocalDate dueDate;
    @Enumerated(EnumType.STRING) @Column(nullable = false)
BillStatus status = BillStatus.DUE;

    public Bill() {}
    public Bill(Long id, String username, Payee payee, BigDecimal
amount, LocalDate dueDate, BillStatus status) {
        this.id=id; this.username=username; this.payee=payee;
this.amount=amount; this.dueDate=dueDate; this.status=status;
    }
    public Long getId() { return id; }
    public String getUsername() { return username; }
    public Payee getPayee() { return payee; }
    public BigDecimal getAmount() { return amount; }
    public LocalDate getDueDate() { return dueDate; }
    public BillStatus getStatus() { return status; }
    public void setId(Long id) { this.id=id; }
    public void setUsername(String username) {
this.username=username; }
    public void setPayee(Payee payee) { this.payee=payee; }
    public void setAmount(BigDecimal amount) { this.amount=amount; }
    public void setDueDate(LocalDate dueDate) {
this.dueDate=dueDate; }
    public void setStatus(BillStatus status) { this.status=status; }
}

@Entity class Payment {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY) Long id;
    @ManyToOne(optional = false, fetch = FetchType.EAGER) Bill bill;
    @Column(nullable = false, precision = 12, scale = 2) BigDecimal
amount;
    @Column(nullable = false) LocalDate scheduledDate;
    @Enumerated(EnumType.STRING) @Column(nullable = false)
PaymentStatus status;
    @Column(nullable = false) OffsetDateTime createdAt =
OffsetDateTime.now();

    public Payment() {}
    public Payment(Long id, Bill bill, BigDecimal amount, LocalDate
scheduledDate, PaymentStatus status) {
        this.id=id; this.bill=bill; this.amount=amount;
this.scheduledDate=scheduledDate; this.status=status;
    }
    public Long getId() { return id; }
    public Bill getBill() { return bill; }
    public BigDecimal getAmount() { return amount; }
    public LocalDate getScheduledDate() { return scheduledDate; }
    public PaymentStatus getStatus() { return status; }
    public OffsetDateTime getCreatedAt() { return createdAt; }
    public void setId(Long id) { this.id=id; }
    public void setBill(Bill bill) { this.bill=bill; }
    public void setAmount(BigDecimal amount) { this.amount=amount; }
    public void setScheduledDate(LocalDate scheduledDate) {
this.scheduledDate=scheduledDate; }
```

```
    public void setStatus(PaymentStatus status) {
this.status=status; }
    public void setCreatedAt(OffsetDateTime createdAt) {
this.createdAt=createdAt; }
}

// ===== Repositories =====
interface PayeeRepo extends JpaRepository<Payee, Long> {
    Optional<Payee> findByName(String name);
}
interface BillRepo extends JpaRepository<Bill, Long> {
    List<Bill> findByUsername(String username);
}
interface PaymentRepo extends JpaRepository<Payment, Long> {
    List<Payment> findByBill_Id(Long billId);
}

// ===== DTOs =====
record CreatePayeeRequest(@NotBlank String name, @NotBlank String
accountNumber) {}
record CreateBillRequest(@NotNull Long payeeId, @NotNull
@DecimalMin("0.01") BigDecimal amount, @NotNull LocalDate dueDate) {}
record CreatePaymentRequest(@NotNull Long billId, @NotNull
@DecimalMin("0.01") BigDecimal amount, @NotNull LocalDate
scheduledDate) {}

// ===== Controllers =====
@RestController
@RequestMapping("/api/payees")
class PayeeController {
    private final PayeeRepo repo;
    PayeeController(PayeeRepo repo) { this.repo = repo; }

    @GetMapping List<Payee> list() { return repo.findAll(); }
    @PostMapping @ResponseStatus(HttpStatus.CREATED)
    Payee create(@Validated @RequestBody CreatePayeeRequest req) {
        return repo.save(new Payee(null, req.name(),
req.accountNumber()));
    }
}

@RestController
@RequestMapping("/api/bills")
class BillController {
    private final BillRepo bills; private final PayeeRepo payees;
    BillController(BillRepo bills, PayeeRepo payees) {
this.bills=bills; this.payees=payees; }
```

```
    @GetMapping List<Bill> myBills(@RequestHeader("Authorization")
String auth) {
        String username = "user"; // demo: single user from Basic
Auth
        return bills.findByUsername(username);
    }

    @PostMapping @ResponseStatus(HttpStatus.CREATED)
    Bill create(@Validated @RequestBody CreateBillRequest req) {
        Payee p = payees.findById(req.payeeId()).orElseThrow(() ->
new ResponseStatusException(HttpStatus.NOT_FOUND, "Payee not
found"));
        Bill b = new Bill(null, "user", p, req.amount(),
req.dueDate(), BillStatus.DUE);
        return bills.save(b);
    }
}

@RestController
@RequestMapping("/api/payments")
class PaymentController {
    private final PaymentRepo payments; private final BillRepo bills;
    PaymentController(PaymentRepo payments, BillRepo bills) {
this.payments=payments; this.bills=bills; }

    @GetMapping List<Payment> list() { return payments.findAll(); }

    @PostMapping @ResponseStatus(HttpStatus.CREATED)
    Payment pay(@Validated @RequestBody CreatePaymentRequest req) {
        Bill b = bills.findById(req.billId()).orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Bill not found"));
        if (b.getStatus() == BillStatus.PAID) throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Bill already paid");
        if (req.amount().compareTo(b.getAmount()) != 0) throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Amount must equal
bill amount in demo");

        Payment p = new Payment(null, b, req.amount(),
req.scheduledDate(), PaymentStatus.SCHEDULED);
        // demo: execute immediately if scheduled today or earlier
        if (!req.scheduledDate().isAfter(LocalDate.now())) {
            p.setStatus(PaymentStatus.COMPLETED);
            b.setStatus(BillStatus.PAID);
            bills.save(b);
        }
        return payments.save(p);
    }
}
```

4) How to run

```
# from project root (where pom.xml lives)
mvn spring-boot:run
```

App starts on `http://localhost:8080` and H2 console at `http://localhost:8080/h2`.

Demo Auth (change in `application.properties`):

```
username: user
password: password
```

5) Quick API test (curl)

```
# list payees
curl -u user:password http://localhost:8080/api/payees

# create payee
curl -u user:password -H 'Content-Type: application/json' -d '{
  "name": "Mortgage Lender",
  "accountNumber": "M-445566"
}' http://localhost:8080/api/payees

# list my bills
curl -u user:password http://localhost:8080/api/bills

# create a bill (use an actual payeeId from /api/payees)
curl -u user:password -H 'Content-Type: application/json' -d '{
  "payeeId": 1,
  "amount": 120.50,
  "dueDate": "2025-09-01"
}' http://localhost:8080/api/bills

# pay a bill (use an actual billId)
curl -u user:password -H 'Content-Type: application/json' -d '{
  "billId": 1,
  "amount": 89.45,
  "scheduledDate": "2025-08-24"
}' http://localhost:8080/api/payments
```

```
# list payments
curl -u user:password http://localhost:8080/api/payments
```

---

6) Next steps (hardening & features)

- Replace Basic Auth with JWT/OAuth2 and add user registration & password hashing
- Multi-user ownership on bills/payments (derive username from auth principal)
- Add idempotency keys to payment creation
- Integrate a real payment processor (Stripe/Adyen/PayPal) via webhooks
- Add recurring payments & reminders (scheduled jobs)
- Switch H2 → PostgreSQL/MySQL; add Flyway migrations
- Validation rules (grace periods, partial payments, fees)
- Observability (actuator metrics, logs, tracing)
- API docs (OpenAPI/Swagger)
- Frontend (React/Vue) or mobile client
