PRICE DROP TRACKER

DESIGN DOCUMENT // SEVEN 30 GAMES // JAVA + SPRING BOOT

Spring Boot | PostgreSQL | Jsoup | JWT Auth | Spring Scheduler | REST API

// PROJECT OVERVIEW

What We're Building

A full-stack price tracking tool that lets users paste any product URL, set a target price, and get notified the moment the price drops. Real users, real value, and a backend that actually has to do work — scraping, scheduling, alerting, and analytics.

WHY THIS PROJECT

This is not a CRUD app dressed up. It has a scraping engine, background job scheduling, event-driven notifications, time-series data, and real business logic. Every piece is a talking point in an interview.

TIME ESTIMATE

MVP in 2–3 weeks working part-time. Core scraper + scheduler + notifications = Week 1. Auth + frontend polish + deployment = Week 2. Price history analytics + bookmarklet = Week 3.

TARGET AUDIENCE

Anyone who shops online — which is everyone. Particularly useful for devs, gamers tracking hardware drops, or deal hunters. Self-hostable = privacy-conscious users love it.

// TECH STACK

Chosen Technologies

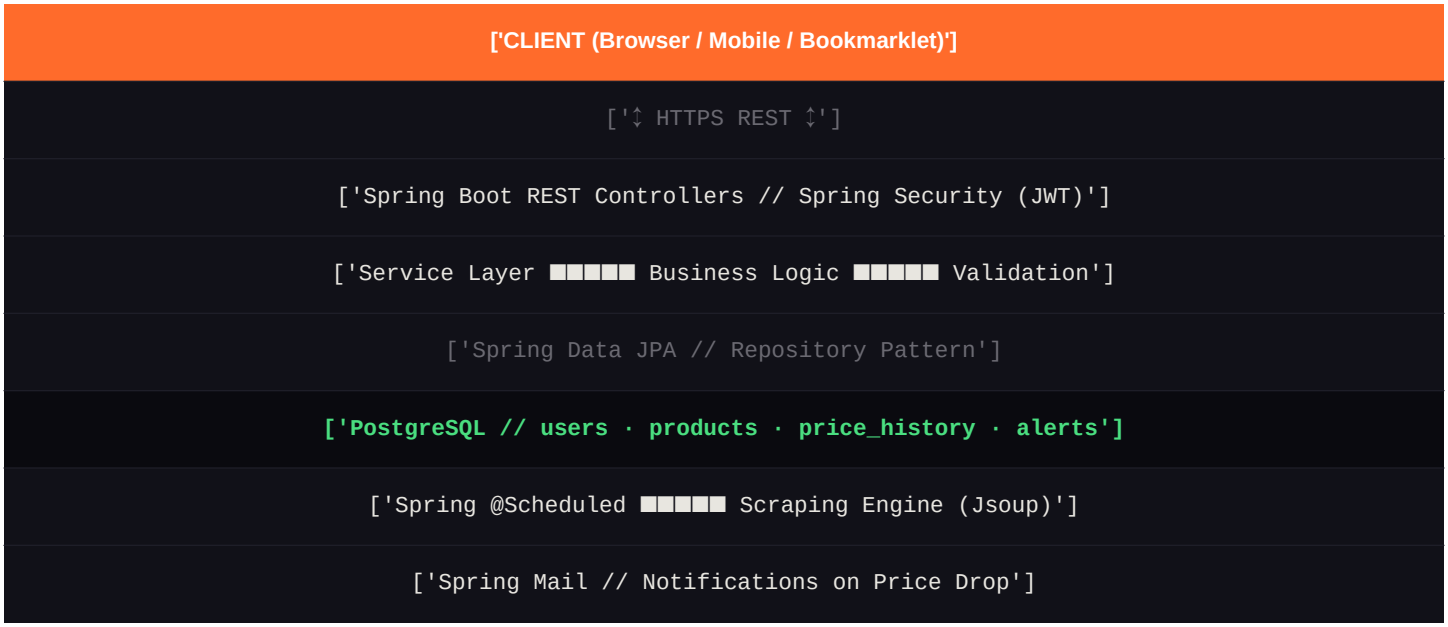
Layer	Technology	Why

Framework	`Spring Boot 3.x`	Industry standard, great for REST + scheduling
Language	`Java 21`	Virtual threads, modern records, pattern matching
Database	`PostgreSQL`	Time-series price data, relational user/product models
ORM	`Spring Data JPA + Hibernate`	Clean repository pattern, no boilerplate
Scraping	`Jsoup`	HTML parsing, selector-based extraction
Scheduling	`Spring @Scheduled`	Built-in, cron-based price polling
Auth	`Spring Security + JWT`	Stateless, recruiter-friendly, industry standard
Email	`Spring Mail + Mailgun`	Free tier, reliable delivery
Cache	`Redis (optional)`	Rate limit scrapers, cache recent prices
Deployment	`Railway or Render`	Free tier, one-click PostgreSQL, instant HTTPS
Build	`Maven`	Standard, widely understood

// SYSTEM ARCHITECTURE

How It All Fits Together

The system is split into four distinct layers: the REST API layer handling user requests, the scraping engine running in the background, the notification engine reacting to price changes, and the data layer persisting everything in PostgreSQL.



// DATA MODELS

Database Schema

users

id UUID // PK
email VARCHAR(255) // UNIQUE
password_hash VARCHAR(255)
notification_email VARCHAR(255)
created_at TIMESTAMP
is_active BOOLEAN

products

id UUID // PK
user_id UUID // FK → users
url TEXT
name VARCHAR(500)
retailer VARCHAR(100)
target_price DECIMAL(10,2)
is_active BOOLEAN
created_at TIMESTAMP

price_history

```
id UUID // PK
product_id UUID // FK → products
price DECIMAL(10,2)
currency CHAR(3)
scraped_at TIMESTAMP
in_stock BOOLEAN
raw_html_hash CHAR(64)
```

alerts

```
id UUID // PK
product_id UUID // FK → products
triggered_price DECIMAL(10,2)
triggered_at TIMESTAMP
notification_sent BOOLEAN
alert_type VARCHAR(50)
```

// REST API DESIGN

Endpoints

Authentication

POST	/api/auth/register	Register new user, returns JWT
POST	/api/auth/login	Login, returns JWT + refresh token
POST	/api/auth/refresh	Refresh access token

Products (Tracked Items)

GET	/api/products	List all tracked products for user
POST	/api/products	Add new product URL to track
GET	/api/products/{id}	Get single product + price history
PUT	/api/products/{id}	Update target price or settings
DELETE	/api/products/{id}	Stop tracking a product
POST	/api/products/{id}/check	Manually trigger a price check

Price History + Analytics

GET	/api/products/{id}/history	Full price history (time-series)
GET	/api/products/{id}/history?days=30	Price history for last N days
GET	/api/products/{id}/stats	Lowest, highest, avg price + trend

Alerts

GET	/api/alerts	Get all triggered alerts for user
DELETE	/api/alerts/{id}	Dismiss an alert

// SCRAPING ENGINE

The Core Technical Challenge

The scraper is what makes this project interesting. Websites actively try to prevent scraping, so you need to handle that gracefully. Here's the approach:

RETAILER STRATEGY

Each retailer gets its own scraper strategy class implementing a common ScraperStrategy interface. Amazon, Best Buy, Walmart, Newegg each have different HTML structures. This is a textbook Strategy Pattern — great interview talking point.

```
public interface ScraperStrategy {
boolean canHandle(String url);
PriceResult scrape(String url) throws ScrapingException;
}
@Component
public class AmazonScraper implements ScraperStrategy {
public PriceResult scrape(String url) {
Document doc = Jsoup.connect(url)
.userAgent("Mozilla/5.0 ...")
.referrer("https://google.com")
.get();
String price = doc.select("#corePriceDisplay_desktop_feature_div
.a-price-whole").first().text();
return new PriceResult(parsePrice(price), "USD", true);
}
}
```

ANTI-BOT MEASURES

Rotate user agent strings. Add random delays between requests (500ms–3s). Respect robots.txt. If you get a 429, back off exponentially. Handle Cloudflare-protected pages gracefully (just mark as temporarily unavailable).

SCHEDULING

Use @Scheduled(cron = "0 */30 * * * *") to check prices every 30 minutes. Stagger checks so you don't hammer 500 URLs at once — process in batches with a delay. Log every scrape attempt with status for debugging.

// FEATURE LIST

What To Build (MVP vs V2)

MVP — Ship in 2–3 Weeks	V2 — The Polish Pass
✓ User registration + JWT auth	■ Price history chart (line graph)
✓ Add product URL to track	■ "Lowest in 30 days" badge
✓ Auto price check every 30 mins	■ Browser bookmarklet (one-click add)
✓ Email alert when price drops	■ Multiple alert types (% drop, stock)
✓ Price history stored in DB	■ Slack / Discord webhook notifications
✓ List + manage tracked products	■ Multi-retailer support (Best Buy etc)
✓ Manual "check now" trigger	■ Public shareable product link
✓ Basic REST API documentation	■ Admin dashboard for monitoring

// IMPLEMENTATION PLAN

Week-by-Week Breakdown

Week 1	**// The Engine**
	■ Spring Boot project setup, Maven config, PostgreSQL connection
	■ JPA entities: User, Product, PriceHistory, Alert
	■ Repository layer + Flyway migrations
	■ Jsoup scraper for Amazon — ScraperStrategy interface
	■ Spring @Scheduled price polling — test with 2-3 URLs
	■ Basic Spring Mail notification on price drop
Week 2	**// The API**

	☐ Spring Security config + JWT token filter
	☐ Auth endpoints: register, login, refresh
	☐ Product CRUD endpoints (all 6)
	☐ Price history endpoint with date filtering
	☐ Input validation, error handling, GlobalExceptionHandler
	☐ Deploy to Railway with prod PostgreSQL

Week 3	**// The Polish**
	☐ Price stats endpoint: min, max, avg, trend
	☐ Add Best Buy / Newegg scraper strategies
	☐ Browser bookmarklet (3 lines of JS, huge UX win)
	☐ Simple React or plain HTML frontend with price chart
	☐ README with architecture diagram, setup instructions, live demo link
	☐ Postman collection exported + documented

// PROJECT STRUCTURE

Spring Boot Package Layout

```
com.seven30games.pricetracker
■■■ config/
■ ■■■ SecurityConfig.java # Spring Security + JWT setup
■ ■■■ SchedulerConfig.java # Scheduling thread pool
■ ■■■ WebConfig.java # CORS config
■■■ controller/
■ ■■■ AuthController.java
■ ■■■ ProductController.java
■ ■■■ AlertController.java
■■■ service/
■ ■■■ AuthService.java
■ ■■■ ProductService.java
■ ■■■ PriceCheckService.java # Orchestrates scraping
■ ■■■ NotificationService.java # Email alerts
■■■ scraper/
■ ■■■ ScraperStrategy.java # Interface
■ ■■■ ScraperDispatcher.java # Picks correct strategy
■ ■■■ AmazonScraper.java
■ ■■■ BestBuyScraper.java
■ ■■■ GenericScraper.java # Fallback
■■■ scheduler/
■ ■■■ PriceCheckScheduler.java # @Scheduled jobs
■■■ model/
■ ■■■ User.java
■ ■■■ Product.java
■ ■■■ PriceHistory.java
■ ■■■ Alert.java
■■■ repository/
■ ■■■ UserRepository.java
■ ■■■ ProductRepository.java
■ ■■■ PriceHistoryRepository.java
■■■ dto/
```

```
■ ■■■ ProductRequest.java
■ ■■■ PriceHistoryResponse.java
■ ■■■ AuthRequest.java
■■■ exception/
■ ■■■ GlobalExceptionHandler.java
■ ■■■ ScrapingException.java
■■■ security/
■■■ JwtTokenProvider.java
■■■ JwtAuthFilter.java
```

// INTERVIEW TALKING POINTS

What To Brag About

Strategy Pattern	The scraper uses the Strategy pattern — each retailer is a separate class implementing ScraperStrategy. The dispatcher picks the right one based on the URL. Clean, extensible, SOLID.
Concurrency & Scheduling	Spring Scheduler runs background jobs without blocking the API. Price checks are batched to avoid hammering external servers. Shows you understand async execution.
Time-Series Data Design	price_history is append-only — never update, always insert. That's how you model time-series data correctly. You can then query for min/max/avg over any time window.
Real-World Engineering	Handling scraping failures gracefully, exponential backoff on 429s, logging every attempt, marking products as temporarily unavailable — this is production thinking, not tutorial thinking.
JWT Auth	Stateless authentication with access + refresh tokens. Spring Security filter chain. Most junior devs can't explain this in an interview — you can.
Deployed & Live	It's not a GitHub repo gathering dust — it's running on Railway with a real domain. You can demo it in 60 seconds. That ends interviews early in the best way.