Virtual Everywhere

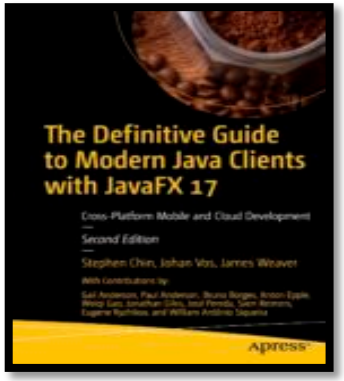
January 25, 2022

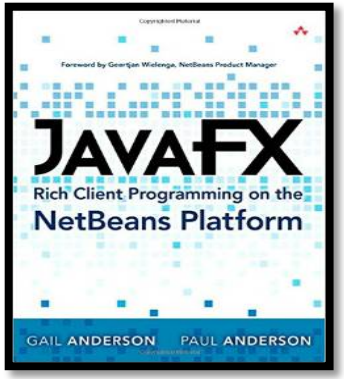
Front and Center!
JavaFX with Spring Boot

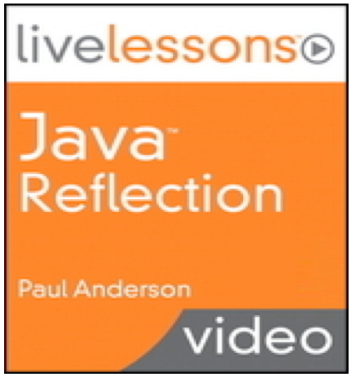
Paul Anderson
Gail Anderson
Anderson Software Group, Inc.
asgteach.com

So Who Are We?

- Training Company
  - Java, JavaFX Courses
- JavaFX Authors
  - Definitive Guide to JavaFX
  - JavaFX Rich Client Programming on the NetBeans Platform
- LiveLesson Videos
  - JavaFX Programming
  - Java Reflection







Agenda

- Why JavaFX?
- Why Spring Boot?
- Design Approach
- Gluon Ignite Demo
- JPA with H2, MySQL
- REST Service
- Music Demo, WebClient
- Reactive Stream Demo
- Wrap Up, Q & A

Why JavaFX?

- Modern Clients
  - Platform independent source code
  - “Write Once, Install Everywhere”
- JavaFX Features
  - Rich UI controls, graphics, media engines
  - Concurrency library for asynchronous tasks
- JavaFX Advantages
  - Java UI, scene graph, nodes, FXML views
  - Properties, listeners, binding, event handlers

Why Spring Boot?

- Advantages
  - Easy to use and understand
  - Reduces development time
- Benefits
  - Starter dependencies
  - Annotation based
  - Eases dependency management
  - Manages REST endpoints
  - Includes embedded servlet container

Spring Boot Starters

- What is a Starter?
  - Handles dependency management
  - Starter dependencies specified with Maven/Gradle
  - Adds jar files to classpath
  - Format: `spring-boot-starter-type`
- Examples
  - Web: `spring-boot-starter-web`
  - Test: `spring-boot-starter-test`
  - JPA: `spring-boot-starter-data-jpa`

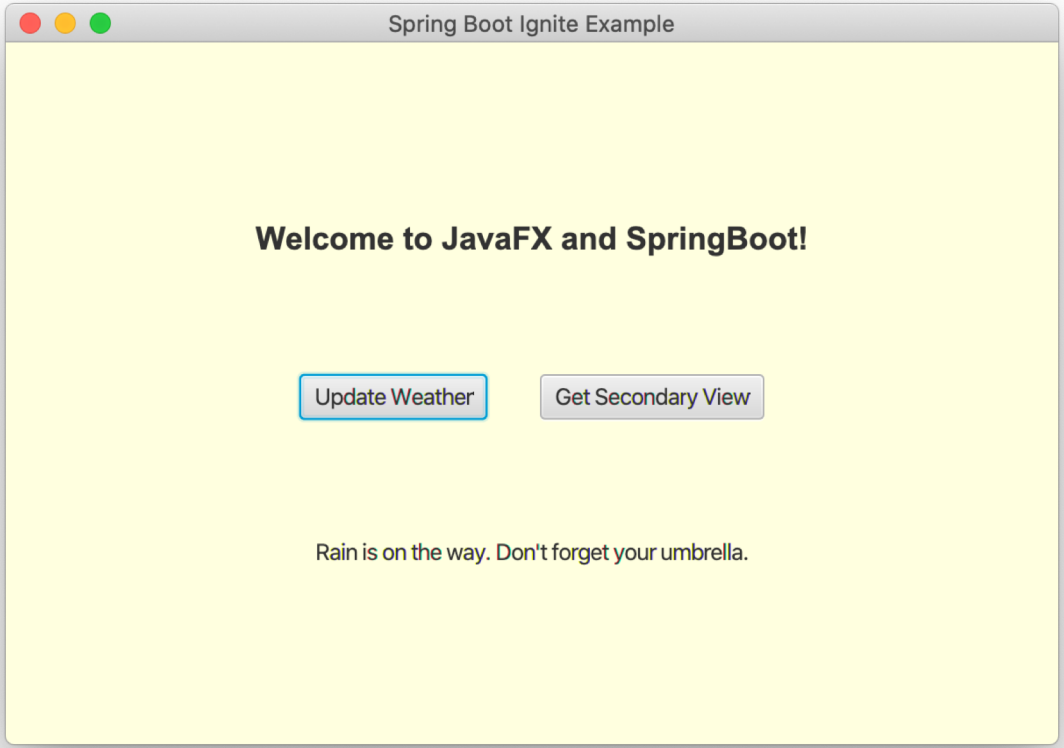
Design Approach

- Main Issues
  - JavaFX has its own lifecycle and controllers
  - FXML Loader not created and managed by Spring
- Integration Approach
  - Use Gluon Ignite libraries
  - Add JavaFX controller and FXML view
  - Spring Boot main application launches JavaFX
  - Both Ignite and Spring Boot contexts initialized
  - JavaFX loader builds scene graph

Gluon Ignite

- Why Use Gluon Ignite?
  - Supports popular DI frameworks
  - Allows DI in JavaFX applications
  - Also in FXML controllers
  - Supports multiple views
- How Do You Use Ignite?
  - Include as a dependency
  - Initializes the Spring Context

Ignite Demo



JPA Persistence

- What is JPA?
  - Java Persistence API
  - Defines entities, attributes, relationships
  - Provides Entity Manager, JPQL, Criteria API
- What is Hibernate?
  - Implements JPA with Object Relational Mappings
  - ORM framework on top of JPA
  - Provides mappings between tables and database
  - Handles exceptions and transactions

JPA Entities

- What is an Entity?
  - Java POJO that can be persisted to the database
  - Represents a table stored in a database
  - Every instance represents a *row* in the table
- JPA Annotations
  - `@Entity`, `@Table`, `@Version`
  - `@Id`, `@Column`, `@GeneratedValue`, `@Basic`
  - `@JoinColumn`, `@JoinTable`, `@OrderBy`
  - `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`

JpaRepository Interface

- What is `JpaRepository<T,ID>`?
  - Provided by Spring framework
  - Methods for CRUD operations, sorting, paging
- Using `JpaRepository<T,ID>`
  - Repository interface extends `JpaRepository<T,ID>`
  - Inherit JPA methods, define your own
  - `count()`, `findById()`, `findAll()`, `findAllById()`
  - `save()`, `saveAll()`, `existsById()`, `delete()`
  - `deleteById()`, `deleteAll()`, `deleteAllById()`

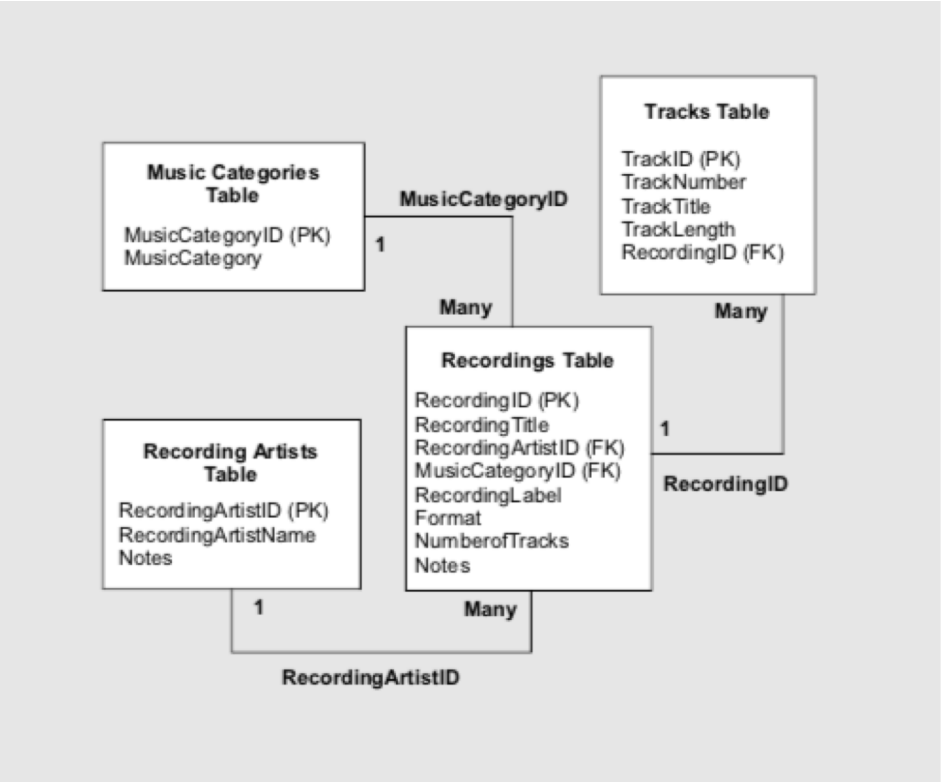
JavaFX, JPA with Spring Boot

- Server Setup
  - Starter dependencies, Application properties
  - H2 or MySQL Database
  - Domain Entities, JPA Repository
  - REST Controllers
- Client Setup
  - Spring Boot Application with JavaFX
  - Domain POJOs, REST Service
  - View Controller

REST Service

- What is a REST Service?
  - Producer/consumer with Service resources
  - Service is stateless and cacheable
  - Clients use middle-layer for Service
- REST, HTTP with `@RestController`
  - `@GetMapping` : read resource
  - `@PutMapping` : update existing resource
  - `@PostMapping` : create new resource
  - `@DeleteMapping` : delete resource

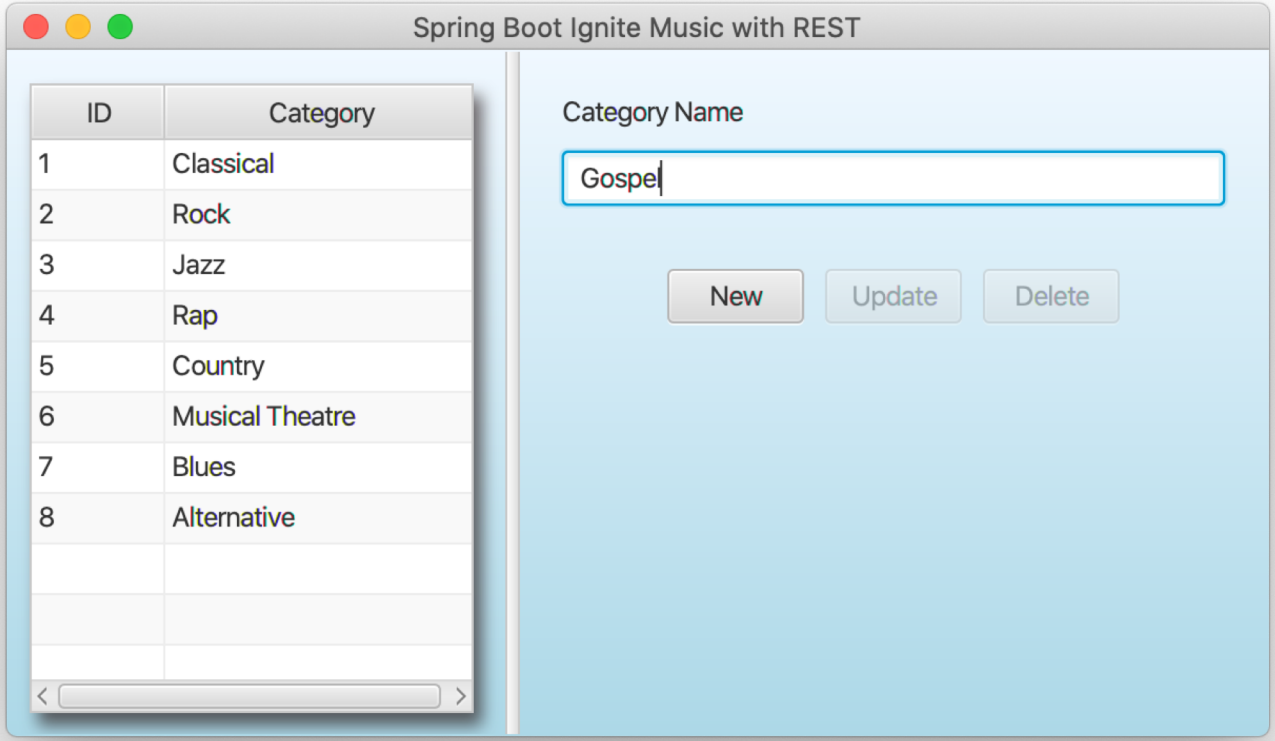
Music Service Demo



WebClient

- What is `WebClient`?
  - Replaces `RestTemplate` in Spring 5
  - Reactive streams approach
  - Provides blocking, non-blocking modes
- Using `WebClient`
  - Dependency `spring-boot-starter-webflux`
  - Inject builder, retrieve with `Mono`, `Flux` wrappers
  - Consume REST service with `retrieve()`
  - Use `block()` for synchronous retrieves

Music Demo



Reactive Streams

- What are Reactive Streams?
  - Reactive Core Java 8 library
  - Provides asynchronous stream processing
  - Publish-Subscribe model
- Using Reactive Streams
  - Publisher interface includes `Flux` and `Mono`
  - Subscribers request data from the stream
  - Events are pushed to subscribers
  - Method intervals, transforms, back pressure drops

Reactive Demo

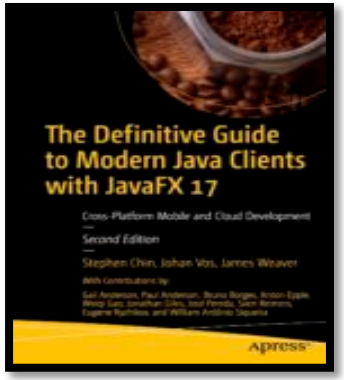
Summary

- JavaFX and Spring Boot
  - All Java stack for development
  - Gluon Ignite for integration
  - Separates UI from backend operations
- JPA, REST, WebClient, Reactive
  - Spring Boot simplifies JPA, REST services
  - WebClient for non-reactive systems
  - Reactive clients and servers

Wrap Up

- Thanks for Attending!

paul@asgteach.com @paul_asgteach
gail@asgteach.com @gail_asgteach



- GitHub Source Code

https://github.com/gailasgteach/JavaFX-SpringBoot-Samples

- Q & A

