**SOLID Principles Evaluation Report**

**Project Name**: *Live Betting Application - Bulletin Management*  
**Author**: *Doruk Eskicorapci*  
**Date**: June 19, 2025

**1. Single Responsibility Principle (SRP)**

**Definition**: Each class should have only one reason to change, meaning it should only have one job/responsibility.

**Did I follow this principle before the updates?**  
☐ Yes  ☐ Partially  ☐ No

**Good Implementation(s):**

1. *Class Name(s):WrongArgumentException, MissingArgumentException*  
   *Responsibility: Handle errors for specified cases.*
2. *Class Name(s): ApiExceptionHandler*  
   *Responsibility: Single responsibility of handling exceptions globally*

**Violations and Issues**

1. *Class Name(s):* *ScheduledTasks Class*  
   *Explanation: Multiple Responsibility: Updating info every second, generating valid odd values.*

*How to improve: Move generateValidOdds() function to EventService to keep business logic in one file.*

*Did I apply the improvement: YES*

1. *Class Name(s):* *EventService Class*  
   *Explanation: Multiple Responsibility: Validation, Odds Generation, Conversion*

*How to improve: Create and move different functions into different service files*

*Did I apply the improvement: YES*

*Extra notes:* After applying the suggested improvements, AI is telling me ‘EventValidationService’ is handling two jobs: validation and update. However I do not know how to separate them.

1. *Class Name(s):* *ScheduledTasks*  
   *Explanation: Handling scheduling and updating*

*How to improve: A new service file can be created to have the responsibilty of updating event odds which will be called by scheduler.*

*Did I apply the improvement: YES*

**SRP Compliance Score Given by AI After the Updates: 6/10**

**SRP Compliance Score Given by AI After the Updates: 10/10**

**2. Open/Closed Principle (OCP)**

**Definition**: Classes should be open for extension but closed for modification.

**Did I follow this principle?**  
☐ Yes  ☐ Partially  ☐ No

**Good Implementation(s):**

1. *Class Name(s): ApiExceptionHandler (@ControllerAdvice)*  
   *Explanation: Controller advice is open for extension. New exceptions can be easily added.*
2. *Class Name(s): EventErrorResponse*  
   *Explanation: Consistent response stucture*

**Violations and Issues**

1. *Class Name(s):* EventValidationService >> validateAndApplyUpdates()  
   *Explanation:* Hard-coded field handling - closed for extension

*How to improve: Do validation in different class or function so that in the future developers can handle codes without needing to change the main code.*

*Did I apply the improvement: NO*

1. *Class Name(s):* EventValidationService >> validateRequiredFields()  
   *Explanation: If conditions are making the code more difficult for extensions.*

*How to improve: Add new validators by implementing `RequiredFieldValidator`*

*Did I apply the improvement: NO*

1. *Class Name(s):* EventValidationService  
   *Explanation: Single class handles all validation types - not extensible. - Cannot plug in new validation rules without modifying the core class*

*How to improve: Split into separate validation strategies managed by a central orchestrator to make it extensible. Replace hard-coded field updates with a handler system for dynamic field processing. Use a result object to simplify adding new validation rules. `EventValidationService`* s*plit into separate validation strategies managed by a central orchestrator to make it extensible. Replace hard-coded field updates with a handler system. Use a result object to simplify adding new validation rules. `EventValidationService`*

*Did I apply the improvement: NO*

1. *Class Name(s):* ApiExceptionHandler   
   *Explanation: The limitation in \*\*`ApiExceptionHandler`\*\* arises because each exception type (e.g., , ) is handled in separate methods, resulting in duplicated code and inconsistency in handling `WrongArgumentException``MissingArgumentException`*

*How to improve: 1. Add a more reusable exception-handling mechanism to cover multiple exception cases uniformly*

*Did I apply the improvement: YES*

**SRP Compliance Score Given by AI After the Updates: 7.5/10**

**3. Liskov Substitution Principle (LSP)**

**Definition**: Objects of a superclass should be replaceable with objects of its subclasses without altering the correctness of the program.

**Did I follow this principle?**  
☐ Yes  ☐ Partially  ☐ No

**Good Implementation(s):**

1. *Class Name(s): ApiExceptionHandler and it’s child classes*  
   *Explanation: Controller advice is open for extension. New exceptions can be easily added.*
2. *Class Name(s): EventErrorResponse*  
   *Explanation: Extends Spring Data’s JpaRepository<Event, Integer> interface. Any other JpaRepository implementation could be swapped in and all callers would still work*

**LSP Compliance Score Given by AI After the Updates: 10/10**

**4. Interface Segregation Principle (ISP)**

**Definition:** Clients should not be forced to depend on interfaces they do not use; many client-specific interfaces are better than one general-purpose interface.

**Did I follow this principle?**  
☐ Yes  ☐ Partially  ☐ No

**Good Implementation(s):**

1. *Class Name(s): JpaRepository usage (EventRepository)*  
   *Explanation: JpaRepository already splits read-only and write operations through smaller interfaces (CrudRepository, PagingAndSortingRepository).*
2. *Class Name(s): OddsGeneratorService*  
   *Explanation: Exposes a single, well-focused method generateValidOdds()*

**Violations and Issues**

1. *Class Name(s):* EventService  
   *Explanation:* Despite the refactor, the service still couples CRUD, conversion, and validation logic. Consumers requiring only read operations are forced to depend on the entire service.

*How to improve: Create separate interfaces like EventReader (for getAllEvents/getEventById) and EventWriter (for save/update operations).*

*Did I apply the improvement: NO*

1. *Class Name(s):* EventValidationService  
   *Explanation:* Provides multiple unrelated validation helpers (validateEventId, validateOddRange, etc.). If another component needs only ID validation it must still depend on odds checks.

*How to improve: Create separate validator interfaces (IdValidator, OddsValidator) and inject only the specific validator needed instead of the entire EventValidationService.*

*Did I apply the improvement: NO*

1. *Class Name(s):* OddsUpdateService  
   *Explanation:* Depends on the whole EventService although it really needs only an “update odds” capability.

*How to improve: Create an EventUpdater interface with just the updateEvent method and make OddsUpdateService depend on that instead of the full EventService.*

*Did I apply the improvement: NO*

**ISP Compliance Score Given by AI After the Updates: 6/10**

**5. Dependency Inversion Principle (DIP)**

**Definition**: High-level modules should not depend on low-level modules. Both should depend on abstractions.

**Did I follow this principle?**  
☐ Yes  ☐ Partially  ☐ No

**Good Implementation(s):**

1. *Class Name(s): Spring Dependency Injection*

*Explanation: The project leverages constructor injection (@Autowired) so classes don’t create their own dependencies*

1. *Class Name(s): Spring Dependency Injection*  
   *Explanation: Spring Dependency Injection*

**Violations and Issues**

1. *Class Name(s):* EventRestController  
   *Explanation:* : Controller depends directly on concrete EventService class instead of an interface.

*How to improve: Create IEventService interface and make controller depend on that abstraction, allowing easy swapping of implementations.*

*Did I apply the improvement: NO*

1. *Class Name(s):* OddsUpdateService  
   *Explanation:* Depends on concrete EventService and OddsGeneratorService classes. This makes testing difficult and prevents easy substitution of different odds generation algorithms

*How to improve: : Define IEventService and IOddsGenerator interfaces, inject these abstractions to enable easy swapping of implementations without changing the service logic.*

*Did I apply the improvement: NO*

1. *Class Name(s):* EventService  
   *Explanation:* Calls concrete EventValidationService and EventConversionService without interface contracts. This creates rigid dependencies - you can't easily substitute different validation strategies or conversion logic.

*How to improve: Create IEventValidator and IEventConverter interfaces, make EventService depend on these abstractions to enable flexible validation and conversion strategies.*

*Did I apply the improvement: NO*

**ISP Compliance Score Given by AI After the Updates: 5/10**