BlockBid Auction System

Design Document Updates - Deliverable 2

Course: EECS 4413 - Building e-Commerce Systems
Team: Kyle Williamson
Date: November 2, 2025
Version: 2.0

---

1. Implementation Technology Change

1.1 Original Design (Deliverable 1)

In Deliverable 1, we proposed:

- **Backend:** Node.js with Express.js framework
- **Database:** MongoDB (catalogue), PostgreSQL (transactions)
- **Real-time:** Socket.io for bidding updates
- **Blockchain:** Ethereum smart contracts for UC8

1.2 Actual Implementation (Deliverable 2)

For Deliverable 2, we implemented:

- **Backend:** Java with Spring Boot 3.2.0
- **Database:** H2 in-memory database (development)
- **API Framework:** Spring Web with HATEOAS support
- **ORM:** Spring Data JPA with Hibernate

1.3 Justification for Change

Reasons for switching from Node.js to Java/Spring Boot:

1. **Development Expertise:** Stronger background in Java development
2. **Development Speed:** Faster implementation due to familiarity with Java ecosystem
3. **Spring Boot Benefits:**
    - Built-in HATEOAS support (marking scheme requirement)
    - Excellent JPA/Hibernate integration
    - Robust dependency injection
    - Easy transaction management
4. **Course Compatibility:** Project spec allows any language choice
5. **Deliverable 3 Preparation:** Spring Boot microservices work well with Docker/Kubernetes

From project specification:

> "You can implement the back-end subsystem in any language of your choice"

1.4 What Changed vs. What Stayed the Same

CHANGED (Implementation Details):

- Programming language: JavaScript → Java
- Framework: Express.js → Spring Boot
- Database: MongoDB/PostgreSQL → H2
- Dependencies: npm packages → Maven dependencies

STAYED THE SAME (Architecture & Design):

- (Done) Microservices architecture (service-based design)
- (Done) All use case logic and business rules
- (Done) Database schema and entity relationships
- (Done) REST API endpoint design
- (Done) HATEOAS implementation
- (Done) All sequence diagrams from Deliverable 1
- (Done) All activity diagrams from Deliverable 1
- (Done) Component architecture (User, Catalogue, Auction, Payment services)
- (Done) Layered architecture (Controller → Service → Repository)

Key Insight: The change was purely at the implementation layer. The system design, architecture patterns, and use case specifications remain identical to Deliverable 1.

---

2. Architecture Implementation

2.1 Layer Separation

We implemented a clean 3-layer architecture:

```
Controller Layer (REST API)
    ↓
Service Layer (Business Logic)
    ↓
Repository Layer (Data Access)
    ↓
Database (H2)
```

Benefits:

- Clear separation of concerns
- Easy to test each layer independently
- Follows Spring Boot best practices
- Ready for microservice decomposition in Deliverable 3

2.2 Service Modularity

Each service is self-contained:

- **UserService** (com.blockbid.user): UC1 authentication
- **ItemService** (com.blockbid.catalogue): UC2, UC7 item management
- **AuctionService** (com.blockbid.auction): UC3, UC4 bidding logic
- **PaymentService** (com.blockbid.payment): UC5, UC6 payment processing

This modular design allows easy conversion to separate microservices for Deliverable 3.

2.3 Database Design

Entity Relationships:

- User (1) → (Many) Items (seller relationship)
- User (1) → (Many) Bids (bidder relationship)
- Item (1) → (Many) Bids
- Item (1) → (One) Payment
- User (1) → (Many) Payments (buyer relationship)

Transaction Management: Critical operations use `@Transactional`:

- Bidding (update bid + update item price atomically)
- Payment (record payment + verify winner atomically)

---

3. Implementation Choices

3.1 H2 In-Memory Database

Choice: Used H2 instead of production database for Deliverable 2

Reasoning:

- Faster development and testing
- No external database installation required
- Easy for TA to run and test
- Will migrate to PostgreSQL for Deliverable 3 deployment

Trade-off: Data resets on application restart (acceptable for development)

3.2 Simplified Security

Choice: Plain text passwords, no JWT tokens yet

Reasoning:

- Focus on core functionality first
- Security enhancements planned for Deliverable 3
- Deliverable 2 emphasizes backend logic, not authentication infrastructure

Implementation:

- Password hashing with BCrypt: Deliverable 3
- JWT token authentication: Deliverable 3
- Role-based access control: Deliverable 3

3.3 HATEOAS Implementation

Choice: Used Spring HATEOAS library

Implementation:

- All POST endpoints return EntityModel with links
- Links show available next actions
- Meets marking scheme requirement: "follows REST principles (HATEOAS)"

Example Response:

```
{
  "id": 1,
  "username": "kyle",
  "_links": {
    "profile": {"href": "/api/users/1"},
    "search-items": {"href": "/api/items/search"}
  }
}
```

3.4 Manual Auction Ending

Choice: Manual endpoint to end auctions instead of automatic timer

Reasoning:

- Easier to test and demonstrate
- Automatic expiration checking requires background jobs
- Can add scheduled tasks in Deliverable 3

Current: `POST /api/auctions/items/{id}/end`
Future: Background scheduler checks `auctionEndTime` periodically

---

4. Test Cases

4.1 Test Coverage Summary

Implemented **16 comprehensive test cases** covering all use cases:

Test ID	Category	Description	Status
TC001	UC1.1	Successful user sign-up	Pass
TC002	UC1.1	Duplicate username validation	Pass
TC003	UC1.2	Successful login	Pass
TC004	UC1.2	Invalid credentials	Pass
TC005	UC2.1	Browse all active items	Pass
TC006	UC2.2	Search items by keyword	Pass
TC007	UC7	Successful item creation	Pass
TC008	UC7	Invalid seller ID	Pass
TC009	UC3	Successful bid placement	Pass
TC010	UC3	Bid amount too low	Pass
TC011	UC3	Bid on ended auction	Pass
TC012	UC3	Get bid history	Pass
TC013	UC4	End auction successfully	Pass
TC014	UC4	Check winner	Pass
TC015	UC5	Successful payment (winner)	Pass

Test ID	Category	Description	Status
TC016	UC5	Payment by non-winner (rejected)	Pass

All 16 test cases pass successfully.

4.2 Test Organization

Tests are organized in Postman collection by use case:

- **UC1 - User Management:** 4 tests
- **UC2 - Browse and Search:** 2 tests
- **UC7 - Item Upload:** 2 tests
- **UC3 - Bidding:** 4 tests
- **UC4 - Auction End:** 2 tests
- **UC5 - Payment:** 2 tests

4.3 Robustness Testing

Each use case includes both positive and negative test cases:

- (Done) Valid inputs (happy path)
- (Done) Invalid inputs (error handling)
- (Done) Boundary conditions (edge cases)
- (Done) Business rule violations (authorization, validation)

---

5. Project Plan Updates

5.1 Completed Milestones

Week 1-2: Design Phase (Deliverable 1)

- (Done) System architecture design
- (Done) UML diagrams (sequence, activity, component, class)
- (Done) Use case specifications
- (Done) Initial technology selection (Node.js)

Week 3-6: Implementation Phase (Deliverable 2)

- (Done) Technology pivot to Java/Spring Boot
- (Done) Project setup with Maven and Spring Boot
- (Done) Entity and repository layer implementation
- (Done) Service layer with business logic
- (Done) REST controller layer with HATEOAS
- (Done) UC1: User management (sign-up, login)
- (Done) UC2: Browse and search items
- (Done) UC7: Seller item upload
- (Done) UC3: Bidding with validation
- (Done) UC4: Auction end and winner determination
- (Done) UC5: Payment processing
- (Done) UC6: Receipt generation
- (Done) Comprehensive testing with Postman
- (Done) Documentation (README, design updates)

5.2 Deliverable 3 Planning

Remaining Work:

- Security: JWT authentication, password hashing
- Real-time: WebSocket for live bid updates
- Deployment: Docker containerization
- Orchestration: Kubernetes deployment
- UC8: Blockchain integration (advanced feature)
- Frontend: Progressive Web App
- Production database: PostgreSQL migration
- Advanced features: Scheduled auction expiration

---

6. Team Meeting Logs

Meeting 1: October 2, 2025

Attendees: Kyle Williamson
Duration: 2 hours
Topics:

- Reviewed project requirements and deliverables

- Selected UC8 (Blockchain) as advanced feature
- Created initial architecture design
- Decided on Node.js/Express technology stack
- Completed Deliverable 1 draft

Decisions:

- Microservices architecture
- MongoDB for catalogue, PostgreSQL for transactions
- Socket.io for real-time updates

Meeting 2: October 15, 2025

Attendees: Kyle Williamson
Duration: 1.5 hours
Topics:

- Started Deliverable 2 planning
- Discussed implementation approach
- Evaluated Node.js vs Java for backend

Decisions:

- **MAJOR DECISION:** Switch from Node.js to Java/Spring Boot
- Rationale: Development expertise, faster development, Spring Boot benefits
- Agreed to keep all architectural designs from Deliverable 1
- Decided to use H2 database for development

Meeting 3: October 30, 2025

Attendees: Kyle Williamson
Duration: 6 hours (implementation session)
Topics:

- Set up GitHub repository and project structure
- Implemented User service (UC1.1, UC1.2)
- Implemented Catalogue service (UC2.1, UC2.2, UC7)
- Added HATEOAS support
- Created initial Postman tests

Achievements:

- Complete User authentication working
- Item browsing and search functional
- 8 test cases passing

Meeting 4: November 1, 2025

Attendees: Kyle Williamson
Duration: 5 hours (implementation session)
Topics:

- Implemented Auction service (UC3, UC4)
- Implemented Payment service (UC5, UC6)
- Created comprehensive test suite
- Tested all use cases end-to-end

Achievements:

- Complete auction lifecycle working
- All 16 test cases passing
- Payment processing with winner validation
- Full integration testing complete

Meeting 5: November 2, 2025

Attendees: Kyle Williamson
Duration: 3 hours
Topics:

- Created Postman collection for submission
- Wrote README with installation instructions
- Updated design document
- Final testing and validation
- Prepared submission materials

Deliverables Completed:

- (Done) Complete backend implementation
- (Done) 16 passing test cases
- (Done) Postman collection
- (Done) README documentation
- (Done) Design update document
- (Done) Git repository with organized commits

---

7. Conclusion

7.1 Implementation Success

Successfully implemented all required use cases for Deliverable 2:

- 8 use cases fully functional
- 16 test cases with 100% pass rate
- Clean, modular, maintainable code
- Comprehensive documentation

7.2 Technology Change Impact

The switch from Node.js to Java/Spring Boot was successful:

- **Positive:** Faster development, stronger type safety, better tooling
- **Neutral:** All architectural designs remained valid
- **Trade-off:** Will need to update bibliography references

7.3 Readiness for Deliverable 3

The modular architecture positions the project well for Deliverable 3:

- Services are already separated and can be containerized
- REST APIs are well-defined for microservice communication
- Database relationships are clear for potential separation
- HATEOAS implementation supports service discovery

7.4 Lessons Learned

1. **Architecture matters more than technology:** Good design translates across languages
2. **Layered architecture pays off:** Clear separation made testing easier
3. **Test-driven development:** Building tests alongside code caught issues early
4. **Systematic approach:** Regular development sessions kept progress steady

---

Appendix A: Updated Technology Stack

Component	Deliverable 1 Proposal	Deliverable 2 Implementation
Backend Language	JavaScript	Java 17
Framework	Express.js	Spring Boot 3.2.0
Database	MongoDB + PostgreSQL	H2 (in-memory)
ORM	Mongoose + Sequelize	Spring Data JPA + Hibernate
API Style	REST	REST with HATEOAS
Build Tool	npm	Maven 3.9
Testing	Jest	Postman
Real-time	Socket.io	Planned for D3
Blockchain	Ethereum	Planned for D3 (UC8)

Appendix B: API Endpoint Reference

Complete list of implemented endpoints:

User Management (UC1)

- POST /api/users/signup - User registration
- POST /api/users/login - User authentication
- GET /api/users/{id} - Get user profile

Catalogue Management (UC2, UC7)

- GET /api/items - Browse all active items
- GET /api/items/search?keyword={keyword} - Search items
- POST /api/items?sellerId={id} - Create auction item
- GET /api/items/{id} - Get item details

Auction Management (UC3, UC4)

- POST /api/auctions/items/{id}/bid?bidderId={id}&amount={amount} - Place bid
- GET /api/auctions/items/{id}/bids - Get bid history
- GET /api/auctions/items/{id}/highest-bid - Get current highest bid
- POST /api/auctions/items/{id}/end - End auction
- GET /api/auctions/items/{id}/winner?userId={id} - Check if user won

Payment Management (UC5, UC6)

- POST /api/payments/items/{id}?userId={id} - Process payment

---

Document Version: 2.0
Last Updated: November 2, 2025
Prepared by: Kyle Williamson