

**VIETNAM NATIONAL UNIVERSITY – HCM
INTERNATIONAL UNIVERSITY**



SEMESTER 1 (2025-2026)

Lecturer: Nguyen Van Sinh

Lab Lecturer: Nguyen Trung Nghia

Web Application Development

TOPIC: Hyper bid web game

Team Members

Full Name	Student ID
Pham Trung Kien	ITCSIU23020
Nguyen Ha An Thanh	ITITWE22051
Tran Thi Kieu My	ITCSIU23026

Github Repository:

<https://github.com/Nobodywinsbutme/tyk-webapp>

Contents

1 Abstract	1
2 Project overview	2
2.1 Project name	2
2.2 Game description	2
3 Requirements	2
3.1 Functional Requirements	2
3.2 Non-Functional Requirements	3
4 System Architecture	3
4.1 Architecture Style	3
5 Technology stack	4
6 API list	4
6.1 Register	6
6.1.1 Register Entry Point – AuthController	6
6.1.2 Registration Logic – AuthService	6
6.1.3 Register Request DTO	6
6.1.4 User Persistence	6
6.2 Explain workflow	7
6.3 Login	7
6.3.1 Login Entry Point – AuthController	7
6.3.2 Authentication Logic – AuthService	7
6.3.3 Login Request DTO	8
6.3.4 Security Configuration	8
6.4 Explain workflow	8
6.5 Logout	9
6.5.1 Logout Entry Point – AuthController	9
6.5.2 Session Termination	9
6.5.3 Security Context Cleanup	9
6.6 Explain workflow	9
6.7 Marketplace Sell Implementation	9
6.7.1 Sell Endpoint – MarketRestController	9
6.7.2 Sell Logic – MarketService	9
6.7.3 Sell Request DTO	10
6.7.4 Transactional Consistency	10
6.8 Explain workflow	11
6.9 Marketplace Buy Implementation	11
6.9.1 Buy Endpoint – MarketRestController	11
6.9.2 Buy Logic – MarketService	11
6.9.3 Inventory Transfer via InventoryService	12
6.9.4 Transactional Consistency	12
6.10 Explain workflow	12
6.11 Inventory View Implementation	12

6.11.1	Inventory Page Route – InventoryViewController	12
6.11.2	Inventory View API – InventoryRestController	12
6.11.3	Inventory Filtering and Tabs – InventoryService	13
6.11.4	Filtered Inventory Query – InventoryRepository	14
6.11.5	Frontend Inventory Loading – inventory.html	15
6.12	Explain workflow	15
6.13	Upload a New Community Design Implementation	15
6.13.1	Upload Endpoint – DesignController	15
6.13.2	Upload Logic – DesignService	16
6.13.3	Image Storage – saveFile	16
6.13.4	Serving Uploads – WebConfig	17
6.13.5	Design Entity – DesignSubmission	17
6.14	Explain workflow	18
6.15	Delete a Specific Design Implementation	18
6.15.1	Delete Endpoint – DesignController	18
6.15.2	Delete Logic – DesignService	18
6.15.3	Database Deletion – DesignRepository	19
6.15.4	Transactional Consistency	19
6.16	Explain workflow	19
7	Database design (ERD)	19
7.1	Suggested Entities	19
8	Workflows and User Journeys	20
9	UI / Screenshots	24
10	Deployment and Operations	25
10.1	Local Development	25
10.2	Production	25
11	Testing Plan	26
12	Security Considerations	26
13	Project Plan and Timeline	27
14	Contributions	28
15	References	29

1 Abstract

This report describes the development of a web-based game item marketplace created for the Web Application Development course. The system is built using Spring Boot 3, Java 17, MySQL, and Thymeleaf, and follows a monolithic architecture combining RESTful APIs with server-side rendering. The application allows users to trade virtual items using in-application currency, ensuring data consistency through transactional processing. This document outlines the system objectives, architecture, core

functionalities, API design, database schema, user workflows, testing approach, and deployment guidelines.

2 Project overview

2.1 Project name

Hyper bid web game (skeleton)

2.2 Game description

This project is a web-based game application that provides a virtual environment where users can register, authenticate, manage in-game items, and trade them through an integrated marketplace. The system uses an in-application currency to support buying and selling activities and ensures data consistency during transactions. It is developed using Spring Boot with Java for the backend, MySQL for data storage, and HTML, CSS, and JavaScript for the frontend, following a monolithic architecture that combines RESTful services with server-side rendering.

3 Requirements

3.1 Functional Requirements

- The system shall allow users to register a new account with unique credentials.
- The system shall allow users to log in and log out using session-based authentication.
- The system shall provide a user profile endpoint to retrieve account information (e.g., coin balance).
- The system shall allow users to browse marketplace listings with optional filtering (type, price range, keyword).
- The system shall allow users to list items for sale from their inventory with a specified price and quantity.
- The system shall allow users to purchase an active listing and automatically update both users' coin balances.
- The system shall transfer purchased item quantities to the buyer's inventory after a successful purchase.
- The system shall allow sellers to cancel their listings and return items to their inventory.
- The system shall allow sellers to update the price of an active listing.
- The system shall allow users to view their inventory with filtering and pagination, including an "On Sale" tab.

- The system shall allow users to upload community designs with images and meta-data (title, description, category).
- The system shall allow users to view their own uploaded designs and manage them (edit/delete).
- The system shall allow the public to view approved community designs.
- The system shall allow administrators to review pending designs and approve or reject them.
- The system shall provide a news feed and allow administrators to create, update, and delete news posts.

3.2 Non-Functional Requirements

- **Security:** Passwords shall be stored using strong hashing (BCrypt).
- **Consistency:** Marketplace purchase operations shall be executed within database transactions to prevent partial updates.
- **Usability:** The UI shall provide clear feedback for common actions (login, listing, purchase, upload, moderation).
- **Maintainability:** The application shall follow a layered structure (Controller, Service, Repository, Entity, DTO).
- **Scalability:** The database schema shall support growth in users, inventory items, listings, designs, and news posts.

4 System Architecture

4.1 Architecture Style

The application follows a **monolithic architecture** implemented using Spring Boot. It combines:

- **Server-Side Rendering (SSR)** using Thymeleaf templates for main pages.
- **REST APIs** for asynchronous operations (authentication, inventory, marketplace, designs, news).

Architecture diagram:

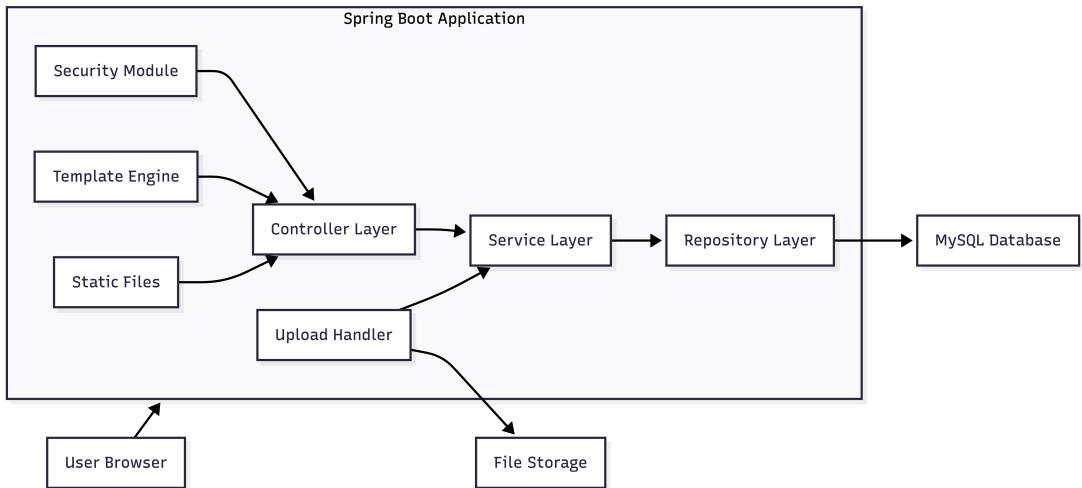


Figure 1: High-level architecture of the web-based game application.

5 Technology stack

Backend	Java 17 with Spring Boot 3, using Spring MVC and Spring Data JPA for business logic and data access.
Frontend	HTML5, CSS (Bootstrap 5), and JavaScript (Vanilla JS with Fetch API), combined with Thymeleaf for server-side rendering.
Database	MySQL relational database, managed through JPA/Hibernate.
Authentication	Spring Security with session-based authentication and BCrypt password hashing.
Build Tools	Apache Maven for dependency management and build automation.
Containerization	Docker (Dockerfile provided for application deployment).
Version Control	Git with GitHub for source code management and collaboration.

6 API list

The following table summarizes the core REST APIs implemented in the system.

Endpoint	Method	Description	Auth
/api/auth/register	POST	Register a new user account	No
/api/auth/login	POST	Authenticate user and create session	No
/api/auth/logout	POST	Invalidate current user session	Yes
/api/auth/profile/{username}	GET	Retrieve user profile and coin balance	Yes
/api/market/listings	GET	Retrieve marketplace listings with filters	No
/api/market/sell/{userId}	POST	List an item on the market	Yes
/api/market/buy/{userId}/{listingId}	POST	Purchase a marketplace listing	Yes
/api/market/{listingId}	DELETE	Cancel an active marketplace listing	Yes
/api/market/{listingId}	PUT	Update the price of a listing	Yes
/api/inventory/{userId}	GET	Retrieve user inventory with filters	Yes
/api/inventory/sell/{userId}	POST	Sell item directly from inventory	Yes
/api/designs/upload	POST	Upload a new community design	Yes
/api/designs/public	GET	Retrieve approved public designs	No
/api/designs/my-designs	GET	Retrieve designs uploaded by current user	Yes
/api/designs/pending	GET	Retrieve pending designs for moderation	Yes (Admin)
/api/designs/{id}	DELETE	Delete a specific design	Yes
/api/designs/{id}	PUT	Update design details	Yes
/api/designs/{id}/status	PUT	Approve or reject a design submission	Yes (Admin)
/api/news/list	GET	Retrieve published news articles	No
/api/news/create	POST	Create a new news article	Yes (Admin)
/api/news/update/{id}	PUT	Update an existing news article	Yes (Admin)
/api/news/delete/{id}	DELETE	Delete a news article	Yes (Admin)

Table 1: Summary of Implemented REST APIs

6.1 Register

6.1.1 Register Entry Point – AuthController

```
1 @PostMapping("/register")
2 public ResponseEntity<?> register(
3     @RequestBody RegisterRequestDTO request) {
4     authService.register(request);
5     return ResponseEntity.status(HttpStatus.CREATED).body("User
6     registered successfully");
}
```

Listing 1: Register endpoint in AuthController

6.1.2 Registration Logic – AuthService

```
1 public void register(RegisterRequestDTO request) {
2
3     if (userRepository.existsByUsername(request.getUsername())) {
4         throw new RuntimeException("Username already exists");
5     }
6
7     if (userRepository.existsByEmail(request.getEmail())) {
8         throw new RuntimeException("Email already exists");
9     }
10
11    User user = new User();
12    user.setUsername(request.getUsername());
13    user.setEmail(request.getEmail());
14    user.setPassword(passwordEncoder.encode(request.getPassword()));
15    user.setRole(Role.USER);
16    user.setCoinBalance(1000L);
17
18    userRepository.save(user);
19 }
```

Listing 2: Register logic implemented in AuthService

6.1.3 Register Request DTO

```
1 public class RegisterRequestDTO {
2     private String username;
3     private String email;
4     private String password;
5 }
```

Listing 3: Register request data transfer object

6.1.4 User Persistence

```
1 userRepository.save(user);
```

Listing 4: User repository usage during registration

6.2 Explain workflow

1. The user fills in the registration form with a username, email, and password.
2. The client sends a registration request to the backend endpoint: POST /api/auth/register
3. The request is received by the authentication controller (AuthController).
4. The controller forwards the request to the authentication service by calling AuthService.register()
5. The authentication service performs the following operations:
 - Validates the registration data (e.g., username and email uniqueness).
 - Encrypts the user's password using BCrypt.
 - Creates a new User entity with the provided information.
 - Assigns default values such as role and initial coin balance.
6. The new user entity is saved to the database.
7. The database stores the new user record permanently.
8. The system returns a successful response to the client, confirming that the account has been created.

6.3 Login

6.3.1 Login Entry Point – AuthController

```
1 @PostMapping("/login")
2 public ResponseEntity<?> login(
3     @RequestBody LoginRequestDTO request,
4     HttpServletRequest httpRequest) {
5     return authService.login(request, httpRequest);
6 }
```

Listing 5: Login endpoint in AuthController

6.3.2 Authentication Logic – AuthService

```
1 public ResponseEntity<?> login(
2     LoginRequestDTO request,
3     HttpServletRequest httpRequest) {
4
5     Authentication authentication =
6         authenticationManager.authenticate(
7             new UsernamePasswordAuthenticationToken(
8                 request.getUsername(),
9                 request.getPassword()
10            )
11        );
12
13     SecurityContextHolder.getContext().setAuthentication(
14         authentication);
```

```

14     httpServletRequest.getSession(true);
15
16     User user = userRepository
17         .findByUsername(request.getUsername())
18         .orElseThrow();
19
20     return ResponseEntity.ok(
21         new LoginResponseDTO(
22             user.getUsername(),
23             user.getRole().name(),
24             user.getCoinBalance()
25         )
26     );
27 }

```

Listing 6: Login logic implemented in AuthService

6.3.3 Login Request DTO

```

1 public class LoginRequestDTO {
2     private String username;
3     private String password;
4 }

```

Listing 7: Login request data transfer object

6.3.4 Security Configuration

6.4 Explain workflow

1. The user enters a username and password in the login form and submits the request.
2. The client sends a login request to the backend endpoint: POST /api/auth/login
3. The request is received by the authentication controller (AuthController).
4. The controller forwards the request to the authentication service (AuthService).
5. The authentication service validates the credentials using Spring Security's authentication manager.
6. If authentication is successful:
 - A server-side HTTP session is created.
 - The security context is updated to mark the user as authenticated.
7. The system retrieves the user's information from the database.
8. A successful response containing user information is returned to the client.
9. The user is now considered logged in and can access protected resources.

6.5 Logout

6.5.1 Logout Entry Point – AuthController

```
1 @PostMapping("/logout")
2 public ResponseEntity<?> logout(HttpServletRequest request) {
3     request.getSession().invalidate();
4     SecurityContextHolder.clearContext();
5     return ResponseEntity.ok("Logged out successfully");
6 }
```

Listing 8: Logout endpoint in AuthController

6.5.2 Session Termination

```
1 request.getSession().invalidate();
2 SecurityContextHolder.clearContext();
```

Listing 9: Session invalidation during logout

6.5.3 Security Context Cleanup

The logout functionality is implemented by invalidating the server-side session and clearing the Spring Security context, ensuring that the user's authenticated state is fully terminated.

6.6 Explain workflow

6.7 Marketplace Sell Implementation

6.7.1 Sell Endpoint – MarketRestController

```
1 @PostMapping("/sell/{userId}")
2 public ResponseEntity<?> sellItem(@PathVariable Long userId,
3                                     @RequestBody SellRequestDTO request) {
4     try {
5         marketService.sellItem(userId, request);
6         return ResponseEntity.ok("Item listed successfully");
7     } catch (Exception e) {
8         return ResponseEntity.badRequest().body(e.getMessage());
9     }
}
```

Listing 10: Sell endpoint in MarketRestController

6.7.2 Sell Logic – MarketService

```
1 @Transactional
2 public void sellItem(Long userId, SellRequestDTO request) {
3     InventoryItem item = inventoryRepository.findById(request.
4                                                       getItemId())
5             .orElseThrow(() -> new RuntimeException("Item not found"));
```

```

6   if (!item.getUser().getId().equals(userId)) {
7     throw new RuntimeException("You don't own this item");
8   }
9
10  if (item.getQuantity() < request.getQuantity()) {
11    throw new RuntimeException("Not enough items!");
12  }
13
14 // Create listing
15 MarketListing listing = new MarketListing();
16 listing.setSeller(item.getUser());
17 listing.setItemDefinition(item.getItemDefinition());
18 listing.setPrice(request.getPrice());
19 listing.setQuantity(request.getQuantity());
20 listing.setStatus("ACTIVE");
21 listing.setListedAt(LocalDateTime.now());
22
23 marketRepository.save(listing);
24
25 // Deduct from inventory
26 int newQuantity = item.getQuantity() - request.getQuantity();
27 if (newQuantity > 0) {
28   item.setQuantity(newQuantity);
29   inventoryRepository.save(item);
30 } else {
31   inventoryRepository.delete(item);
32 }
33 }
```

Listing 11: Sell logic implemented in MarketService

6.7.3 Sell Request DTO

```

1 public class SellRequestDTO {
2   private Long itemId;
3   private Long price;
4   private int quantity;
5 }
```

Listing 12: SellRequestDTO fields used by the sell workflow

6.7.4 Transactional Consistency

Summary: The marketplace sell feature is implemented through a REST endpoint in MarketRestController and business logic in MarketService. It validates ownership and quantity, creates an active listing, and updates the seller's inventory within a transactional boundary.

6.8 Explain workflow

6.9 Marketplace Buy Implementation

6.9.1 Buy Endpoint – MarketRestController

```
1 @PostMapping("/buy/{userId}/{listingId}")
2 public ResponseEntity<?> buyItem(@PathVariable Long userId,
3     @PathVariable Long listingId) {
4     try {
5         marketService.buyItem(userId, listingId);
6         return ResponseEntity.ok("Purchase successful");
7     } catch (Exception e) {
8         return ResponseEntity.badRequest().body(e.getMessage());
9     }
}
```

Listing 13: Buy endpoint in MarketRestController

6.9.2 Buy Logic – MarketService

```
1 @Transactional
2 public void buyItem(Long buyerId, Long listingId) {
3     MarketListing listing = marketRepository.findById(listingId)
4         .orElseThrow(() -> new RuntimeException("Listing not found"))
5 ;
6
7     if (!"ACTIVE".equals(listing.getStatus())) {
8         throw new RuntimeException("Item unavailable");
9     }
10
11    if (listing.getSeller().getId().equals(buyerId)) {
12        throw new RuntimeException("Cannot buy your own item");
13    }
14
15    User buyer = userRepository.findById(buyerId).orElseThrow();
16    User seller = listing.getSeller();
17
18    if (buyer.getCoinBalance() < listing.getPrice()) {
19        throw new RuntimeException("Not enough coins!");
20    }
21
22    // Coin transfer
23    buyer.setCoinBalance(buyer.getCoinBalance() - listing.getPrice())
24 ;
25    seller.setCoinBalance(seller.getCoinBalance() + listing.getPrice());
26
27    // Transfer items to buyer inventory
28    inventoryService.addItemToInventory(buyer, listing.
getDefinition(), listing.getQuantity());
29
30    // Finalize listing
```

```

29     listing.setStatus("SOLD");
30     marketRepository.save(listing);
31
32     userRepository.save(buyer);
33     userRepository.save(seller);
34 }
```

Listing 14: Buy logic implemented in MarketService

6.9.3 Inventory Transfer via InventoryService

```

1 inventoryService.addItemToInventory(buyer, listing.getItemDefinition
    (), listing.getQuantity());
```

Listing 15: Inventory update call during purchase

6.9.4 Transactional Consistency

Summary: The marketplace buy feature is implemented through a REST endpoint in MarketRestController and transactional business logic in MarketService. It validates listing availability and buyer balance, performs coin transfer, updates the buyer's inventory, and finalizes the listing status in an atomic transaction.

6.10 Explain workflow

6.11 Inventory View Implementation

6.11.1 Inventory Page Route – InventoryViewController

```

1 @GetMapping("/inventory")
2 public String showInventoryPage(Model model, HttpSession session,
3     Principal principal) {
4     User currentUser = userService.getAuthenticatedUser(session,
5     principal);
6
7     if (currentUser == null) {
8         if (session != null) {
9             session.invalidate();
10        }
11        return "redirect:/?message=expired";
12    }
13    model.addAttribute("user", currentUser);
14    return "inventory";
15 }
```

Listing 16: SSR route for the inventory page

6.11.2 Inventory View API – InventoryRestController

```

1 @GetMapping("/{userId}")
2 public ResponseEntity<?> getUserInventory(
3     @PathVariable Long userId,
```

```

4     @RequestParam(required = false) String type,
5     @RequestParam(required = false, defaultValue = "all") String
tab,
6     @RequestParam(required = false) String rarity,
7     @RequestParam(required = false) String search,
8     @RequestParam(defaultValue = "1") int page,
9     @RequestParam(defaultValue = "12") int size
10 ) {
11     return ResponseEntity.ok(inventoryService.getMyInventory(userId,
tab, type, rarity, search, page, size));
12 }

```

Listing 17: Inventory REST API with filtering and pagination

6.11.3 Inventory Filtering and Tabs – InventoryService

```

1 public Page<InventoryResponseDTO> getMyInventory(Long userId, String
tab, String typeStr, String rarityStr,
                                                 String keyword, int
page, int size) {
2
3
4     ItemType type = null;
5     if (typeStr != null && !typeStr.isEmpty() && !typeStr.
equalsIgnoreCase("ALL")) {
6         try { type = ItemType.valueOf(typeStr.toUpperCase()); } catch
(IllegalArgumentException e) { type = null; }
7     }
8
9     ItemRarity rarity = null;
10    if (rarityStr != null && !rarityStr.isEmpty() && !rarityStr.
equalsIgnoreCase("ALL")) {
11        try { rarity = ItemRarity.valueOf(rarityStr.toUpperCase()); } catch
(IllegalArgumentException e) { rarity = null; }
12    }
13
14    Pageable pageable = PageRequest.of(page - 1, size, Sort.by("id").
descending());
15
16    // Tab "On Sale" -> read from MarketListing
17    if ("on_sale".equalsIgnoreCase(tab)) {
18        Page<MarketListing> marketPage =
19            marketListingRepository.findBySellerIdAndStatus(userId, "ACTIVE",
pageable);
20
21        return marketPage.map(listing -> InventoryResponseDTO.builder()
22
23            .id(listing.getId())
24            .name(listing.getItemDefinition().getName())
25            .imageUrl(listing.getItemDefinition().getImageUrl())
26            .type(listing.getItemDefinition().getType().toString
())
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
311
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1
```

```

26         .rarity(listing.getItemDefinition().getRarity()).
27         toString())
28         .quantity(listing.getQuantity())
29         .description(listing.getItemDefinition().
30             getDescription())
31             .price(listing.getPrice())
32             .listedAt(listing.getListedAt().toString())
33             .isTradable(true)
34             .isBookmarked(false)
35             .build());
36
37     }
38
39
40     // Tab "Owned" (default) -> read from InventoryItem
41     Page<InventoryItem> rawPage =
42         inventoryRepository.findByUserAndFilter(userId, type, rarity,
43             keyword, pageable);
44
45     return rawPage.map(item -> InventoryResponseDTO.builder()
46         .id(item.getId())
47         .name(item.getItemDefinition().getName())
48         .imageUrl(item.getItemDefinition().getImageUrl())
49         .type(item.getItemDefinition().getType().toString())
50         .rarity(item.getItemDefinition().getRarity() != null ?
51             item.getItemDefinition().getRarity().toString() : "COMMON")
52         .quantity(item.getQuantity())
53         .description(item.getItemDefinition().getDescription())
54         .isTradable(true)
55         .isBookmarked(false)
56         .build());
57 }

```

Listing 18: Inventory view logic with Owned and On-Sale tabs

6.11.4 Filtered Inventory Query – InventoryRepository

```

1 @Query("SELECT i FROM InventoryItem i " +
2     "JOIN i.itemDefinition d " +
3     "WHERE i.user.id = :userId " +
4     "AND (:type IS NULL OR d.type = :type) " +
5     "AND (:rarity IS NULL OR d.rarity = :rarity) " +
6     "AND (:keyword IS NULL OR LOWER(d.name) LIKE LOWER(CONCAT('%',
7 :keyword, '%')))")
8 Page<InventoryItem> findByUserAndFilter(
9     @Param("userId") Long userId,
10    @Param("type") ItemType type,
11    @Param("rarity") ItemRarity rarity,
12    @Param("keyword") String keyword,
13    @Param("pageable") Pageable pageable
14 );

```

Listing 19: JPA query for filtering inventory items

6.11.5 Frontend Inventory Loading – inventory.html

```
1 async function loadInventory(page = 1) {
2     const search = document.getElementById('searchInput').value;
3
4     let url = `/api/inventory/${userId}?page=${page}&size=12&tab=${currentTab}`;
5     if (currentType) url += `&type=${currentType}`;
6     if (currentRarity) url += `&rarity=${currentRarity}`;
7     if (search) url += `&search=${search}`;
8
9     const response = await fetch(url);
10    const data = await response.json();
11
12    renderItems(data.content);
13    renderPagination(data.totalPages, page);
14 }
```

Listing 20: Frontend fetch call to load inventory data

Summary: The inventory view is implemented using a combination of server-side rendered navigation (/inventory) and a REST API (/api/inventory/{userId}) for dynamic data loading. The backend supports filtering, pagination, and tab-based views (Owned vs On Sale), while the frontend fetches and renders the paginated results using JavaScript.

6.12 Explain workflow

6.13 Upload a New Community Design Implementation

6.13.1 Upload Endpoint – DesignController

```
1 @PostMapping(value = "/upload", consumes = MediaType.
2     MULTIPART_FORM_DATA_VALUE)
3 public ResponseEntity<?> uploadDesign(
4     @RequestParam("title") String title,
5     @RequestParam("description") String description,
6     @RequestParam("category") String category,
7     @RequestParam(value = "image", required = false)
8     MultipartFile image,
9     @AuthenticationPrincipal UserDetails userDetails) {
10
11
12     try {
13         DesignResponseDTO response = designService.createDesign(
14             title, description, category, image, userDetails.
15             getUsername()
16             );
17         return ResponseEntity.ok(response);
18     } catch (Exception e) {
19         return ResponseEntity.badRequest().body("Error uploading
design: " + e.getMessage());
20     }
21 }
```

17 }

Listing 21: Upload endpoint in DesignController

6.13.2 Upload Logic – DesignService

```
1 public DesignResponseDTO createDesign(String title, String desc,
2                                         String category,
3                                         MultipartFile image, String
4                                         username) throws IOException {
5     User user = userRepository.findByUsername(username)
6         .orElseThrow(() -> new RuntimeException("User not found"))
7 );
8
9     DesignSubmission design = new DesignSubmission();
10    design.setTitle(title);
11    design.setDescription(desc);
12    design.setCreator(user);
13
14    // Admin uploads are approved immediately; user uploads are
15    // pending
16    if ("ADMIN".equals(user.getRole())) {
17        design.setStatus(DesignSubmission.SubmissionStatus.APPROVED);
18        design.setApprovedBy(user);
19    } else {
20        design.setStatus(DesignSubmission.SubmissionStatus.PENDING);
21    }
22
23    try {
24        design.setCategory(DesignSubmission.DesignCategory.valueOf(
25            category.toUpperCase()));
26    } catch (Exception e) {
27        throw new RuntimeException("Category not valid");
28    }
29
30    if (image != null && !image.isEmpty()) {
31        design.setImageUrl(saveFile(image));
32    } else {
33        throw new RuntimeException("Must have an image!");
34    }
35
36    DesignSubmission saved = designRepository.save(design);
37    return DesignResponseDTO.fromEntity(saved);
38 }
```

Listing 22: Create design logic implemented in DesignService

6.13.3 Image Storage – saveFile

```
1 private String saveFile(MultipartFile file) throws IOException {
2     if (!Files.exists(UPLOAD_DIR)) Files.createDirectories(UPLOAD_DIR
3 );
```

```

3     String originalName = file.getOriginalFilename();
4     String extension = originalName != null && originalName.contains(
5         ".")
6             ? originalName.substring(originalName.
7                 lastIndexOf(".")) :
8                 ".jpg";
9
10    String newName = UUID.randomUUID().toString() + extension;
11    Path path = UPLOAD_DIR.resolve(newName);
12    Files.copy(file.getInputStream(), path, StandardCopyOption.
13        REPLACE_EXISTING);
14
15    return "/uploads/" + newName;
16
17 }
```

Listing 23: Image saving logic in DesignService

6.13.4 Serving Uploads – WebConfig

```

1 @Override
2 public void addResourceHandlers(ResourceHandlerRegistry registry) {
3     Path uploadDir = Paths.get("./uploads");
4     String uploadPath = uploadDir.toUri().toString();
5     if (!uploadPath.endsWith("/")) {
6         uploadPath += "/";
7     }
8     registry.addResourceHandler("/uploads/**")
9         .addResourceLocations(uploadPath);
10 }
```

Listing 24: Static resource mapping for /uploads/**

6.13.5 Design Entity – DesignSubmission

```

1 @Entity
2 @Table(name = "design_submissions")
3 public class DesignSubmission {
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7
8     @Column(nullable = false)
9     private String title;
10
11    @Column(columnDefinition = "TEXT")
12    private String description;
13
14    @Column(length = 500)
15    private String imageUrl;
16
17    @Enumerated(EnumType.STRING)
```

```

18     @Column(nullable = false)
19     private DesignCategory category;
20
21     @Enumerated(EnumType.STRING)
22     private SubmissionStatus status = SubmissionStatus.PENDING;
23
24     @ManyToOne(fetch = FetchType.LAZY)
25     @JoinColumn(name = "creator_id", nullable = false)
26     private User creator;
27
28     @Column(updatable = false)
29     private LocalDateTime createdAt;
30
31     @PrePersist
32     protected void onCreate() {
33         this.createdAt = LocalDateTime.now();
34     }
35 }
```

Listing 25: DesignSubmission entity fields

Summary: The community design upload feature is implemented through the multipart upload endpoint in DesignController and the business logic in DesignService. The system stores uploaded images on the server file system, persists design metadata in the database, and applies a moderation workflow where user uploads are marked as PENDING while admin uploads can be approved immediately.

6.14 Explain workflow

6.15 Delete a Specific Design Implementation

6.15.1 Delete Endpoint – DesignController

```

1 @DeleteMapping("/{id}")
2 public ResponseEntity<?> deleteDesign(@PathVariable Long id,
3                                         @AuthenticationPrincipal
4                                         UserDetails userDetails) {
5     try {
6         designService.deleteDesign(id, userDetails.getUsername());
7         return ResponseEntity.ok("Design deleted successfully");
8     } catch (Exception e) {
9         return ResponseEntity.badRequest().body("Error deleting
design: " + e.getMessage());
10    }
11 }
```

Listing 26: Delete endpoint in DesignController

6.15.2 Delete Logic – DesignService

```

1 @Transactional
2 public void deleteDesign(Long designId, String username) {
```

```

3   User user = userRepository.findByUsername(username)
4       .orElseThrow(() -> new RuntimeException("User not found"))
5   );
6
7   DesignSubmission design = designRepository.findById(designId)
8       .orElseThrow(() -> new RuntimeException("Design not found
9   "));
10
11  // Authorization: allow owner or admin
12  boolean isOwner = design.getCreator().getId().equals(user.getId())
13  );
14  boolean isAdmin = "ADMIN".equalsIgnoreCase(user.getRole());
15
16  if (!isOwner && !isAdmin) {
17      throw new RuntimeException("You are not allowed to delete
18 this design");
19  }
20
21  // (Optional) delete the uploaded image file if stored locally
22  // deleteImageIfExists(design.getImageUrl());
23
24  designRepository.delete(design);
25 }
```

Listing 27: Delete design logic implemented in DesignService

6.15.3 Database Deletion – DesignRepository

```
1 designRepository.delete(design);
```

Listing 28: Repository delete operation

6.15.4 Transactional Consistency

Summary: The design deletion feature is implemented through the REST endpoint `DELETE /api/designs/{id}` in `DesignController` and the authorization-protected business logic in `DesignService`. Only the creator of the design or an administrator is permitted to delete a submission, and the deletion is performed transactionally through

6.16 Explain workflow

7 Database design (ERD)

7.1 Suggested Entities

- **User** (users): stores account information such as `id`, `username`, `email`, hashed password, `role`, `coinBalance`, and moderation fields (e.g., banned status).
- **ItemDefinition** (`item_definitions`): defines game item metadata including `id`, `name`, `description`, `type`, `rarity`, `imageUrl`, and `basePrice`.

- **InventoryItem** (`inventory_items`): represents items owned by users with `id`, `userId` (FK), `itemId` (FK to `ItemDefinition`), and `quantity`.
 - **MarketListing** (`market_listings`): stores marketplace sale listings with `id`, `sellerId` (FK to `User`), `itemDefinitionId` (FK), `price`, `quantity`, `status` (e.g., `ACTIVE/SOLD`), and `listedAt`.
 - **DesignSubmission** (`design_submissions`): stores community design uploads with `id`, `title`, `description`, `imageUrl`, `category`, `status` (`PENDING/APPROVED/REJECTED`), `creatorId` (FK), `approverId` (FK), and `createdAt`.
 - **News** (`news`): stores announcements/news posts with `id`, `title`, `description`, `createdAt`, and `authorId` (FK to `User`).
 - **Transaction** (`transactions`): stores trading history with `id`, `buyerId` (FK), `sellerId` (FK), `amount`, `transactionDate`, and a `type` field to classify transaction actions.

ERD:

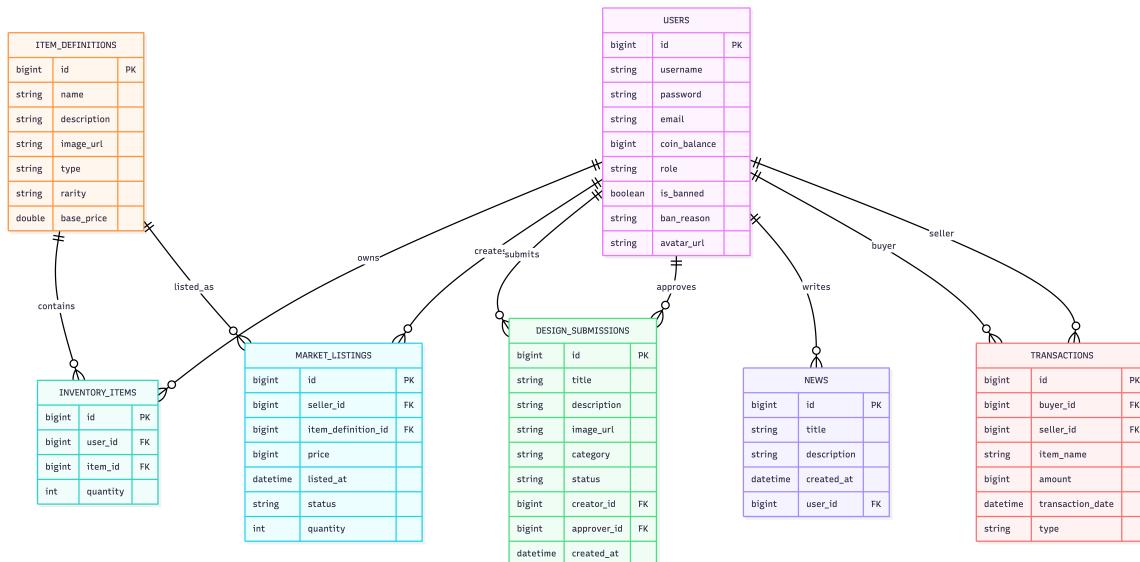


Figure 2: Entity Relationship Diagram (ERD).

8 Workflows and User Journeys

This section describes the main user flows implemented in the system:

- User registration and login (session-based authentication)
 - Inventory browsing and filtering (Owned vs On Sale, search, pagination)
 - Marketplace trading flow (sell item, buy item, cancel listing, update price)
 - Community design submission and moderation (upload, pending review, approve/reject)
 - Admin content management (news create/update/delete)

Workflow diagram:

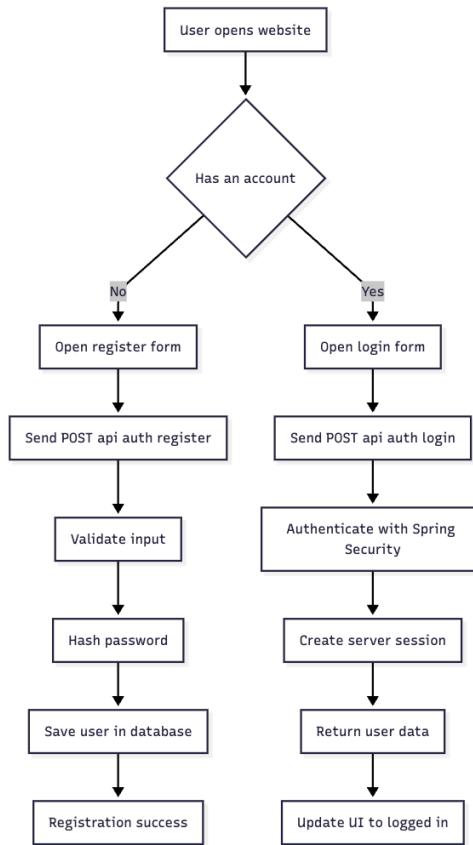


Figure 3: Auth flow (Register + Login)

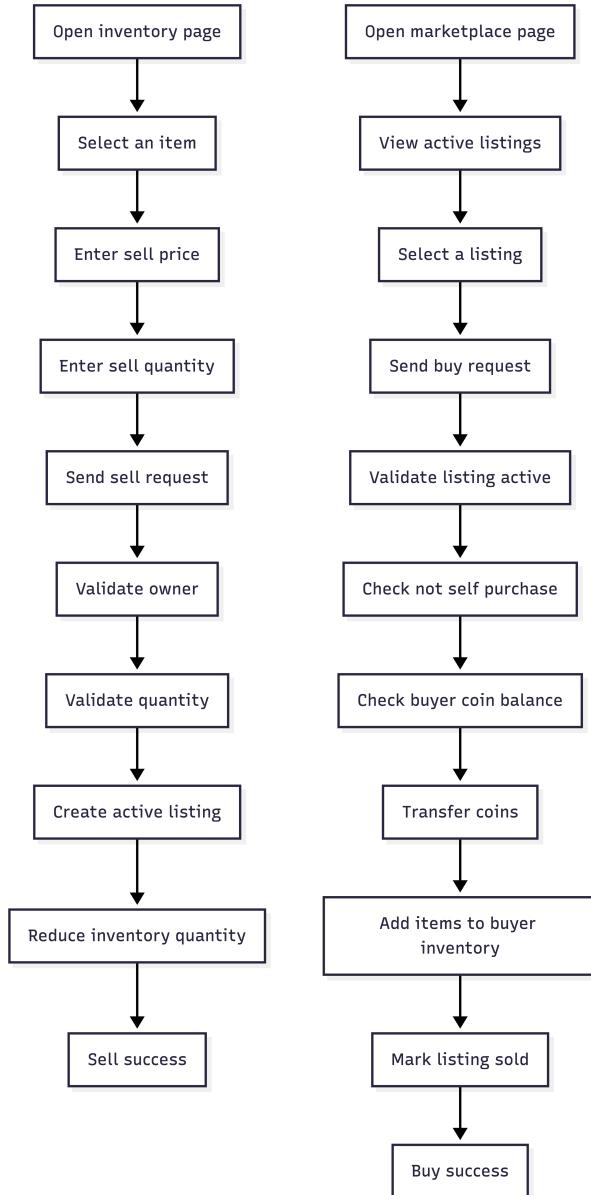


Figure 4: Marketplace trading workflow (sell and buy)

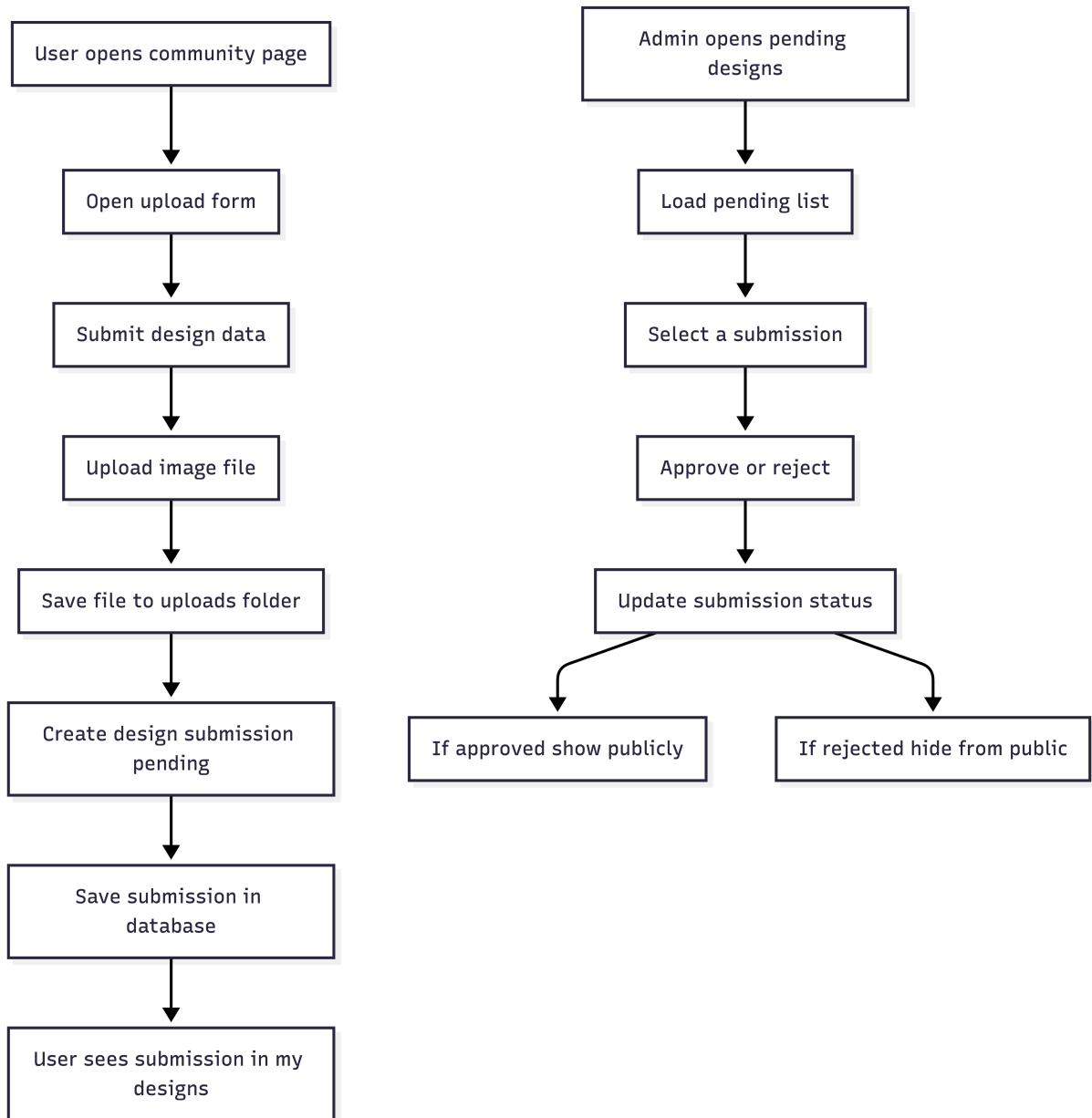


Figure 5: Design upload + admin moderation flow

9 UI / Screenshots

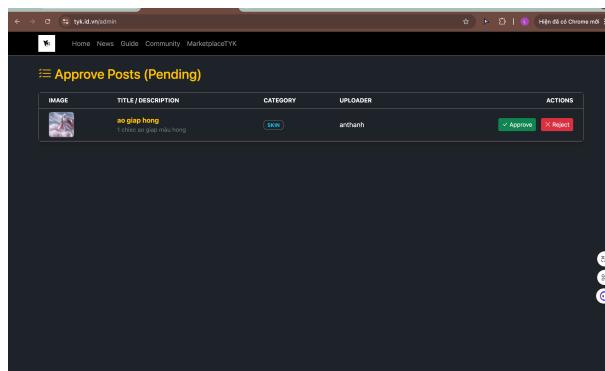


Figure 6: admin

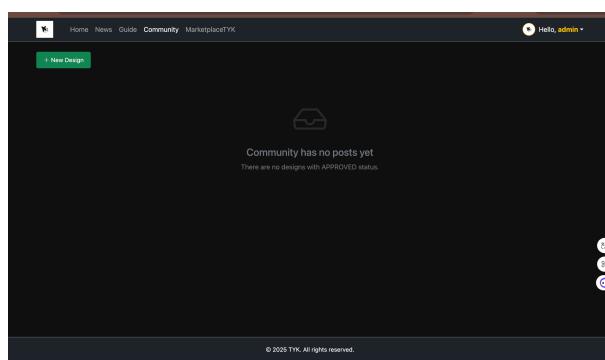


Figure 7: Community

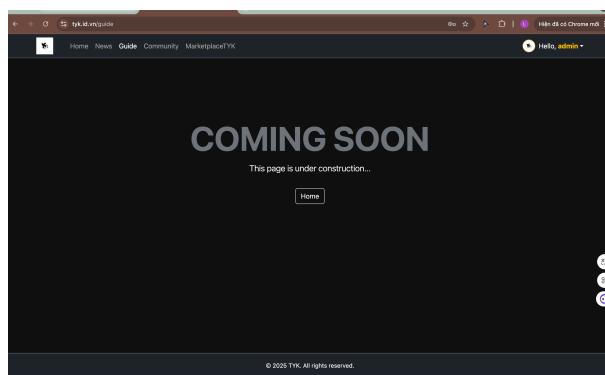


Figure 8: Guide

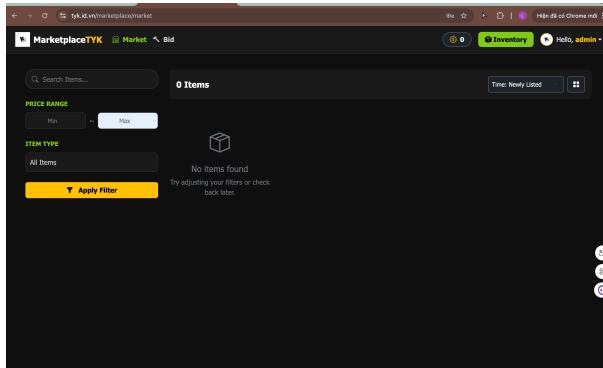


Figure 9: Market

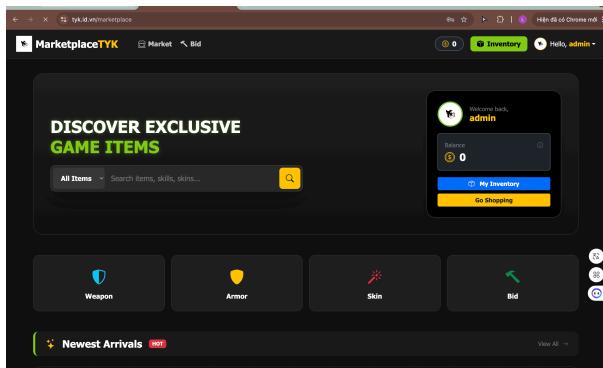


Figure 10: Marketplace

10 Deployment and Operations

10.1 Local Development

1. Clone the repository: <https://github.com/Nobodywinsbutme/tyk-webapp>
2. Open the project in an IDE (e.g., IntelliJ IDEA) or use the terminal.
3. Ensure MySQL is running locally (or start it using Docker). Update database credentials in `application.properties` or `application.yml` if needed.
4. Build and run the backend using Maven: `mvn clean install`
5. Start the Spring Boot application: `mvn spring-boot:run`
Alternatively, run the main class directly from the IDE.
6. Access the application in a browser: `http://localhost:8080`

10.2 Production

- **Containerization:** Package the Spring Boot application as a Docker image and run it as a single service (the frontend is served by Thymeleaf and static resources within the same application).

- **Database:** Use a managed MySQL service (e.g., Amazon RDS or Google Cloud SQL) and configure credentials via environment variables or a secure secret manager.
- **CI/CD:** Configure a CI pipeline (e.g., GitHub Actions) to build the project with Maven, execute automated tests, build the Docker image, and push it to a container registry.
- **Reverse Proxy and TLS:** Deploy a reverse proxy (e.g., Nginx) in front of the application to terminate TLS (HTTPS), handle routing, and optionally enable gzip compression and caching for static assets.
- **Operations:** Enable structured logging, basic health checks, and resource monitoring to support troubleshooting and maintain stable operation in production.

11 Testing Plan

- **Unit Testing (Backend):** Use JUnit 5 and Spring Boot Test to validate core service logic, including authentication validation, inventory filtering, and marketplace transaction rules (sell, buy, cancel, update price).
- **Integration Testing:** Test REST endpoints with Spring MockMvc (or TestRestTemplate) and execute database-related tests against a MySQL test database (local instance or Docker) to verify persistence and query behavior.
- **Transaction Consistency Testing:** Verify that purchase operations are atomic by testing failure scenarios (e.g., insufficient coin balance, inactive listing) to ensure no partial updates occur.
- **UI and Workflow Testing (Manual):** Manually validate end-to-end user journeys on the web interface, including registration/login/logout, inventory browsing with filters and pagination, marketplace selling and buying, design upload and deletion, admin approval/rejection, and news management.
- **File Upload Testing:** Validate multipart upload behavior for community designs, including file storage, URL accessibility under /uploads/**, and correct rendering on public pages.

12 Security Considerations

- **Password Storage:** User passwords are stored as salted hashes using BCrypt via Spring Security's password encoder.
- **Transport Security:** HTTPS/TLS should be enforced in production to protect credentials and session cookies in transit.
- **Session Security:** Authentication is session-based; therefore, secure cookie settings should be enabled (e.g., HttpOnly, Secure, and appropriate SameSite policy) to reduce the risk of session hijacking.
- **Access Control:** Role-based authorization should restrict administrative endpoints (e.g., moderation actions and news management) to administrators only.

- **Transactional Safety:** Marketplace purchase operations should be executed within database transactions to prevent inconsistent states (e.g., coin deducted without receiving items) and to reduce race-condition risks during concurrent purchases.
- **Input Validation:** All client inputs (query parameters and request bodies) should be validated on the server side to prevent invalid data and reduce injection risks.
- **File Upload Security:** Uploaded images should be validated for type and size limits, stored outside the application classpath, and served only through controlled static mappings (e.g., /uploads/**).
- **Operational Protections:** Rate-limiting and logging for authentication endpoints are recommended to mitigate brute-force login attempts and to support incident investigation.

13 Project Plan and Timeline

The project was developed by a team of three members. The work was divided into three main tracks: (1) authentication and administration (Dev 1), (2) item and inventory system (Dev 2), and (3) marketplace trading logic (Dev 3). Table 2 summarizes the planned timeline.

Week	Planned Activities (Dev 1 / Dev 2 / Dev 3)
Week 1	Dev 1: Define requirements, set up repository structure, project dependencies, and basic UI layout. Dev 2: Draft item and inventory data model (ItemDefinition, InventoryItem). Dev 3: Draft marketplace data model (MarketListing, Transaction) and trading rules.
Week 2	Dev 1: Implement authentication (register/login/logout), user entity, and Spring Security configuration. Dev 2: Implement ItemDefinition and InventoryItem entities and repositories. Dev 3: Implement MarketListing entity and marketplace repository queries (listing filters).
Week 3	Dev 1: Implement admin-related foundations (role handling, navigation updates, basic admin pages). Dev 2: Implement inventory service and inventory REST API (filters, pagination, data mapping DTOs). Dev 3: Implement marketplace service core flows (sell, cancel, update price) with transactional handling.
Week 4	Dev 1: Implement community features (design submission pages and moderation views) and news module endpoints. Dev 2: Integrate inventory UI with Fetch API and connect inventory-to-sell flow. Dev 3: Implement purchase workflow (buy) including coin transfer and inventory updates; integrate marketplace UI.
Week 5	Dev 1: Finalize admin moderation (approve/reject designs) and admin news management (CRUD). Dev 2: Add validation, edge-case handling, and improve inventory UI (tabs: Owned / On Sale). Dev 3: Add robustness checks (self-buy prevention, inactive listing handling) and transaction consistency tests.
Week 6	Dev 1: Prepare deployment configuration (Docker build/run) and finalize documentation sections. Dev 2: Perform system-wide testing and UI polishing. Dev 3: Perform end-to-end testing of marketplace flows and assist with final report and screenshots.

Table 2: Proposed project timeline for a three-developer team.

14 Contributions

- **Dev 1 -Pham Trung Kien- (Team Lead) – Authentication and Administration:** Implemented user registration, login/logout, Spring Security configuration, and core user management. Developed the base Thymeleaf UI structure (navigation and shared layout), and implemented administrative features including design moderation and news management.
- **Dev 2 -Tran Thi Kieu My- Items and Inventory System:** Designed and implemented the item and inventory modules, including database entities and repositories for item definitions and user inventory. Developed inventory APIs with filtering and pagination, and integrated the inventory UI for browsing and managing items.
- **Dev 3 -Nguyen Ha An Thanh- Marketplace Trading Logic:** Designed and implemented marketplace functionality, including listing creation (sell), purchasing (buy), cancellation, and price updates. Ensured transactional consistency for coin transfers and inventory updates, and integrated marketplace APIs with the frontend

user flows.

15 References

- Project repo: <https://github.com/Nobodywinsbutme/tyk-webapp>
- Course materials and slides
- Library docs: Spring Boot