Comprehensive Clean Architecture Design for Etoile Yachts Platform

I understand your request for a more comprehensive analysis incorporating all entities and functionality in the application. I will not implement any changes without your express approval.

Comparison of Our Approaches

Your structure offers a high-level view focusing on the functional aspects and user roles, while my proposal was more implementation-oriented. Let me combine both to create a more comprehensive architecture plan:

Expanded Clean Architecture Framework

1. Domain Layer (Core Business Rules)

**Entities (Pure Domain Objects)**:

* **User Domain**
  + User (base entity with core properties)
  + Administrator (invitation-based, approval workflows, MFA requirements)
  + Consumer (booking abilities, loyalty program)
  + Producer (yacht owner, experience provider)
  + Partner (service add-ons, complementary services)
* **Yacht & Experience Domain**
  + Yacht (core properties, without storage references)
  + YachtExperience (bookable package with pricing, availability)
  + Location (coordinates, marina details, region)
  + VirtualTour (immersive yacht exploration)
  + CustomizationOption (experience modifications)
  + Media (images and videos with validation rules)
* **Add-On Domain**
  + AddOn (base add-on entity)
  + ServiceAddOn (services offered by partners)
  + ProductAddOn (physical products available)
  + AddOnReference (relationship between add-ons and experiences)
* **Booking & Payment Domain**
  + Booking (reservation of experiences)
  + Payment (transaction details)
  + PaymentMethod (credit card, digital wallet)
  + BookingConfirmation (confirmed reservation)
  + BookingStatus (lifecycle states)
* **Marketing & Engagement Domain**
  + Promotion (special offers)
  + Review (customer feedback)
  + Notification (system alerts)
  + LoyaltyPoints (rewards system)
  + Article (content marketing)
  + Event (promotional events)

**Value Objects**:

* EmailAddress (validated email format)
* PhoneNumber (validated phone format)
* Password (secure password requirements)
* Money (currency, amount)
* DateRange (booking periods)
* GeoLocation (coordinates)
* MediaValidationResult (validation outcome)

**Domain Services**:

* YachtAvailabilityService (checks conflicts)
* BookingValidationService (validates bookings)
* MediaTypeDetectionService (distinguishes images/videos)
* UserVerificationService (validates user credentials)
* PricingCalculationService (dynamic pricing rules)
* CommissionCalculationService (partner payments)

2. Application Layer (Use Cases)

**User Management Use Cases**:

* RegisterUserUseCase (with variants for each role)
* AdminInvitationUseCase (generate/validate invites)
* AdminApprovalUseCase (review/approve admins)
* AuthenticationUseCase (login, validation)
* MfaEnrollmentUseCase (setup multi-factor)
* MfaVerificationUseCase (verify codes)
* SessionManagementUseCase (timeouts, refresh)
* UserProfileUseCase (view/update profiles)

**Yacht Management Use Cases**:

* CreateYachtUseCase (add new yacht)
* UpdateYachtUseCase (modify details)
* PublishYachtUseCase (make available)
* SearchYachtsUseCase (find by criteria)
* FeaturedYachtsUseCase (promotional listings)
* VirtualTourUseCase (create/view tours)
* YachtAvailabilityUseCase (manage calendar)

**Add-On Management Use Cases**:

* CreateAddOnUseCase (partner/producer adds service)
* BundleAddOnsUseCase (combine with yacht)
* ApproveAddOnUseCase (review partner add-ons)
* SearchAddOnsUseCase (find compatible add-ons)

**Booking Process Use Cases**:

* CreateBookingUseCase (reserve experience)
* CalculatePricingUseCase (total with add-ons)
* ProcessPaymentUseCase (charge customer)
* ConfirmBookingUseCase (after payment)
* CancelBookingUseCase (with refund logic)
* RescheduleBookingUseCase (date changes)

**Media Management Use Cases**:

* UploadMediaUseCase (images/videos)
* ValidateMediaUseCase (check URLs/content)
* RepairMediaUseCase (fix broken URLs)
* ResolveBlobUrlsUseCase (convert temporary URLs)

**Engagement Use Cases**:

* CreatePromotionUseCase (special offers)
* SubmitReviewUseCase (after booking)
* SendNotificationUseCase (alerts)
* RewardLoyaltyPointsUseCase (after booking)
* PublishContentUseCase (articles/events)

**Administrative Use Cases**:

* MediaValidationReportUseCase (system health)
* UserManagementUseCase (admin functions)
* SystemConfigurationUseCase (settings)
* AuditLogUseCase (security tracking)

3. Interface Adapters Layer

**Repositories (Interfaces)**:

* IUserRepository
* IYachtRepository
* IAddOnRepository
* IBookingRepository
* IPaymentRepository
* IMediaRepository
* IPromotionRepository
* IReviewRepository
* INotificationRepository
* IContentRepository

**Controllers**:

* AuthController (authentication flows)
* AdminController (admin functions)
* YachtController (yacht operations)
* BookingController (reservation process)
* PaymentController (financial transactions)
* MediaController (media operations)
* PartnerController (partner interactions)
* NotificationController (alerts)

**Presenters**:

* YachtPresenter (format yacht data for UI)
* BookingPresenter (format booking data)
* UserPresenter (format user profiles)
* MediaValidationPresenter (validation reports)
* DashboardPresenter (analytics data)

**Data Transfer Objects (DTOs)**:

* UserRegistrationDto
* YachtCreationDto
* BookingRequestDto
* PaymentDetailsDto
* MediaUploadDto
* ValidationReportDto

4. Infrastructure Layer

**Repositories (Implementations)**:

* FirebaseUserRepository
* FirestoreYachtRepository
* FirestoreAddOnRepository
* FirestoreBookingRepository
* StripePaymentRepository
* FirebaseStorageMediaRepository
* FirestorePromotionRepository
* FirestoreReviewRepository
* FirebaseMessagingRepository
* FirestoreContentRepository

**External Services Adapters**:

* FirebaseAuthAdapter
* FirestoreAdapter
* FirebaseStorageAdapter
* CloudMessagingAdapter
* StripePaymentAdapter
* EmailServiceAdapter
* WeatherApiAdapter
* AlgoliaSearchAdapter

**UI Components**:

* **Admin Portal**
  + AdminLoginComponent
  + MfaSetupComponent
  + MfaVerifyComponent
  + AdminDashboardComponent
  + MediaValidationComponent
  + UserManagementComponent
* **Consumer Interface**
  + YachtSearchComponent
  + YachtDetailsComponent
  + BookingFlowComponent
  + PaymentComponent
  + UserProfileComponent
  + VirtualTourComponent
* **Producer Dashboard**
  + YachtManagementComponent
  + BookingCalendarComponent
  + RevenueAnalyticsComponent
  + MediaUploadComponent
* **Partner Dashboard**
  + AddOnManagementComponent
  + PartnerAnalyticsComponent
  + CommissionReportComponent

**Configuration**:

* FirebaseConfig
* StripeConfig
* StorageConfig
* SecurityConfig
* EmailTemplateConfig

5. Main Layer (Application Entry Points)

* ServerBootstrap (Express setup)
* AppBootstrap (React initialization)
* FirebaseInitialization
* DependencyInjection (wire components)
* ErrorHandling (global handlers)
* Logging (application monitoring)

Technical Implementation Strategy

Step 1: Foundational Abstractions

1. Define interfaces for repositories and services
2. Create core entities with validation logic
3. Implement basic authentication and user management

Step 2: Media Validation Refactoring

1. Extract pure validation logic to domain services
2. Create repository interfaces for media operations
3. Implement use cases for validation and repair
4. Build adapters for Firebase Storage

Step 3: Admin Authentication Refactoring

1. Create domain entities and services for authentication
2. Implement use cases for admin workflow
3. Build repository interfaces and adapters
4. Refactor UI components to use presenters

Step 4: Booking System Refactoring

1. Extract booking logic to domain services
2. Create use cases for booking operations
3. Implement repository interfaces
4. Build adapters for Firestore and Stripe

Step 5: Partner Add-Ons Refactoring

1. Extract add-on logic to domain services
2. Create use cases for add-on operations
3. Implement repository interfaces
4. Connect with yacht experiences

Step 6: Testing Framework

1. Create mocks for repositories
2. Build test utilities for use cases
3. Implement integration test framework
4. Create end-to-end test suites

Directory Structure Proposal

/etoile-yachts/

├── /core/

│ ├── /domain/ # Pure business entities and rules

│ │ ├── /user/

│ │ ├── /yacht/

│ │ ├── /booking/

│ │ ├── /media/

│ │ ├── /add-on/

│ │ └── /marketing/

│ │

│ └── /application/ # Use cases

│ ├── /auth/

│ ├── /yacht-management/

│ ├── /booking-process/

│ ├── /media-validation/

│ ├── /add-on-management/

│ └── /marketing/

│

├── /adapters/ # Interface adapters

│ ├── /repositories/

│ │ ├── /interfaces/ # Repository interfaces

│ │ └── /implementations/ # Concrete implementations

│ │

│ ├── /controllers/ # Request handlers

│ ├── /presenters/ # Data formatters for UI

│ └── /dtos/ # Data transfer objects

│

├── /infrastructure/ # External frameworks and tools

│ ├── /firebase/ # Firebase adapters

│ ├── /stripe/ # Payment processing

│ ├── /storage/ # Media storage

│ ├── /search/ # Search functionality

│ ├── /messaging/ # Notifications

│ └── /email/ # Email services

│

├── /client/ # Frontend application

│ ├── /src/

│ │ ├── /pages/

│ │ ├── /components/

│ │ ├── /hooks/

│ │ └── /lib/

│

├── /server/ # Backend API

│ ├── /routes/

│ ├── /middleware/

│ └── /config/

│

└── /tests/ # Test suites

├── /unit/

├── /integration/

├── /e2e/

└── /utils/