**ReqVista - Key Design Considerations**

**1. Architecture Patterns**

**1.1 Domain-Driven Design (DDD)**

* **Bounded Contexts**: Define clear boundaries around the portfolio, project, and requirement domains
* **Ubiquitous Language**: Establish consistent terminology across codebase reflecting the ER diagram entities
* **Rich Domain Models**: Implement domain entities that encapsulate both data and behavior
* **Value Objects**: Use immutable value objects for concepts like status, priority, and dates
* **Aggregates**: Define aggregate roots (Portfolio, Project, Requirement) to maintain consistency boundaries

**1.2 Clean Architecture**

* **Layered Architecture**: Implement presentation, application, domain, and infrastructure layers
* **Dependency Inversion**: Dependencies point inward toward domain layer
* **Use Cases**: Define application services around specific user goals
* **Interface Segregation**: Define interfaces at boundaries between layers
* **Testability**: Ensure all components are easily testable through abstraction

**1.3 API Design**

* **RESTful Principles**: Follow REST conventions for resource naming and HTTP methods
* **API Versioning**: Implement versioning strategy for future evolution
* **Consistent Response Format**: Standardize response structure across all endpoints
* **OData Support**: Consider OData for flexible querying capabilities
* **GraphQL Evaluation**: Evaluate GraphQL for complex data retrieval operations
* **API Documentation**: Use Swagger/OpenAPI for automatic documentation generation

**2. Cross-Platform Strategy**

**2.1 Code Sharing**

* **.NET Standard Libraries**: Share business logic through .NET Standard libraries
* **Shared ViewModels**: Use shared ViewModels with [ObservableProperty] attributes
* **Interface Abstractions**: Abstract platform-specific services behind common interfaces
* **Component Library**: Build shared UI component library compatible with Blazor and MAUI
* **Design System**: Implement consistent design system across platforms

**2.2 Mobile-Specific Considerations**

* **Offline Mode Strategy**: Design comprehensive offline data access and sync strategy
* **Conflict Resolution**: Implement mechanisms for resolving data conflicts during sync
* **Partial Synchronization**: Allow selective sync of relevant data sets
* **Local Storage Optimization**: Optimize LiteDB schema for mobile performance
* **Battery and Data Usage**: Consider efficiency for battery life and data consumption
* **Cross-Platform UI**: Balance between native feel and development efficiency

**2.3 Web-Specific Considerations**

* **Blazor Rendering Mode**: Choose between Server-Side or WebAssembly based on requirements
* **Progressive Web App**: Consider PWA capabilities for offline access
* **Browser Compatibility**: Ensure compatibility with major modern browsers
* **SPA Navigation**: Implement efficient client-side navigation patterns
* **State Management**: Define clear state management strategy

**3. Data Management**

**3.1 Entity Framework Implementation**

* **Database-First vs. Code-First**: Choose appropriate approach for schema evolution
* **Migration Strategy**: Plan for schema migrations and database updates
* **Performance Optimization**: Configure for optimal query performance
* **Lazy Loading vs. Eager Loading**: Determine appropriate loading strategies
* **Concurrency Control**: Implement optimistic concurrency control
* **Global Query Filters**: Implement tenant or user-specific filtering at data access layer

**3.2 Offline Data Synchronization**

* **Delta Synchronization**: Sync only changed data to minimize transfer
* **Prioritized Sync**: Sync critical data first when reconnecting
* **Background Synchronization**: Implement background sync processes
* **Sync State Tracking**: Track synchronization state of each entity
* **Conflict Resolution Strategy**: Define rules for resolving synchronization conflicts
* **Data Consistency**: Maintain referential integrity during partial synchronization

**3.3 Data Security**

* **Data Classification**: Classify data by sensitivity level
* **Encryption**: Implement encryption for sensitive data at rest and in transit
* **Data Masking**: Mask sensitive data in logs and non-critical displays
* **Row-Level Security**: Implement row-level security in database
* **Data Retention**: Define and implement data retention policies
* **Secure Deletion**: Ensure proper data deletion when required

**4. User Experience Design**

**4.1 Responsive UI Strategy**

* **Mobile-First Design**: Design for mobile experience first, then expand
* **Fluid Layouts**: Implement fluid layouts that adapt to different screen sizes
* **Component Adaptivity**: Design components that transform based on viewport
* **Touch-Friendly Interface**: Ensure touch-friendly interaction elements
* **Graceful Degradation**: Provide graceful fallbacks for less capable devices

**4.2 Performance Optimization**

* **Lazy Loading**: Implement lazy loading for UI components and data
* **Virtual Scrolling**: Use virtualization for large data sets
* **Image Optimization**: Optimize images for different devices and connections
* **Bundle Optimization**: Minimize and optimize JavaScript and CSS bundles
* **Caching Strategy**: Implement appropriate client-side caching
* **Prefetching**: Implement prefetching for anticipated user actions

**4.3 Accessibility**

* **WCAG Compliance**: Comply with WCAG 2.1 Level AA standards
* **Keyboard Navigation**: Ensure complete keyboard navigation support
* **Screen Reader Compatibility**: Test and ensure screen reader compatibility
* **Color Contrast**: Maintain appropriate color contrast ratios
* **Semantic HTML**: Use semantic HTML elements for better accessibility
* **Accessibility Testing**: Implement automated and manual accessibility testing

**5. Security Architecture**

**5.1 Authentication and Authorization**

* **Identity Provider Integration**: Integrate with Azure AD or other identity providers
* **Multi-Factor Authentication**: Support MFA for enhanced security
* **Role-Based Access Control**: Implement role-based permissions aligned with entity model
* **Claims-Based Authorization**: Use claims for fine-grained access control
* **JWT Tokens**: Implement JWT for API authentication
* **Session Management**: Secure session management for web application

**5.2 API Security**

* **API Key Management**: Secure management of API keys for integrations
* **Rate Limiting**: Implement rate limiting to prevent abuse
* **Input Validation**: Thorough validation of all API inputs
* **CORS Configuration**: Proper configuration of Cross-Origin Resource Sharing
* **Security Headers**: Implement security headers for API responses
* **API Gateway**: Consider API gateway for centralized security controls

**5.3 Threat Mitigation**

* **SQL Injection Prevention**: Parameterized queries and ORM security
* **XSS Prevention**: Output encoding and Content Security Policy
* **CSRF Protection**: Anti-forgery tokens and proper cookie configuration
* **DDOS Protection**: Leverage Azure DDOS protection capabilities
* **Vulnerability Scanning**: Regular scanning and remediation process
* **Logging and Monitoring**: Comprehensive security event logging

**6. Scalability and Performance**

**6.1 Azure Architecture**

* **Microservices vs. Monolith**: Evaluate based on complexity and team structure
* **Service Bus Integration**: Use Azure Service Bus for reliable message processing
* **Caching Strategy**: Implement Redis Cache for performance optimization
* **CDN Integration**: Use Azure CDN for static content delivery
* **Storage Optimization**: Choose appropriate storage options for different data types
* **Compute Scaling**: Configure auto-scaling for compute resources

**6.2 Database Performance**

* **Indexing Strategy**: Design efficient indexes based on query patterns
* **Query Optimization**: Optimize EF Core queries for performance
* **Read/Write Separation**: Consider read replicas for reporting scenarios
* **Database Partitioning**: Plan for horizontal partitioning as data grows
* **In-Memory Options**: Evaluate In-Memory OLTP for high-throughput scenarios
* **Performance Monitoring**: Implement monitoring for database performance

**6.3 API Performance**

* **Pagination**: Implement consistent pagination for large result sets
* **Projection**: Use projection to return only required fields
* **Compression**: Enable response compression for bandwidth optimization
* **Caching Headers**: Implement appropriate HTTP caching headers
* **Batch Operations**: Support batch operations for multiple updates
* **Async Processing**: Use background processing for long-running operations

**7. DevOps and Infrastructure Considerations**

**7.1 Containerization Strategy**

* **Container Orchestration**: Kubernetes vs. Azure Container Apps evaluation
* **Microservices Boundaries**: Define service boundaries for containerization
* **Container Registry**: Setup private container registry in Azure
* **Podman Development Workflow**: Establish developer workflow with Podman
* **Multi-Stage Builds**: Optimize container builds with multi-stage approach
* **Container Security**: Implement container security best practices

**7.2 CI/CD Pipeline**

* **Build Automation**: Automate build process for all components
* **Testing Integration**: Integrate automated testing in pipeline
* **Environment Promotion**: Establish promotion flow between environments
* **Infrastructure as Code**: Use ARM templates or Terraform for infrastructure
* **Artifact Management**: Establish versioning and storage for build artifacts
* **Release Approval Process**: Define approval workflows for production releases

**7.3 Monitoring and Operations**

* **Application Insights**: Implement comprehensive telemetry
* **Centralized Logging**: Establish centralized logging solution
* **Alerting Strategy**: Define alerting thresholds and notification channels
* **Health Checks**: Implement health checks for all services
* **Runbooks**: Create operational runbooks for common scenarios
* **Disaster Recovery**: Implement and test disaster recovery procedures

**8. Integration Capabilities**

**8.1 External System Integration**

* **Integration Patterns**: Define appropriate integration patterns (sync vs. async)
* **API Gateway**: Consider API gateway for external system access
* **Webhook Support**: Implement webhooks for event notifications
* **File Import/Export**: Support for importing/exporting requirement data
* **Authentication for Integrations**: Secure authentication for integrated systems
* **Error Handling**: Robust error handling and recovery for integrations

**8.2 Reporting Integration**

* **Reporting Framework**: Integration with reporting tools and services
* **Data Warehouse Connection**: Strategy for data warehouse integration
* **Export Formats**: Support for various export formats (PDF, Excel, etc.)
* **Scheduled Reports**: Capability for scheduled report generation
* **Custom Report Templates**: Flexible template system for reports
* **Embedded Analytics**: Consider embedded analytics capabilities

**9. Extensibility and Customization**

**9.1 Plugin Architecture**

* **Extension Points**: Define clear extension points in the architecture
* **Plugin Registration**: Mechanism for registering and managing plugins
* **Versioning Support**: Support for managing plugin compatibility
* **Isolation Strategy**: Ensure plugins cannot destabilize core system
* **Custom Field Support**: Ability to add custom fields to entities
* **Scripting Support**: Consider limited scripting capabilities for customization

**9.2 Workflow Customization**

* **Workflow Engine**: Flexible workflow engine for requirement lifecycle
* **Custom States**: Allow configuration of custom states and transitions
* **Approval Processes**: Configurable approval processes
* **Notification Rules**: Customizable notification rules
* **Conditional Logic**: Support for conditional logic in workflows
* **Audit Trail**: Comprehensive tracking of workflow history