Architecture Report - Executive Summary

Date: 2025-11-19
Status: Ready for Team Review

📋 What's in This Report

We analyzed your Clean Architecture implementation and created a comprehensive guide for improving it. Here are the documents:

1. Architecture Analysis Report (Main Document)

- 📄 **50+ pages** of detailed analysis
- Current state assessment
- Problems identified
- Complete migration plan
- Code examples

2. Quick Reference Guide (Daily Use)

- 📖 **Quick decision trees**
- "Where does this code go?"
- Common patterns
- Anti-patterns to avoid

3. Migration Example (Step-by-Step)

- 🔨 **Concrete refactoring example**
- Before/after code
- Step-by-step instructions
- Apply to all 24 use cases

4. Thunk Layer Discussion (Addresses Concerns)

- 💬 **"Won't thunks make Presentation too large?"**

🎯 Key Findings (TL;DR)

✅ What's Working Well

1. **4-layer architecture is appropriate** for your complexity level
2. **Domain layer is excellent** - pure, no framework dependencies
3. **Repository pattern works well** - abstracts Firebase/Sanity
4. **DI provides value** for repositories and use cases
5. **Clear separation** between layers (mostly)

⚠️ Critical Issues Found

1. **24 use cases coupled to Redux Toolkit** (`createAsyncThunk`)
   - Cannot test without Redux
   - Cannot reuse in scripts/workers
   - Violates framework-agnostic principle
2. **StateManager in wrong layer** (`application/` → should be `presentation/`)
   - Redux is a delivery mechanism, not application logic
3. **3 dependency violations** (Application imports Presentation)
   - `MetricWithValue` type in wrong location
   - Hormone generators in screens folder

🔧 Recommended Solution

Decision: Keep 4 Layers + Decouple Use Cases

Why 4 layers:

- ✅ Right complexity for 24 use cases
- ✅ Team already familiar
- ✅ Industry standard
- ✅ Clear boundaries

What changes:

1. Move thunks from use cases → presentation layer
2. Move StateManager → presentation layer
3. Fix type import violations
4. Use cases return `Promise<T>`, not `createAsyncThunk`

📐 New Architecture Pattern

Before (Problematic)

```
// ❌ Application layer
@injectable()
export class GetDataUseCase {
  execute = createAsyncThunk(/* Redux stuff */);
}
```

After (Clean)

```
// ✅ Application layer - pure business logic
@injectable()
export class GetDataUseCase {
  async execute(params: Params): Promise<Result> {
    return await this.repository.getData(params);
  }
}

// ✅ Presentation layer - Redux integration
export const fetchData = createAsyncThunk("data/fetch", async (params) => {
  const useCase = container.resolve(GetDataUseCase);
  return await useCase.execute(params);
});
```

📊 Migration Scope

Affected Files

Category	Count	Effort
Use cases to refactor	24	15-20 min each
Thunk files to create	~6	1-2 hours total
Redux slices to update	13	15 min each
ViewModels to update	~20	5 min each
Test files to update	~24	20 min each

Total Estimated Time: 27-36 hours (4-5 days)

Affected Use Cases (All 24)

LifeContext (7):

- GetLifeContextUseCase
- InitiateMenstruationLifeContextUseCase
- RestartMenstruationLifeContextUseCase
- CalculateMenstrualCyclesUseCase
- UpdateMenstrualPhaseLengthsDataUseCase
- RecomputeAfterLifeContextChangeUseCase
- GetStatusModeUseCase

MetricLog (4):

📅 Suggested Timeline

Phase 1: Quick Wins (Week 1 - 1 day)

- ✅ Fix type import violations (30 min)
- ✅ Move StateManager to Presentation (3 hours)
- ✅ Move DI container to `src/container.ts` (15 min)
- ✅ Move legacy code to `src/old/` (30 min)
- **Deliverable:** Cleaner layer structure

Phase 2: Decouple Use Cases (Weeks 2-3 - 3-4 days)

- ✅ Create thunk files (6-8 hours)
- ✅ Refactor all 24 use cases (8-12 hours)
- ✅ Update Redux slices (3-4 hours)
- ✅ Update ViewModels (2-3 hours)
- **Deliverable:** Framework-agnostic use cases

Phase 3: Optimization (Week 4 - 1 day)

- ✅ Simplify stateless services (2 hours)
- ✅ Fix infrastructure violations (1 hour)
- ✅ Add architecture tests (2 hours)
- **Deliverable:** Enforced boundaries

🎁 Benefits

Immediate

✅ **Testability:** Use cases test without Redux
✅ **Clarity:** Clear separation of concerns
✅ **Standards:** Follows Clean Architecture principles

Long-term

✅ **Flexibility:** Could swap Redux for Zustand/MobX
✅ **Reusability:** Use cases in scripts, workers, CLI
✅ **Maintainability:** Easier onboarding, clearer patterns
✅ **Quality:** Enforced boundaries prevent violations

🚀 Next Steps

1. Team Review (This Week)

- [ ] Read executive summary (this document)
- [ ] Review main report sections 1-6
- [ ] Review migration example
- [ ] Discuss and approve plan

2. Plan Sprint (Next Week)

- [ ] Create tracking issues (24 use cases + setup tasks)
- [ ] Assign work (pair programming recommended)
- [ ] Set up branch: `architecture/decouple-use-cases`

3. Execute Migration (Weeks 2-4)

- [ ] Phase 1: Quick wins
- [ ] Phase 2: Refactor use cases (can be parallelized)
- [ ] Phase 3: Optimization

4. Document (Ongoing)

- [ ] Update team wiki with new patterns
- [ ] Add quick reference to onboarding
- [ ] Code review checklist

📚 How to Use These Documents

For Team Leads

👉 Read: **Architecture Analysis Report** (full details)

For Developers (Daily)

👉 Use: **Quick Reference Guide** (where to put code)

For Refactoring

👉 Follow: **Migration Example** (step-by-step guide)

For Addressing Concerns

👉 See: **Thunk Layer Discussion** (helper/layer concerns)
👉 See: **Container Location Rationale** (why outside layers)

For Architecture Decisions

👉 See: **Abstraction Strategy** (what should be abstracted)

❓ FAQ

Q: Do we have to do this?

A: No, but highly recommended. Current pattern violates Clean Architecture principles and makes testing/reusability difficult.

Q: Can we do this incrementally?

A: Yes! Start with Phase 1 (quick wins), then tackle use cases one at a time.

Q: Will this break anything?

A: No. This is a refactoring - behavior stays the same, structure improves.

Q: How long will this take?

A: 27-36 hours total (~4-5 days of focused work). Can be parallelized across team.

Q: What's the ROI?

A: Better testability, maintainability, and flexibility. Easier to onboard new developers.

Q: Can we simplify to 3 layers instead?

A: Not recommended. You have 24 use cases - the Application layer provides clear value.

📞 Questions or Clarifications?

- **Architecture concerns:** Review Section 6 of main report
- **Migration steps:** See Migration Example
- **Daily decisions:** Use Quick Reference
- **"Too large" concerns:** Read Thunk Layer Discussion
- **Container location:** Read Container Location Rationale
- **What to abstract:** Read Abstraction Strategy
- **Technical details:** See Main Report

✅ Approval Checklist

Before starting migration:

- [ ] Team has reviewed all documents
- [ ] Migration plan is approved
- [ ] Timeline is agreed upon
- [ ] Tracking issues created
- [ ] Branch created
- [ ] Pair programming partners assigned

Report Status: ✅ Complete and ready for review
Recommended Action: Schedule team review meeting
Priority: High (architectural debt)
Risk: Low (pure refactoring, no behavior changes)

📂 Complete Document Set

1. **Architecture Report Summary** ← You are here
2. **Architecture Analysis Report** (50+ pages, detailed)
3. **Architecture Quick Reference** (daily use)
4. **Migration Example** (step-by-step guide)
5. **Thunk Layer Discussion** (addresses size concerns)
6. **Container Location Rationale** (bootstrap design)
7. **Abstraction Strategy** (what to abstract)

Generated: 2025-11-19
Documents created: 7 comprehensive guides