Migration Example: Decoupling SaveMetricLogsUseCase

This document shows a **complete step-by-step example** of refactoring a use case from the old pattern (coupled to Redux) to the new pattern (framework-agnostic).

Before: Current Implementation

File Structure (Before)

```
src/
├── application/
│   ├── useCases/metricLog/
│   │   └── SaveMetricLogs.useCase.ts  ← Has createAsyncThunk
│   └── StateManager/
│       └── slices/metricLog/
│           └── metricLogSlice.ts  ← Consumes thunk from use case
└── presentation/
    └── viewModels/MetricLog/
        └── useMetricLog.ts  ← Dispatches thunk
```

Step 0: Original Code

`src/application/useCases/metricLog/SaveMetricLogs.useCase.ts`

```typescript
import { injectable, inject } from "tsyringe";
import { createAsyncThunk } from "@reduxjs/toolkit";
import type {
  MetricLogRepository,
  SuccessMessage,
} from "@/domain/repositories/MetricLogRepository";
import { MetricId } from "@/domain/entities/MetricLog/MetricLog.entity";

@injectable()
export class SaveMetricLogsUseCase {
  constructor(
    @inject("MetricLogRepository")
    private repository: MetricLogRepository,
  ) {}

  execute = createAsyncThunk<
    SuccessMessage,
    {
      logDate: Date;
      inserts: { metric_id: MetricId; value: number | null }[];
      deletes: MetricId[];
    },
    { rejectValue: string }
  >(
    "metricLog/saveLogs",
    async ({ logDate, inserts, deletes }, { rejectWithValue }) => {
      try {
        const res = await this.repository.saveMetricLogs(
          logDate,
          inserts,
          deletes,
        );
        return res;
      } catch (error) {
        const errorMessage =
          error instanceof Error ? error.message : "Failed to save metric logs";
        return rejectWithValue(errorMessage);
      }
    },
  );
}
```

`src/application/StateManager/slices/metricLog/metricLogSlice.ts`

```ts
import { createSlice } from "@reduxjs/toolkit";
import { container } from "tsyringe";
import { SaveMetricLogsUseCase } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";

const metricLogSlice = createSlice({
  name: "metricLog",
  initialState: {
    logs: [],
    loading: false,
    error: null,
  },
  reducers: {},
  extraReducers: (builder) => {
    const saveMetricLogsThunk = container.resolve(
      SaveMetricLogsUseCase,
    ).execute;

    builder
      .addCase(saveMetricLogsThunk.pending, (state) => {
        state.loading = true;
        state.error = null;
      })
      .addCase(saveMetricLogsThunk.fulfilled, (state, action) => {
        state.loading = false;
        // Handle success
      })
      .addCase(saveMetricLogsThunk.rejected, (state, action) => {
        state.loading = false;
        state.error = action.payload || "Failed to save";
      });
  },
});

export default metricLogSlice.reducer;
```

Problems:

- ✗ Slice directly references use case
- ✗ `container.resolve()` called in extraReducers (not ideal)

src/presentation/viewModels/MetricLog/useMetricLog.ts

```typescript
import { useCallback } from "react";
import { container } from "tsyringe";
import { useAppDispatch } from "@/application/StateManager/store";
import { SaveMetricLogsUseCase } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";

export const useMetricLog = () => {
  const dispatch = useAppDispatch();
  const saveMetricLogsUseCase = container.resolve(SaveMetricLogsUseCase);

  const saveMetricLogs = useCallback(
    async (logDate, inserts, deletes) => {
      try {
        await dispatch(
          saveMetricLogsUseCase.execute({ logDate, inserts, deletes }),
        ).unwrap();
      } catch (error) {
        console.error("Failed to save metric logs:", error);
      }
    },
    [dispatch, saveMetricLogsUseCase],
  );

  return { saveMetricLogs };
};
```

Problems:

- ✗ ViewModel directly resolves use case (should use thunk)
- ✗ References `StateManager` from Application layer

After: Refactored Implementation

File Structure (After)

```
src/
├── application/
│   └── useCases/metricLog/
│       └── SaveMetricLogs.useCase.ts  ← Pure business logic
├── presentation/
│   ├── state/  ← MOVED from application/StateManager
│   │   ├── slices/metricLog/
│   │   │   └── metricLogSlice.ts
│   │   ├── thunks/
│   │   │   └── metricLog.thunks.ts  ← NEW: Redux integration
│   │   └── store.ts
│   └── viewModels/MetricLog/
│       └── useMetricLog.ts  ← Uses thunk
```

Step 1: Refactor Use Case

Remove Redux dependency, make it framework-agnostic:

`src/application/useCases/metricLog/SaveMetricLogs.useCase.ts` (NEW)

```typescript
import { injectable, inject } from "tsyringe";
import type {
  MetricLogRepository,
  SuccessMessage,
} from "@/domain/repositories/MetricLogRepository";
import { MetricId } from "@/domain/entities/MetricLog/MetricLog.entity";

// ✅ Define input/output types
export interface SaveMetricLogsParams {
  logDate: Date;
  inserts: { metric_id: MetricId; value: number | null }[];
  deletes: MetricId[];
}

@injectable()
export class SaveMetricLogsUseCase {
  constructor(
    @inject("MetricLogRepository")
    private repository: MetricLogRepository,
  ) {}

  // ✅ Returns Promise instead of createAsyncThunk
  async execute(params: SaveMetricLogsParams): Promise<SuccessMessage> {
    const { logDate, inserts, deletes } = params;
    return await this.repository.saveMetricLogs(logDate, inserts, deletes);
  }
}
```

Changes:

- ✅ Removed `createAsyncThunk` import
- ✅ Changed `execute` from thunk to async method

Step 2: Create Redux Thunk File

Create new file for Redux integration:

`src/presentation/state/thunks/metricLog.thunks.ts` (NEW FILE)

Note: Import container from `src/container.ts` (it lives outside the 4-layer structure)

```typescript
import { createAsyncThunk } from "@reduxjs/toolkit";
import { container } from "tsyringe";
import { SaveMetricLogsUseCase } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";
import type { SaveMetricLogsParams } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";
import type { SuccessMessage } from "@/domain/repositories/MetricLogRepository";

// ✅ Redux thunk in Presentation layer
export const saveMetricLogs = createAsyncThunk<
  SuccessMessage,
  SaveMetricLogsParams,
  { rejectValue: string }
>("metricLog/save", async (params, { rejectWithValue }) => {
  try {
    const useCase = container.resolve(SaveMetricLogsUseCase);
    return await useCase.execute(params);
  } catch (error) {
    const errorMessage =
      error instanceof Error ? error.message : "Failed to save metric logs";
    return rejectWithValue(errorMessage);
  }
});

// Add other thunks here:
// export const listMetricLogs = createAsyncThunk(...)
// export const deleteMetricLog = createAsyncThunk(...)
```

What this does:

- ✅ Thunk resolves use case
- ✅ Thunk handles Redux-specific logic (`rejectWithValue`)

Step 3: Move StateManager to Presentation

Move the entire folder:

```
# Move folder
mv src/application/StateManager src/presentation/state

# Update imports across ~50 files
# From: @/application/StateManager
# To:   @/presentation/state
```

Step 4: Update Redux Slice

Update slice to use new thunk:

`src/presentation/state/slices/metricLog/metricLogSlice.ts` (UPDATED)

```typescript
import { createSlice } from "@reduxjs/toolkit";
import { saveMetricLogs } from "../../thunks/metricLog.thunks";

interface MetricLogState {
  logs: any[];
  loading: boolean;
  error: string | null;
}

const initialState: MetricLogState = {
  logs: [],
  loading: false,
  error: null,
};

const metricLogSlice = createSlice({
  name: "metricLog",
  initialState,
  reducers: {
    // Regular reducers here
  },
  extraReducers: (builder) => {
    // ✅ Use imported thunk instead of resolving from container
    builder
      .addCase(saveMetricLogs.pending, (state) => {
        state.loading = true;
        state.error = null;
      })
      .addCase(saveMetricLogs.fulfilled, (state, action) => {
        state.loading = false;
        // Handle success - action.payload is SuccessMessage
      })
      .addCase(saveMetricLogs.rejected, (state, action) => {
        state.loading = false;
        state.error = action.payload || "Failed to save metric logs";
      });
  },
});
```

Step 5: Update ViewModel

Update hook to use new thunk:

`src/presentation/viewModels/MetricLog/useMetricLog.ts` **(UPDATED)**

```ts
import { useCallback } from "react";
import { useAppDispatch } from "@/presentation/state/store";
import { saveMetricLogs } from "@/presentation/state/thunks/metricLog.thunks";
import type { SaveMetricLogsParams } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";

export const useMetricLog = () => {
  const dispatch = useAppDispatch();

  const handleSaveMetricLogs = useCallback(
    async (params: SaveMetricLogsParams) => {
      try {
        await dispatch(saveMetricLogs(params)).unwrap();
      } catch (error) {
        console.error("Failed to save metric logs:", error);
        throw error; // Re-throw for component to handle
      }
    },
    [dispatch],
  );

  return { saveMetricLogs: handleSaveMetricLogs };
};
```

Changes:

- ✅ Import thunk instead of use case
- ✅ Dispatch thunk directly
- ✅ No more `container.resolve()`
- ✅ Cleaner dependencies

Step 6: Update Tests

Before: Testing was complex

```
// ❌ BEFORE: Must mock Redux
import { SaveMetricLogsUseCase } from "../SaveMetricLogs.useCase";

describe("SaveMetricLogsUseCase", () => {
  it("should save logs", async () => {
    // Complex Redux mocking required
    const mockDispatch = jest.fn();
    const mockRejectWithValue = jest.fn();
    // ...
  });
});
```

After: Testing is simple

```
// ✅ AFTER: Pure function testing
import { SaveMetricLogsUseCase } from "../SaveMetricLogs.useCase";
import type { MetricLogRepository } from "@/domain/repositories/MetricLogRepository";

describe("SaveMetricLogsUseCase", () => {
  let useCase: SaveMetricLogsUseCase;
  let mockRepository: jest.Mocked<MetricLogRepository>;

  beforeEach(() => {
    mockRepository = {
      saveMetricLogs: jest.fn(),
    } as any;

    useCase = new SaveMetricLogsUseCase(mockRepository);
  });

  it("should save metric logs successfully", async () => {
    // Arrange
    const params = {
      logDate: new Date("2024-01-15"),
      inserts: [{ metric_id: "mood", value: 5 }],
      deletes: [],
    };
    const mockResponse = { success: true, message: "Saved" };
    mockRepository.saveMetricLogs.mockResolvedValue(mockResponse);

    // Act
    const result = await useCase.execute(params);

    // Assert
```

Benefits Realized

✅ Framework Independence

Can now use in scripts:

```typescript
// scripts/export-metrics.ts
import { container } from "tsyringe";
import { SaveMetricLogsUseCase } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";

async function exportMetrics() {
  const useCase = container.resolve(SaveMetricLogsUseCase);

  const result = await useCase.execute({
    logDate: new Date(),
    inserts: [
      /* ... */
    ],
    deletes: [],
  });

  console.log("Metrics saved:", result);
}
```

✅ Easier Testing

Unit test (pure function):

```
const useCase = new SaveMetricLogsUseCase(mockRepository);
const result = await useCase.execute(params);
expect(result).toEqual(expected);
```

Integration test (with Redux):

```
const store = mockStore({ metricLog: initialState });
await store.dispatch(saveMetricLogs(params));
expect(store.getActions()[1].type).toBe("fulfilled");
```

✅ Better Separation of Concerns

Use Case: Pure business logic
Thunk: Redux integration
Slice: State updates
ViewModel: UI coordination

✅ Alternative State Management

Could switch to Zustand:

```typescript
// store/metricLogStore.ts (Zustand example)
import create from "zustand";
import { container } from "tsyringe";
import { SaveMetricLogsUseCase } from "@/application/useCases/metricLog/SaveMetricLogs.useCase";

export const useMetricLogStore = create((set) => ({
  logs: [],
  loading: false,

  saveMetricLogs: async (params) => {
    set({ loading: true });

    const useCase = container.resolve(SaveMetricLogsUseCase);
    const result = await useCase.execute(params);

    set({ loading: false });
    return result;
  },
}));
```

No changes needed to use case!

Migration Checklist

For each use case, follow these steps:

- [ ] **Step 1:** Refactor use case

  - [ ] Remove `createAsyncThunk` import
  - [ ] Change `execute` to async method
  - [ ] Add params interface
  - [ ] Return `Promise<T>`
  - [ ] Remove Redux-specific error handling

- [ ] **Step 2:** Create thunk file

  - [ ] Create `src/presentation/state/thunks/[domain].thunks.ts`
  - [ ] Export `createAsyncThunk` for use case
  - [ ] Handle errors with `rejectWithValue`

- [ ] **Step 3:** Update slice

  - [ ] Import thunk from thunks file
  - [ ] Replace `container.resolve()` with imported thunk
  - [ ] Update case handlers

- [ ] **Step 4:** Update ViewModel

  - [ ] Import thunk instead of use case
  - [ ] Dispatch thunk directly
  - [ ] Remove `container.resolve()`

Timeline Estimate

For this one use case:

- Step 1 (Refactor use case): **15 minutes**
- Step 2 (Create thunk): **10 minutes**
- Step 3 (Update slice): **5 minutes**
- Step 4 (Update ViewModel): **5 minutes**
- Step 5 (Update tests): **20 minutes**
- Step 6 (Verify): **5 minutes**

Total per use case: ~60 minutes

For 24 use cases: ~24 hours (3 days of focused work)

Quick Reference

Use Case Pattern

```typescript
// ✅ Application layer
export interface UseParams {
  /* ... */
}
export interface UseResult {
  /* ... */
}

@injectable()
export class DoSomethingUseCase {
  async execute(params: UseParams): Promise<UseResult> {
    // Pure business logic
  }
}
```

Thunk Pattern

```typescript
// ✅ Presentation layer
export const doSomething = createAsyncThunk<UseResult, UseParams>(
  "domain/action",
  async (params, { rejectWithValue }) => {
    try {
      const useCase = container.resolve(DoSomethingUseCase);
      return await useCase.execute(params);
    } catch (error) {
      return rejectWithValue(error.message);
    }
  },
);
```

Slice Pattern

Next: Apply this pattern to all 24 use cases listed in the main report.

Last Updated: 2025-11-19

See Also:

- Architecture Analysis Report
- Architecture Quick Reference