PHASE_2_ROADMAP

1 Djed Phase 2 Roadmap

Built on the success of @djed/logger v0.1.0

1.1 🎯 Vision

Expand Djed from a single package to a comprehensive infrastructure ecosystem that accelerates LUXOR project development with production-ready, well-tested packages and templates.

Core Principle: Ship fast, iterate based on real-world usage.

1.2 📊 Phase 1 Learnings (What Worked)

1.2.1 ✅ Successes from @djed/logger

1. **Progressive API Design** (L1→L3) - Users can grow with the package
2. **Comprehensive Testing** - 35 tests, 100% coverage = confidence
3. **Rich Documentation** - 14 docs covering every angle
4. **Quality Gates** - Automated validation (bundle size, tests, security)
5. **Fast Shipping** - MVP to production in focused timeline

1.2.2 💡 Apply to Phase 2

- **All packages** use progressive API (L1 novice → L3 expert)
- **All packages** ship with 100% test coverage
- **All packages** include comprehensive docs
- **All packages** validated before publishing

1.3 🚀 Phase 2A: Core Infrastructure (Next 2-4 Weeks)

1.3.1 Package 2: @djed/config (Priority: HIGH)

Problem: Every project needs configuration management (env vars, secrets, validation)

Solution: Type-safe configuration loader with validation

Features:

```
// L1: Novice (Zero config)
import { loadConfig } from '@djed/config';
const config = loadConfig();  // Loads from .env automatically

// L2: Intermediate (Schema validation)
import { loadConfig, z } from '@djed/config';
const config = loadConfig({
  schema: z.object({
    PORT: z.number().default(3000),
    DATABASE_URL: z.string().url(),
    API_KEY: z.string().min(32)
  })
});

// L3: Expert (Multiple sources, transforms)
const config = loadConfig({
  sources: ['.env', '.env.local', process.env],
  transforms: { PORT: Number },
  validation: 'strict',
  secrets: ['API_KEY', 'DATABASE_URL']
});
```

Why Next? - Used in 100% of LUXOR projects - Pairs naturally with @djed/logger (log config loading) - Prevents common mistakes (missing env vars, type errors)

Timeline: 1 week - Day 1-2: Core implementation + tests - Day 3-4: Documentation + demos - Day 5: Quality review + publish

Dependencies: zod (for schema validation)

1.3.2 Package 3: @djed/errors (Priority: HIGH)

Problem: Inconsistent error handling across projects

Solution: Structured error classes with context

Features:

```
// L1: Novice (Simple error classes)
import { NotFoundError, ValidationError } from '@djed/errors';

throw new NotFoundError('User not found');
throw new ValidationError('Invalid email format');

// L2: Intermediate (Context + serialization)
throw new NotFoundError('User not found', {
  userId: 123,
  requestId: 'abc-123'
});

// L3: Expert (Custom error classes)
import { BaseError } from '@djed/errors';

class PaymentError extends BaseError {
  constructor(message, context) {
    super(message, context);
    this.name = 'PaymentError';
    this.statusCode = 402;
  }
}
```

Why Next? - Pairs with @djed/logger (structured error logging) - Needed for API responses (consistent error format) - Common pattern across all projects

Timeline: 1 week

Dependencies: None (pure TypeScript)

1.3.3 Package 4: @djed/http-client (Priority: MEDIUM)

Problem: Repetitive HTTP client setup (retry, timeout, logging)

Solution: Pre-configured HTTP client with smart defaults

Features:

```
// L1: Novice (Zero config)
import { createClient } from '@djed/http-client';

const api = createClient('https://api.example.com');
const user = await api.get('/users/123');

// L2: Intermediate (Retry + timeout)
const api = createClient('https://api.example.com', {
  timeout: 5000,
  retry: { attempts: 3, backoff: 'exponential' },
  logger: myLogger  // Integrates with @djed/logger
});

// L3: Expert (Interceptors + custom handling)
const api = createClient('https://api.example.com', {
  interceptors: {
    request: (config) => addAuthHeader(config),
    response: (res) => unwrapData(res),
    error: (err) => handleApiError(err)
  }
});
```

Why Next? - Common need (every project calls external APIs) - Integrates with @djed/logger and @djed/errors - Reduces boilerplate

Timeline: 1 week

Dependencies: axios or fetch-based

1.4 🏗 Phase 2B: Templates (Weeks 5-6)

1.4.1 Template 1: mcp-server-minimal (Priority: CRITICAL)

Problem: Creating MCP servers from scratch is repetitive

Solution: Minimal template with best practices built-in

Structure:

```
mcp-server-minimal/
├── src/
│   ├── index.ts          # Server entry
│   ├── tools/            # Tool implementations
│   └── resources/        # Resource handlers
├── tests/
│   └── server.test.ts    # Integration tests
├── package.json
├── tsconfig.json
└── README.md
```

Features: - Uses @djed/logger for logging - Uses @djed/config for configuration - Uses @djed/errors for error handling - 100% typed with TypeScript - Test suite included - Ready to integrate with Claude Code

Why Critical? - Validates the Djed ecosystem (packages work together) - High-value output (accelerates MCP development) - Showcases best practices

Timeline: 4 days - Day 1-2: Template structure + implementation - Day 3: Tests + documentation - Day 4: Integration testing with real MCP use case

---

1.4.2 Template 2: express-api-starter (Priority: MEDIUM)

Problem: Setting up Express APIs with best practices takes time

Solution: Production-ready Express template

Features: - Express server with middleware - @djed/logger for request logging - @djed/config for environment variables - @djed/errors for error handling - @djed/http-client for external API calls - TypeScript strict mode - Test suite with supertest - Docker support

Timeline: 5 days

1.5 📦 Phase 2C: Advanced Packages (Weeks 7+)

1.5.1 Package 5: @djed/telemetry (Priority: LOW)

Problem: No standardized metrics/tracing

Solution: OpenTelemetry wrapper

Features:

```
import { createTelemetry } from '@djed/telemetry';

const telemetry = createTelemetry({
  service: 'my-api',
  exporters: ['console', 'jaeger']
});

// Automatic instrumentation
telemetry.instrument(app);  // Express auto-instrumentation
```

1.5.2 Package 6: @djed/cache (Priority: LOW)

Problem: Caching implementations vary widely

Solution: Multi-backend cache abstraction

Features:

```
import { createCache } from '@djed/cache';

// L1: In-memory cache
const cache = createCache();

// L2: Redis backend
const cache = createCache({ backend: 'redis', url: REDIS_URL });

// L3: Multi-tier (memory → redis)
const cache = createCache({
  tiers: [
    { backend: 'memory', ttl: 60 },
    { backend: 'redis', ttl: 3600 }
  ]
});
```

1.6 🎯 Phase 2 Success Criteria

1.6.1 Package Quality Gates (All packages must meet)

- ✅ Bundle size: < 10 KB (gzipped)
- ✅ Test coverage: > 90%
- ✅ TypeScript: 100% typed
- ✅ Documentation: README + API docs + examples
- ✅ Zero vulnerabilities
- ✅ Progressive API (L1 → L2 → L3)

1.6.2 Ecosystem Health

- ✅ All packages work together (validated via templates)
- ✅ Consistent APIs across packages
- ✅ Shared testing utilities

- ✅ Centralized documentation

1.6.3 Adoption Metrics (Target)

- 🎯 npm downloads: > 100/week per package
- 🎯 GitHub stars: > 50 (repo total)
- 🎯 Internal usage: Used in 3+ LUXOR projects
- 🎯 Community engagement: 5+ external issues/PRs

1.7 📅 Timeline Summary

Week	Focus	Deliverables
Week 1	@djed/config	Package published, 100% tested
Week 2	@djed/errors	Package published, 100% tested
Week 3	@djed/http-client	Package published, 100% tested
Week 4	Integration	All packages work together
Week 5	mcp-server-minimal	Template published
Week 6	express-api-starter	Template published
Week 7+	Advanced packages	Based on demand

1.8 🔄 Iteration Strategy

1.8.1 Release Cadence

- **Major releases** (x.0.0): New packages/templates
- **Minor releases** (0.x.0): New features
- **Patch releases** (0.0.x): Bug fixes

1.8.2 Feedback Loop

1. **Internal dogfooding**: Use in LUXOR projects
2. **Public release**: npm + GitHub
3. **Monitor usage**: Downloads, issues, feedback
4. **Iterate**: Based on real-world pain points

1.8.3 Prioritization Framework

High Priority = High usage × High pain × Low complexity **Medium Priority** = High usage × Medium pain × Medium complexity **Low Priority** = Nice-to-have or complex to build

1.9 💰 Resource Allocation

1.9.1 Time Budget (Per Package)

- **Implementation**: 2-3 days
- **Testing**: 1 day
- **Documentation**: 1 day
- **Quality review**: 0.5 day
- **Publishing**: 0.5 day

Total per package: ~5-7 days

1.9.2 Parallel Work Opportunities

- Documentation can start on Day 2 (while tests run)
- Multiple packages can be in different phases
- Templates can be scaffolded early (structure first)

1.10 🎓 Learning Goals

1.10.1 Technical Skills Developed

- Monorepo management (Lerna/Turborepo)
- Package publishing workflows
- API design patterns
- Testing strategies
- Documentation best practices

1.10.2 Ecosystem Building

- Community engagement
- Issue triage and support
- Contribution guidelines
- Release management

1.11 🚨 Risk Mitigation

1.11.1 Known Risks

1. Scope Creep - **Risk**: Packages grow too complex - **Mitigation**: Strict L1→L2→L3 progression; L1 ships first

2. Maintenance Burden - **Risk**: Too many packages to maintain - **Mitigation**: Only ship high-value packages; deprecate unused ones

3. Breaking Changes - **Risk**: APIs change, breaking users - **Mitigation**: Semantic versioning; deprecation warnings; migration guides

4. Low Adoption - **Risk**: Nobody uses the packages - **Mitigation**: Internal dogfooding first; promote via blog posts

1.12 📈 Metrics Dashboard (Track Weekly)

1.12.1 Package Health

- npm weekly downloads
- GitHub stars/forks
- Open issues vs closed
- Test coverage %
- Bundle size (KB)

1.12.2 Community

- External contributors
- Pull requests merged
- Issue response time

- Documentation views

1.12.3 Internal Usage

- ## LUXOR projects using Djed
- ## packages per project
- Developer satisfaction (survey)

1.13 🎉 Phase 2 Success Looks Like

By End of Phase 2: - ✅ 6 published packages (@djed/logger + 5 more) - ✅ 2 production templates - ✅ Used in 3+ LUXOR projects - ✅ 100+ weekly npm downloads (total) - ✅ 50+ GitHub stars - ✅ Comprehensive documentation site - ✅ Active community (issues, PRs)

Outcome: Djed becomes the default infrastructure for new LUXOR projects.

1.14 🚀 Next Immediate Actions

1.14.1 Week 1 Kickoff (Starting Now)

Monday: 1. Create `@djed/config` package structure 2. Implement L1 API (zero config) 3. Write initial tests

Tuesday: 4. Implement L2 API (schema validation) 5. Complete test coverage (100%)

Wednesday: 6. Implement L3 API (advanced features) 7. Write documentation (README, API docs, examples)

Thursday: 8. Quality review (bundle size, security, performance) 9. Integration testing with @djed/logger

Friday: 10. Publish to npm 11. Update Djed monorepo README 12. Announce internally (Slack, email)

1.15 📚 Reference Documents

From Phase 1: - Djed Infrastructure Spec v1.1 - @djed/logger Implementation - Publishing Workflow

Templates: - Use @djed/logger as quality template - Follow same structure for all new packages

Phase 2 Start Date: 2025-11-04 **Expected Completion**: 2025-12-15 (6 weeks) **Status**: 🟢 READY TO START

Generated 2025-11-03 - This is a living document. Update based on learnings and real-world usage.