Flutter.js Execution Plan v2.0

Unified Strategy with Reactivity & Obfuscation

Timeline: 22 Weeks to MVP

October 2, 2025

Executive Summary
Flutter.js transpiles Flutter apps to optimized, SEO-friendly HTML/CSS/JS while maintaining 95%+ Material Design fidelity. The system handles all widget types (stateless, stateful, functional), manages reactivity, and produces heavily obfuscated production builds.

Key Differentiator: Flutter Design Systems for HTML, not a Flutter Web replacement.

Contents

1 Phase 0: Foundation & Validation (Week 1-2)

1.1 Goal

Prove technical feasibility before months of development.

1.2 Tasks

1. **Manual Proof of Concept**
    - Transform Container + Text widget manually
    - Transform ElevatedButton with Material Design tokens
    - Create FlatBuffers IR structure
    - Generate HTML/CSS output
    - Compare against Flutter Web and native
2. **Constraint Documentation**
    - CustomPaint widgets limitations
    - Complex animations constraints
    - CSS vs Flutter layout differences
    - Material elevation system in CSS
3. **Research Existing Solutions**
    - Flutter Web (canvas approach)
    - Hummingbird project
    - React Native Web strategy
    - Material Web Components
    - Cupertino CSS libraries

1.3 Success Criteria

- Visual identity preserved?
- Material Design tokens correct?
- Bundle size significantly smaller?
- Decision: Go/No-Go

2 Phase 1: Core Parser with Reactivity Analysis (Week 3-6)

2.1 Milestone 1: Widget Classification System

Parse and categorize all widget types:

Widget Types

```
Widget Types Detected:
- Type A: Pure StatelessWidget -> Static HTML
- Type B: StatefulWidget -> HTML + Runtime
- Type C: Function widgets -> Inline expansion
- Type D: Non-UI classes -> JavaScript modules
- Type E: InheritedWidget -> CSS vars + context
```

Enhanced Detection:

- Material vs Cupertino usage
- Theme references (Theme.of(context))
- Design token usage (Colors.blue[500])
- State management (Provider/Riverpod)
- Widget dependencies and data flow

2.2 Milestone 2: Reactivity Analysis

Track all data dependencies:

- Constructor parameters from parent
- InheritedWidget dependencies (Theme, MediaQuery)
- Global state dependencies (Provider)
- State variable usage
- Prop propagation to children

Output: Reactivity graph showing what triggers each widget's re-render.

2.3 Milestone 3: Property Extraction

Extract and categorize:

- Basic properties (strings, numbers, booleans)
- Complex objects (EdgeInsets, TextStyle, Color)
- Design system properties (Material tokens)
- Theme property resolution
- Prop usage analysis (used in conditionals, loops, text, styles)

2.4 Test Cases

- StatelessWidget with props (parent state change)
- StatefulWidget with setState
- Function returning widget
- Non-UI service class

- Widget with Theme dependency
- Provider consumer widget

3 Phase 1.5: Route & Page Analysis (Week 6.5)

3.1 Milestone: Multi-Page Architecture

Detect and map:

- Navigator routes and definitions
- Page-level vs component widgets
- Lazy-loaded routes for code splitting
- Route parameters and navigation flow

Output Strategy:

Output Strategy

```
Single-Page App -> index.html + inline components
Multi-Page App  -> Separate HTML per route
                -> Shared components as modules
```

4 Phase 2: Enhanced IR Design (Week 7-8)

4.1 Milestone 4: Comprehensive Schema

Widget Schema

```
table Widget {
  id: string;
  type: string;
  classification: WidgetType;  // A, B, C, D, E
  is_stateful: bool;

  // Reactivity tracking
  data_dependencies: DataDependencies;
  reactivity_info: ReactivityInfo;
  prop_usage: [PropUsage];

  // Design system
  material_theme_refs: [ThemeRef];
  design_system: DesignSystem;

  properties: [Property];
  children: [Widget];
}

table DataDependencies {
  constructor_params: [Parameter];
  theme_dependencies: [string];
  mediaquery_dependencies: [string];
  provider_dependencies: [ProviderRef];
  derived_values: [DerivedValue];
}

table ReactivityInfo {
  triggers: [ReactivityTrigger];       // What causes re-render
  affected_properties: [string];        // Which props change
  propagates_to_children: bool;
  re_render_strategy: RenderStrategy;
}

table ClassDefinition {
  type: ClassType;  // Widget, State, Model, Service, Helper
  has_ui: bool;
  methods: [Method];
  dependencies: [ClassRef];
}
```

4.2 Milestone 5: Multi-File IR Generation

- Separate IR per page/route
- Shared component definitions

- Non-UI class metadata
- Design token centralization

5 Phase 3: Flutter-Identity Transpiler (Week 9-12)

5.1 Milestone 6: Widget Mapping System (Week 9)

30 Core Widgets:

- **Tier 1 (Pixel-perfect, 15):** Container, Text, Column, Row, Scaffold, AppBar, Center, Padding, SizedBox, ElevatedButton, TextButton, Icon, Image, Stack, Positioned
- **Tier 2 (Close match, 10):** Card, ListTile, TextField, Checkbox, Switch, IconButton, FloatingActionButton, BottomNavigationBar, Divider, CircularProgressIndicator
- **Tier 3 (Best effort, 5):** Wrap, Flexible, Expanded, Align, AspectRatio

Design Integration:

- Material Design 3 tokens (colors, typography, elevation, shapes, motion)
- Cupertino design tokens (system colors, SF Pro, iOS effects)

5.2 Milestone 7: CSS Generation Engine (Week 10)

Hybrid Strategy:

CSS Strategy

```
Material/Cupertino tokens -> CSS custom properties
Common patterns -> .flutter-* utility classes
Widget-specific -> Inline styles
Theme system -> CSS variables + dark mode
```

Converters:

- EdgeInsets -> padding/margin (all variants)
- BoxDecoration -> CSS (gradients, shadows, borders)
- TextStyle -> Typography (with Material tokens)
- Theme -> CSS custom properties + inheritance

5.3 Milestone 8: Flutter.js Runtime with Reactivity (Week 11-12)

Core Runtime Modules (<15KB total):

Runtime Modules

```
// 1. Reactivity System
FlutterJS.reactivity
- Widget registry
- Dependency tracking
- Render scheduling (batched)
- Prop change detection
- Parent-child propagation

// 2. State Management
FlutterJS.state
- setState (Flutter-like)
- Widget lifecycle hooks
- InheritedWidget context
- Provider/Riverpod bridge

// 3. Event System
FlutterJS.events
- Flutter callbacks -> DOM events
- GestureDetector patterns
- Event object mapping

// 4. Navigation
FlutterJS.navigation
- push/pop with history
- Named routes
- Browser back button

// 5. Theme & Context
FlutterJS.context
- Theme.of(context)
- MediaQuery.of(context)
- CSS variable access
```

Key Feature: Reactive Updates

- StatelessWidget re-renders when parent props change
- Efficient prop diffing (only changed props trigger update)
- Automatic child notification
- InheritedWidget change propagation

6 Phase 4: Integration & Testing (Week 13-14)

6.1 Milestone 9: End-to-End Pipeline

Build Pipeline

```
flutter-js build
- Parse all widget types (Phase 1)
- Analyze reactivity (Phase 1)
- Generate enhanced IR (Phase 2)
- Transpile with reactivity system (Phase 3)
- Output optimized build/
```

6.2 Milestone 10: Comprehensive Test Suite

Reactivity Tests:

1. Stateless widget with props -> parent state change
2. StatefulWidget -> setState updates
3. Function widget -> inline expansion
4. Non-UI class -> ES6 module
5. Theme change -> cascading updates
6. Provider change -> consumer updates
7. Conditional rendering -> prop-based
8. List updates -> key-based reconciliation
9. Multi-page navigation -> route parameters
10. Form validation -> stateful interactions

Benchmarks:

- Bundle size: Flutter Web (2.1MB) vs Flutter.js (<50KB)
- Load time (3G): 12s vs <2s
- Time to Interactive: 15s vs <3s
- SEO Score: 40 vs >90
- Visual Fidelity: >95%

7 Phase 5: Refinement (Week 15)

Refinement Tasks

Iteration based on Phase 4 results:

- Fidelity <95% -> Fix Material tokens, CSS generation
- Bundle too large -> Tree-shaking, runtime optimization
- Performance issues -> Profile and optimize hot paths
- Reactivity bugs -> Fix dependency tracking

8 Phase 6: Obfuscation & Production Optimization (Week 16-18)

8.1 Build Modes

Build Modes

```
# Production (default): Always obfuscated
flutter-js build

# Development: Readable code
flutter-js build --mode=dev
```

8.2 Obfuscation Pipeline

JavaScript:

- Variable/function/class name mangling
- Property name mangling
- String array encoding
- Dead code elimination
- Control flow flattening (aggressive)
- Remove console/debugger statements

HTML:

- Class name shortening (.flutter-container -> .c0)
- Data attribute compression
- Whitespace removal
- Optional tag removal

CSS:

- Selector shortening (matching HTML)
- Custom property minification
- Value optimization (colors, units)
- Unused CSS removal (tree-shaking)

8.3 Output Comparison

Output Comparison

```
Development Build:
- index.html (15 KB, readable)
- flutter.js (45 KB, readable)
- widgets.js (30 KB, readable)
- app.js (25 KB, readable)
- styles.css (20 KB, readable)
Total: 135 KB

Production Build:
- index.html (3 KB, minified)
- app.min.js (28 KB, obfuscated)
- styles.min.css (6 KB, minified)
Total: 37 KB (72% reduction)
Gzipped: 12 KB (91% reduction)
```

8.4 Smart File Generation

File Structure

```
build/
|-- index.html
|-- pages/
|   |-- home.html              (Stateless -> pre-rendered)
|   |-- profile.js             (Stateful -> dynamic)
|-- components/
|   |-- stateless/             (Pure HTML)
|   |-- stateful/              (JS classes)
|-- models/                    (Non-UI classes)
|-- services/                  (Business logic)
|-- styles/
|   |-- material.css           (Design tokens)
|   |-- components.css         (Widget styles)
|-- runtime/
    |-- flutter.min.js         (Obfuscated runtime)
```

9 Phase 7: Production Testing (Week 19)

9.1 Validation

Validation Criteria

Security:
- Code unreadable to humans
- No source maps
- No debug info
- All names mangled

Performance:
- Bundle <50KB uncompressed
- Bundle <15KB gzipped
- Load time <2s (3G)
- TTI <3s

Functionality:
- Obfuscated code functions identically
- Reactivity works correctly
- Navigation functions
- State management intact
- Theme switching works

10 Phase 8: MVP Launch (Week 20)

10.1 Minimum Viable Product

MVP Features

Supported:
- 30 widgets (15 perfect, 10 close, 5 best-effort)
- Material Design 3 + basic Cupertino
- setState + lifecycle hooks
- Navigation system
- Theme system + dark mode
- Form inputs + validation
- Reactive prop updates
- Production obfuscation

Bundle:
- <50KB uncompressed
- <15KB gzipped
- Semantic HTML
- WCAG AA compliant
- SEO-friendly

Fidelity:
- 95%+ for Material widgets
- Indistinguishable in common cases

10.2 Target Use Cases

- Landing pages, marketing sites
- Content-heavy apps
- Forms and CRUD interfaces

- Multi-page applications

NOT For:

- Canvas-heavy apps
- Complex CustomPaint
- Games
- Apps needing native APIs

11 Timeline Summary

Timeline

```
Week 1–2:   Phase 0 – Foundation & Validation
Week 3–6:   Phase 1 – Parser with Reactivity
Week 6.5:   Phase 1.5 – Route Analysis
Week 7–8:   Phase 2 – Enhanced IR
Week 9–12:  Phase 3 – Transpiler + Runtime
Week 13–14: Phase 4 – Integration & Testing
Week 15:    Phase 5 – Refinement
Week 16–18: Phase 6 – Obfuscation
Week 19:    Phase 7 – Production Testing
Week 20:    Phase 8 – MVP Launch
Week 21–22: Documentation & Release
```

12 Evaluation Checkpoints

Week	Checkpoint	Success Criteria
2	Phase 0 Complete	Visual identity maintained?
6	Phase 1 Complete	Theme data extracted? >80% success?
8	Phase 2 Complete	IR preserves design + reactivity info?
12	Phase 3 Complete	Output resembles Flutter? Reactivity works?
14	Phase 4 Complete	50% smaller, faster, 95%+ fidelity, better SEO?
18	Phase 6 Complete	70%+ size reduction? Code obfuscated?

13 Risk Mitigation

14 Key Technical Decisions

1. **Reactivity Architecture**
   - *Choice:* Virtual DOM-like diffing with dependency tracking
   - *Rationale:* Efficient updates, matches Flutter's rebuild behavior
   - *Trade-off:* Small runtime overhead for correctness
2. **Output Strategy**

Risk	**Mitigation**
Can't maintain Material fidelity	Use official MD3 tokens, visual regression tests, document differences
CSS can't replicate layouts	Focus on supported patterns, document limitations, provide escape hatches
Reactivity too complex	Start simple (props only), iterate to advanced (Provider), thorough testing
Runtime too large	Modular design, tree-shaking, focus on core features
Obfuscation breaks code	Extensive testing, preserve critical names, escape hatch annotations
Development takes too long	Ship Tier 1 first, accept imperfect Tier 2/3, iterate on feedback

Output Strategy

```
Widget Analysis -> Output Decision:
- Pure stateless -> Static HTML (no JS)
- Props from parent -> HTML + update function
- Has state -> HTML + full runtime
- Function widget -> Inline expansion
- Non-UI class -> ES6 module
```

3. **Obfuscation Level**
   - *Default:* Aggressive (name mangling, string encoding, dead code injection)
   - *Rationale:* Protect source, minimize size
   - *Trade-off:* Longer build time (+3s for large projects)
4. **Build Output**

Build Output

```
Production: Single-page app (SPA) by default
Optional: Multi-page app (MPA) for SEO-heavy sites
Strategy: Detect from MaterialApp routes configuration
```

15 Success Definition

15.1 6-Month Goal Achieved When:

Success Criteria

Technical:
- 30 widgets supported
- 95%+ Material Design fidelity
- <50KB bundle (production)
- <15KB gzipped
- Reactivity system working
- Navigation functional
- Code fully obfuscated

Quality:
- Semantic HTML
- WCAG AA compliant
- SEO score >90
- Load time <2s (3G)
- TTI <3s

Usability:
- Simple CLI
- Clear documentation
- Example projects
- Migration guide

16 Final Reality Check

Before starting, confirm:

1. Can you commit 15-20 hours/week for 6 months?
2. Are you comfortable with 95% fidelity (not 100%)?
3. Will you accept limited widget support initially?
4. Can you maintain Material Design spec compliance?
5. Do you have test Flutter apps to validate against?
6. Do you understand obfuscation trade-offs?
7. Can you handle reactivity complexity?

If all YES: Start Phase 0 with Material button + reactivity proof.
If any NO: Reconsider scope or timeline.

17 Appendix: Command Reference

Command Reference

```
# Development build (readable code)
flutter-js build --mode=dev

# Production build (obfuscated, default)
flutter-js build

# Maximum compression
flutter-js build --compress-max

# Compare builds
flutter-js build --compare

# Analyze bundle
flutter-js analyze

# Generate documentation
flutter-js docs
```

Document Information

Success Criteria

Document Version: 2.0
Last Updated: October 2, 2025
Status: Ready for Phase 0 execution