Flutter.js Execution Plan: Unified Strategy

Merging Foundation Blueprint with Flutter-Identity Phase 3

October 2, 2025

A comprehensive plan for building a Flutter-to-Web transpiler with Material Design fidelity

Introduction

This document outlines a unified execution plan for Flutter.js, merging the foundation blueprint with Flutter-identity Phase 3 to create a coherent path from zero to a working prototype. The goal is to maintain Flutter's visual identity while leveraging web technologies, producing semantic, SEO-friendly, and lightweight HTML/CSS/JS output. The plan spans 20 weeks, targeting a Minimum Viable Product (MVP) that supports 30 core widgets with 95%+ Material Design fidelity.

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

1.1 Goal

Prove technical feasibility before committing to months of development.

1.2 Tasks

1.2.1 Manual Proof of Concept

- Transform a simple Flutter `Container` with `Text` widget manually.
- Create FlatBuffers Intermediate Representation (IR) structure.
- Generate HTML/CSS output using Material Design tokens.
- Compare against Flutter Web and hand-written HTML.
- Additional task: Transform a `Material ElevatedButton` to verify design system consistency.

1.2.2 Success Criteria

- Can transformation rules preserve Flutter's visual identity?
- Does output correctly use Material Design tokens?
- Is bundle size significantly smaller while maintaining fidelity?

1.2.3 Constraint Analysis

Document limitations:

- `CustomPaint` widgets.
- Complex animations using Flutter's animation framework.
- Widgets dependent on Flutter's precise layout engine.
- Platform-specific rendering behaviors.

Flutter-specific constraints:

- Material components not replicable in CSS.
- Differences between CSS flexbox and Flutter's constraints.
- Preservation of Material elevation system in CSS.

1.2.4 Research Existing Solutions

Study:

- Flutter Web's canvas-based approach.
- Hummingbird project (Flutter's previous web approach).
- React Native Web's transformation strategy.
- Material Web Components' Material Design implementation.
- Cupertino CSS libraries for iOS styling.

Focus: Maintaining design system fidelity during transformation.

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

2.1 Milestone 1: Static Widget Parsing

Build a Dart program to:

1. Read `.dart` files.
2. Extract widget tree using `analyzer` package.
3. Identify widget types and properties.
4. Print tree structure.

Enhanced requirements: Detect:

- Material vs. Cupertino widget usage.
- Theme references (`Theme.of(context)`).
- Design token usage (`Colors.blue`, `TextTheme.headline1`).

Test cases:

- Simple `StatelessWidget` with `Container` + `Text`.
- Widget using Material components (`Card`, `AppBar`).
- Widget referencing `ThemeData`.
- Cupertino widget (`CupertinoButton`).

2.2 Milestone 2: Property Extraction

Extract and categorize:

- Basic properties: strings, numbers, booleans.
- Complex objects: `EdgeInsets`, `TextStyle`, `Color`.
- Design system properties:
  - Material color references (`Colors.blue[500]`).
  - Typography references (`Theme.of(context).textTheme.headline1`).
  - Elevation values, border radius, animation curves.

Handle expressions in IR for:

- Material Design token lookup.
- Theme property resolution.
- Color shade calculation.

2.3 Milestone 3: State Management Detection

Identify:

- `StatefulWidget` vs. `StatelessWidget`.
- State variables and their types.
- `setState` calls and modified properties.
- Widget state dependencies.

Enhanced tracking:

- `InheritedWidget` usage (Theme, `MediaQuery`).
- Provider/Riverpod state management.
- Dynamic vs. static Material properties.

3 Phase 2: IR Design & Implementation (Week 7-8)

3.1 Milestone 4: Schema Design

Design FlatBuffers schema capturing:

- Widget hierarchy, properties, state dependencies.
- Material/Cupertino design data:

```
table MaterialTheme {
  primary_color: Color;
  accent_color: Color;
  text_theme: TextTheme;
  elevation_values: [float];
  border_radius_values: [float];
}

table CupertinoTheme {
  primary_color: Color;
  system_colors: [Color];
  text_styles: [TextStyle];
}

table Widget {
  type: string;
  properties: [Property];
  material_theme_refs: [ThemeRef];
  design_system: DesignSystem;
}
```

Store design tokens separately to enable theme switching, validation, and optimization.

3.2 Milestone 5: FlatBuffers Writer (Dart)

Convert parsed widget tree to `.flir` file, including:

- Material/Cupertino theme data as separate IR section.
- Links between widget properties and theme references.
- Design system metadata for validation.
- Constraint information for CSS layout generation.

4 Phase 3: Flutter-Identity HTML/CSS/JS Transpiler (Week 9-12)

4.1 Milestone 6: Flutter-Aware Widget Mapping System (Week 9)

4.1.1 Task 6.1: Core Widget Mapping

Create transformation specifications for 30 widgets:

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

Example mapping:

```
Widget: ElevatedButton
HTML: <button class=”flutter-elevated-button”>
CSS Strategy: Material tokens + state classes
Material Properties:
  - Uses primary color from theme
  - Elevation 2 by default, 8 on hover
  - Border radius from theme
  - Typography: button text style
Special Handling:
  - onPressed null → disabled state
  - Material ripple effect → CSS pseudo-elements
Limitations:
  - Ripple animation simplified
```

4.1.2 Task 6.2: Material Design Integration

Generate CSS for Material Design 3:

- Color system (all shades, opacity variants).
- Typography scale (13 text styles).
- Elevation system (24 levels with shadows).
- Shape system (border radius).
- Motion system (duration + easing curves).

Example output:

```
:root {
  --md-sys-color-primary: #6750A4;
  --md-sys-color-primary-container: #EADDFF;
  --md-sys-typescale-headline-large-size: 32px;
  --md-sys-typescale-headline-large-weight: 400;
  --md-sys-elevation-level2: 0 1px 2px rgba(0,0,0,0.3);
}
```

4.1.3 Task 6.3: Cupertino Design Integration

Generate iOS design tokens:

- System colors (e.g., `systemBlue`).
- SF Pro font stack.
- iOS-specific effects (blur, translucency).
- iOS spacing and sizing conventions.

4.2 Milestone 7: CSS Generation Engine (Week 10)

4.2.1 Task 7.1: Styling Strategy

Hybrid approach:

- Material/Cupertino tokens □ CSS custom properties.
- Common patterns □ utility classes with `flutter-` prefix.
- Custom widget values □ inline styles.
- Component-specific styles □ scoped classes.

Example HTML:

```
<button
  class=”flutter-elevated-button”
  style=”background: var(--md-sys-color-primary);”
  data-widget-id=”w_042”
>
```

4.2.2 Task 7.2: EdgeInsets Converter

Handle Flutter padding/margin variants:

- `EdgeInsets.all()`.
- `EdgeInsets.symmetric()`.
- `EdgeInsets.only()`.
- `EdgeInsets.fromLTRB()`.

Context-aware padding vs. margin application.

4.2.3 Task 7.3: BoxDecoration Converter

Convert to CSS:

- Color + gradient.

- Multiple box shadows.
- Border with gradient.
- Border radius (per-corner variants).
- `Shape.circle` → `border-radius: 50%`.

Material-specific:

- Elevation → box-shadow from tokens.
- Material color with opacity → `rgba()`.
- Surface tint overlays.

4.2.4 Task 7.4: TextStyle Converter

Map Flutter typography to CSS:

- `fontSize` → `font-size` (px).
- `fontWeight` → numeric weight (bold → `700`).
- `fontFamily` → web-safe fallbacks.
- `letterSpacing`, `height` (line-height).
- Text decorations and shadows.

Use Material typography tokens for `Theme.of(context).textTheme`.

4.2.5 Task 7.5: Theme System

Generate:

1. CSS custom properties for entire theme.
2. Dark mode variants using media queries.
3. Theme inheritance for nested themes.

Example:

```
:root {
  --flutter-theme-primary: #6750A4;
  --flutter-theme-on-primary: #FFFFFF;
}
@media (prefers-color-scheme: dark) {
  :root {
    --flutter-theme-primary: #D0BCFF;
    --flutter-theme-on-primary: #381E72;
  }
}
```

4.3 Milestone 8: Flutter.js Runtime (Week 11-12)

4.3.1 Task 8.1: Architecture Design

Runtime modules (<15KB minified + gzipped):

- `core.js`: `setState`, lifecycle, render scheduling.

- `events.js`: Flutter callback binding.
- `context.js`: `BuildContext` simulation.
- `navigation.js`: `Navigator` implementation.
- `theme.js`: `ThemeData` access.
- `material.js`: Material behaviors (e.g., ripples).

4.3.2 Task 8.2: State Management

Implement Flutter-like `setState`:

- Synchronous state update.
- Asynchronous render scheduling.
- Batch multiple `setState` calls.
- Widget lifecycle (`initState`, `dispose`, `didUpdateWidget`).

Example:

```
class FlutterWidget {
  setState(updater) {
    updater(this.state);
    FlutterJS.scheduleRender(this);
  }
}
```

4.3.3 Task 8.3: Event System

Map Flutter callbacks to DOM events:

- onPressed, onTap, onChanged.
- `GestureDetector` patterns (tap, double tap, long press).
- Flutter-like event objects (`localPosition`, `globalPosition`).

4.3.4 Task 8.4: BuildContext Simulation

Provide:

- `Theme.of(context)` □ theme CSS variables.
- `MediaQuery.of(context)` □ window dimensions, safe area.
- `Navigator.of(context)` □ navigation instance.

4.3.5 Task 8.5: Navigation System

Implement `Navigator`:

- push/pop with browser history.
- Named routes.
- Route transitions (optional for MVP).
- Browser back button handling.

4.3.6 Task 8.6: Material Behaviors (Optional)

Add:

- Ripple effect on button press.
- Ink splash animations.
- Scroll physics (Material overscroll glow).

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

5.1 Milestone 9: End-to-End Pipeline

Build CLI tool:

```
flutter-js build
  ├─ Parse Dart files (Phase 1)
  ├─ Generate .flir with design tokens (Phase 2)
  ├─ Transpile to HTML/CSS/JS with Material fidelity (Phase 3)
  └─ Output build folder
```

Test cases:

1. Hello World (static).
2. Counter app (stateful).
3. Material showcase (Tier 1 widgets).
4. Form with validation.
5. Multi-page app with navigation.

5.2 Milestone 10: Comparison Benchmarks

Metrics:

- Bundle size: Flutter Web (2.1MB Hello World) vs. Flutter.js (<50KB).
- Initial load (3G): Flutter Web (12s) vs. Flutter.js (<2s).
- Time to Interactive: Flutter Web (15s) vs. Flutter.js (<3s).
- Lighthouse SEO: Flutter Web (40) vs. Flutter.js (>90).
- Visual fidelity: >95%.

Visual comparison:

- Flutter Web output.
- Flutter.js output.
- Native mobile screenshot.

Success: Most users can't distinguish at a glance.

6 Phase 5: Iteration & Refinement (Week 15+)

Based on Phase 4:

- If fidelity <95%: Review Material token accuracy, fix CSS generation, enhance layout handling.
- If bundle size too large: Implement tree-shaking, optimize runtime, use FlatBuffers string tables.
- If performance slow: Profile transpiler, optimize CSS, consider Rust for hot paths.

7 Evaluation Checkpoints

- **After Phase 0 (Week 2)**: Can Flutter's visual identity be maintained? Decision: Go/No-Go.
- **After Phase 1 (Week 6)**: Can theme data be extracted? Success rate >80%?
- **After Phase 2 (Week 8)**: Does IR preserve design system information?
- **After Phase 3 (Week 12)**: Does output resemble Flutter? Visual regression tests pass?
- **After Phase 4 (Week 14)**: Is Flutter.js better? Achieve 50% smaller bundles, faster load/TTI, 95%+ fidelity, better SEO.

8 Risk Mitigation

- **Can't maintain Material fidelity**: Use official Material Design 3 tokens, visual regression testing, document minor differences.
- **CSS can't replicate layouts**: Focus on supported layouts, document incompatibilities, provide custom CSS escape hatches.
- **Runtime too large**: Modular runtime, focus on core features (state, events, navigation).
- **Development takes too long**: Ship Tier 1 widgets first, accept imperfect Tier 2/3, iterate based on feedback.

9 Success Definition (6-Month Goal)

9.1 Minimum Viable Product

- Supported: 30 widgets (15 pixel-perfect, 10 close, 5 best effort), Material Design 3, basic Cupertino, `setState`, navigation, theme system, form inputs.
- Bundle: <100KB, semantic HTML, WCAG AA, SEO-friendly.
- Fidelity: 95%+ for Material widgets, indistinguishable in common cases.

9.2 Target Use Cases

- Landing pages, marketing sites, content-heavy apps, forms, CRUD interfaces.

9.3 NOT for

- Canvas-heavy apps, complex custom painters, games, apps needing native APIs, perfect pixel-matching for complex layouts.

10 Timeline Summary

- Week 1-2: Manual proof with Material button.
- Week 3-6: Parser with design token extraction.
- Week 7-8: IR with Material/Cupertino theme storage.
- Week 9-10: Widget mapping + CSS generation.
- Week 11-12: Runtime with Flutter patterns.
- Week 13-14: Integration + benchmarking.
- Week 15-20: Refinement.
- Week 20: MVP launch.

11 Final Reality Check

Before starting:

1. Can you commit 15-20 hours/week for 6 months?
2. Are you comfortable with 95% fidelity?
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?

If all yes: Start Phase 0 with Material button proof of concept. If any no: Reconsider scope or timeline.

12 Key Differentiator

Flutter.js is not a Flutter Web replacement but a ”Flutter Design Systems for HTML” solution, prioritizing design fidelity over engine equivalence.