Project Documentation

JSON-Driven Layout Optimizer

Author: [Your Name]

Course: Non-procedural Programming

Year: 2025

# 1. Introduction

This project implements a system that parses, validates, and optimizes layout specifications defined in JSON format. The system outputs a textual summary of the optimized layout, and a rendered HTML representation styled with interchangeable CSS files. The project is implemented in Haskell using the cabal build system.

# 2. Objectives

- Parse input from JSON format.  
- Support multiple layout constraints and component types.  
- Optimize placement of layout elements into a 12-column grid.  
- Output human-readable summaries and HTML visualizations.  
- Provide multiple CSS styles for different visual themes.  
- Run easily via command-line with arguments.

# 3. System Architecture

## 3.1 Modules

• Parser.hs – Responsible for reading JSON input and converting it into typed Haskell structures (Config, Component, Constraint, InputDoc).  
• Layout.hs – Contains optimization logic that places components into rows and columns according to constraints.  
• HtmlExport.hs – Translates Layout into HTML markup with external CSS.  
• Main.hs – Orchestrates the workflow: parse → optimize → print summary → export HTML.

## 3.2 Data Flow

1. JSON input file → parsed by Parser.hs.  
2. Parsed InputDoc → processed by Layout.hs into an optimized layout.  
3. Layout is printed in textual form for debugging.  
4. HtmlExport.hs generates HTML output, styled with chosen CSS.

# 4. Input Specification

Example JSON input:

{  
 "config": { "gridCols": 12 },  
 "constraints": ["HeaderAtTop", "FooterAtBottom", "ImagesNeedTextNeighbor"],  
 "components": [  
 { "id": "hdr", "kind": "Header", "content": "Welcome to Our Page" },  
 { "id": "img1", "kind": "Image", "src": "https://example.com/image1.jpg" },  
 { "id": "txt1", "kind": "TextBlock", "content": "Our team is dedicated..." },  
 { "id": "ftr", "kind": "Footer", "content": "Contact us: info@example.com" }  
 ]  
}

# 5. Output Specification

Console summary example:

Row 1: hdr@1x12[h=140]  
Row 2: img1@1x6[h=200] | txt1@7x8[h=200]  
Row 3: ftr@1x12[h=120]

HTML output links to external CSS file. Available styles:  
- style-minimal.css  
- style-classic.css  
- style-modern.css

# 6. Styling

Three CSS themes are provided:  
• Minimal – clean, white, subtle borders.  
• Classic – structured with light gray background and serif typography.  
• Modern – dark theme with gradient background, accent colors, hover effects.

# 7. How to Run

Build the project:  
 cabal build  
  
Run with JSON input:  
 cabal run project -- layout\_with\_images.json  
  
With options:  
 cabal run project -- layout\_complex.json --out complex.html --css style-modern.css

# 8. Conclusion

The project successfully implements the main specification: JSON parsing, constraint handling, layout optimization, and HTML/CSS export. The system supports multiple visual styles and can be extended with new constraints or output formats.

# 9. Code Highlights

The project consists of four main Haskell modules. Below are the most important parts of the implementation:

• Parser.hs – Defines the data structures (`Config`, `Component`, `Constraint`, `InputDoc`) and implements JSON parsing using the Aeson library. It ensures that components and constraints are correctly deserialized from the input JSON.

• Layout.hs – Implements the backtracking algorithm for layout optimization. Key function: `optimizeLayout`, which generates candidate layouts, checks feasibility against constraints, and selects the best layout based on a scoring function. Helper functions like `feasibleRow` enforce rules such as ensuring images are next to text if required.

• HtmlExport.hs – Responsible for rendering the optimized layout into HTML. Function `renderHtmlWithCss` constructs a complete HTML document and links an external CSS stylesheet. Each `Placement` is translated into a `<div>` cell, with different handling for headers, footers, text blocks, and images.

• Main.hs – Entry point of the application. It handles command-line arguments, loads the JSON file, runs the optimizer, prints a summary of the layout, and optionally writes the HTML file. It supports options `--out` for HTML output and `--css` for selecting the stylesheet.

Together, these modules implement the full pipeline: JSON input → optimized layout → HTML export. The code is organized to separate concerns clearly: parsing, optimization, rendering, and orchestration.