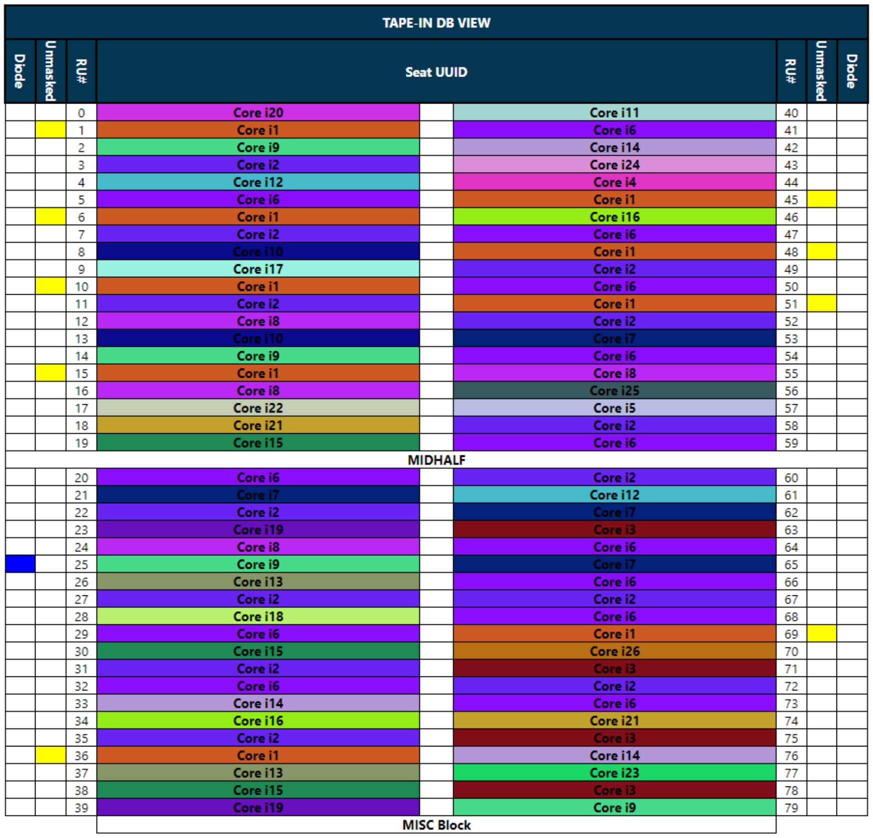
Functional and Technical Documentation

Overview: The project's objective was to create a JavaScript Application in a popular framework (React/Vue/Angular) that takes a JSON dataset containing product names and their repetition count and generates a floor plan with an interactive RU Table while meeting predetermined guidelines.

React was selected for this project because of its advantages, which include its state management capabilities, component-based architecture, and effective rendering. Below are some of the benefits of using React:

- Using React, you can create user interfaces in JavaScript—a widely used tool for front-end development.
- Being a versatile, declarative, and practical framework, developers can quickly develop detailed views and single-page applications.
- Virtual DOMs in React enable quick rendering and memory efficiency.
- In contrast to many other frameworks, react components are easier to reuse and analyze, making code maintenance easier.



TAPE-IN DB VIEW								
Diode	Unmasked	RU#	Seat UUID			RU#	Unmasked	Diode
		0	Core i20		Core i11	40		
	🟨	1	Core i1		Core i6	41		
		2	Core i9		Core i14	42		
		3	Core i2		Core i24	43		
		4	Core i12		Core i4	44		
		5	Core i6		Core i1	45	🟨	
	🟨	6	Core i1		Core i16	46		
		7	Core i2		Core i6	47		
		8	Core i10		Core i1	48	🟨	
		9	Core i17		Core i2	49		
	🟨	10	Core i1		Core i6	50		
		11	Core i2		Core i1	51	🟨	
		12	Core i8		Core i2	52		
		13	Core i10		Core i7	53		
		14	Core i9		Core i6	54		
	🟨	15	Core i1		Core i8	55		
		16	Core i8		Core i25	56		
		17	Core i22		Core i5	57		
		18	Core i21		Core i2	58		
		19	Core i15		Core i6	59		
MIDHALF								
		20	Core i6		Core i2	60		
		21	Core i7		Core i12	61		
		22	Core i2		Core i7	62		
		23	Core i19		Core i3	63		
		24	Core i8		Core i6	64		
🟦		25	Core i9		Core i7	65		
		26	Core i13		Core i6	66		
		27	Core i2		Core i2	67		
		28	Core i18		Core i6	68		
		29	Core i6		Core i1	69	🟨	
		30	Core i15		Core i26	70		
		31	Core i2		Core i3	71		
		32	Core i6		Core i2	72		
		33	Core i14		Core i6	73		
		34	Core i16		Core i21	74		
		35	Core i2		Core i3	75		
	🟨	36	Core i1		Core i14	76		
		37	Core i13		Core i23	77		
		38	Core i15		Core i3	78		
		39	Core i19		Core i9	79		
			MISC Block					

Fig - 1. Floor Plan(RU Table)

Implementation Details for above Floor Plan:

- **Data Analysis:** The JSON dataset, which contains product names and their frequency counts, was thoroughly analyzed and understood. This dataset was integral to the functionality of the RU Table. Following analysis, the dataset was added to a variable called **dataset** in the React project by importing it. This step was crucial for dynamically rendering the products in the RU Table based on the data provided, ensuring that each product was accurately represented in accordance with its frequency and characteristics as outlined in the dataset.

- **Data Optimization and Algorithm Development:** In the implementation, a 2D array structure was created, consisting of four internal arrays to represent the grids. An algorithm was then developed to dynamically allocate the cores across these arrays. This allocation continued until the repetition count of each product reached zero, adhering to specific constraints.

  Notably, Core i4/i5 products were restricted to placement only in grids 1 and 2. Additionally, the algorithm ensured that no two identical products were adjacent, allowing for a product's placement in alternating positions, but not directly next to an identical product.
  Developed a function named **distributeProducts()** to allocate products within the array structure, representing the grids. This function utilized **getRandomPosition()** to determine the index positions **(i and j)** for product placement, considering constraints like allocating Core i4 or Core i5 products exclusively to the first two grids (i values 0 and 1). For other products, allocation spanned across all four grids (i values 0 to 3) and 20 positions (j values 0 to 20) in each grid. Additionally, **isPositionValid()** was employed to ensure valid positioning, adhering to the constraint that no identical products were adjacent and allocated all the data into a variable called **floorValue.**

- **Grid Layout:** Implemented an 80 RU floor plan divided into four grids using React components. This was achieved by defining a state with **useState** in the React component. The state, named **allGridData**, was initialized with an array containing grid data. This initialization involved calling the **getGridData()** function with varied starting indices **(0, 40, 20, 60)** and specific **floorValue** parameters for each grid. This approach sets up the initial state for grid data, enabling each grid to have its own unique data and facilitating dynamic and interactive grid management within the application.

- **Products Render:** A dynamic table representing a grid layout displays product information. The table, structured with headers and body, features interactive columns like **Diode** and **Unmasked**, where user clicks trigger color changes, managed by **toggleDiode**() and **toggleUnmasked()** functions. The **toggleUnmasked()** particularly highlights matching product cells in yellow. The **Diode** column cells turn blue on click. The table includes a **MIDHALF** row for visual segmentation after 20 RUs and concludes with a **MISC Block** section. The grid layout is generated from the **allGridData** state, with dynamic color allocation to the **Seat UUID** column using **getRandomColor**() where each product gets a random color on loading/reloading.

Note: I chose not to include grids and I/O's columns, as these were optional elements. This decision was made to streamline the layout and focus on the essential features of the table, ensuring clarity and functionality in the design.