Senior Frontend Developer Interview Questions

> ℹ️ All resources (videos, references) for these questions can be found in the following link:
> [Frontend Developer Interview]

General Instructions

Tech Stack

- Logic: **TypeScript + React** (plain JavaScript acceptable only where explicitly noted)
- Styling: **styled-components**, **Sass**, or **basic CSS**

Submission

- Create a **Git repository** with your solution.
  - Either host it on **GitHub** and share the link, or
  - Zip the repo and email it to us.

Evaluation Criteria (applies to both questions)

- Ability to extract and translate requirements from provided videos/materials.
- Code structure, readability, and maintainability.
- Correctness and completeness of functionality.
- Use of React + TypeScript best practices.
- Styling clarity and organization.
- Handling of edge cases and overall problem-solving approach.

Question 1: Build a Polygon Visualizer (No Frameworks)

Your task is to implement the following function and visualize it working. Refer to the video *"Question 1 - Sample Demo Idea.mov"* for inspiration.

```
interface Point { x: number; y: number; }

/**
 * Returns the closest point to `pos` that lies inside or on the boundary
 * of the polygon `poly` (non-self-intersecting; may be concave).
 * Time complexity should be O(n) in number of vertices.
 */
function closestPointInPolygon(poly: Point[], pos: Point): Point
```

Requirements

- Create a small **HTML page** (no frameworks/libraries).
- Use the `<canvas>` element to draw and interact.
- Features:
  - Draw a polygon (hardcode a few examples; bonus: allow click-to-add vertices).
  - Show a draggable test point `pos` .
  - Render the returned closest point and a segment from `pos` to that point.
- Use only basic **mouse events** (`mousedown` , `mousemove` , `mouseup`) and the **Canvas 2D API**.

Constraints

- JavaScript or TypeScript only.
- No third-party libraries.
- Polygon is simple (not self-intersecting). It may be convex or concave.
- Points on edges/vertices count as “inside.”
- Be numerically stable (use a small epsilon for comparisons).

What to Implement

1. **Point-in-polygon** test (winding number or ray casting).
2. **Distance-to-segment** helper with clamped projection.
3. **closestPointInPolygon** logic:
   - If `pos` is inside/on-edge → return `pos` .
   - Otherwise → check each polygon edge and return the closest point.

Edge Cases

- `pos` exactly on an edge or vertex.
- Multiple equidistant edges (any correct point is acceptable).
- Degenerate edges (duplicate vertices).
- Polygons with holes are **out of scope**.

Suggested Tests

- Square example from the prompt.
- Concave “L” shape where closest point lies on the inner elbow.
- `pos` inside polygon → identical point returned.
- `pos` near a vertex → vertex returned.

- `pos` near an extended edge → nearest endpoint.

Question 2: Recreate the PortTemplate Component (React + TypeScript)

Watch the video *“Question 2 - Design Conversation”* to understand the requirements. Then implement the `PortTemplate` component and showcase it in a live demo page.

Requirements

- Implement the **PortTemplate** React component.
- Extract requirements and behaviors from the video.
- Build a **demo page** to showcase the component with interactive functionality.