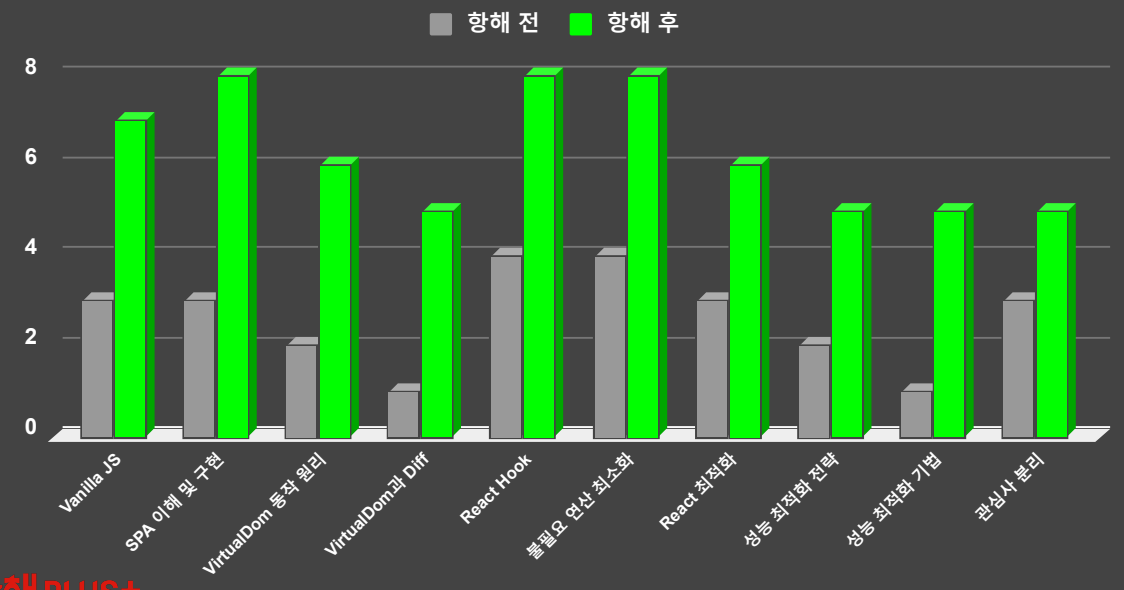
항해 PLUS+	평가 항목 | 챕터1 : JS&React 딥다이브_장동진	점수 (● : 항해 전 / ● : 항해 후)									
		1	2	3	4	5	6	7	8	9	10
1	프레임워크 없이 JavaScript를 사용해 DOM 조작, 이벤트 처리, 상태 관리를 구현할 수 있다.			●				●			
2	Single Page Application(SPA)의 기본 원리(라우팅, 상태 관리 등)를 이해하고 구현할 수 있다.			●					●		
3	React가 해결하려는 문제와 VirtualDOM의 동작 원리를 이해하고 설명할 수 있다.		●				●				
4	VirtualDOM과 Diff 알고리즘의 원리를 이해하고, 이를 활용한 최적화 방법을 설명할 수 있다.	●				●					
5	React의 주요 Hook(useState, useEffect, useContext 등)의 동작 원리를 이해하고, Custom Hook을 설계하여 관심사를 분리할 수				●				●		
6	useMemo와 useCallback을 활용해 불필요한 연산과 재생성을 최소화할 수 있다.				●				●		
7	React 원리를 바탕으로 불필요한 렌더링과 이벤트 처리를 최적화 할 수 있다.			●			●				
8	성능 측정 도구(Lighthouse, React DevTools 등)를 사용해 병목 구간을 파악하고 최적화 전략을 설계할 수 있다.		●			●					
9	번들 크기, 코드 스플리팅, 캐싱 등 프론트엔드 애플리케이션의 성능 최적화 기법을 적용할 수 있다.	●				●					
10	React의 Context API를 활용하여 전역 상태를 관리하고 관심사를 분리할 수 있다.			●		●					
	총합	●26 / ●63									



항해 PLUS+

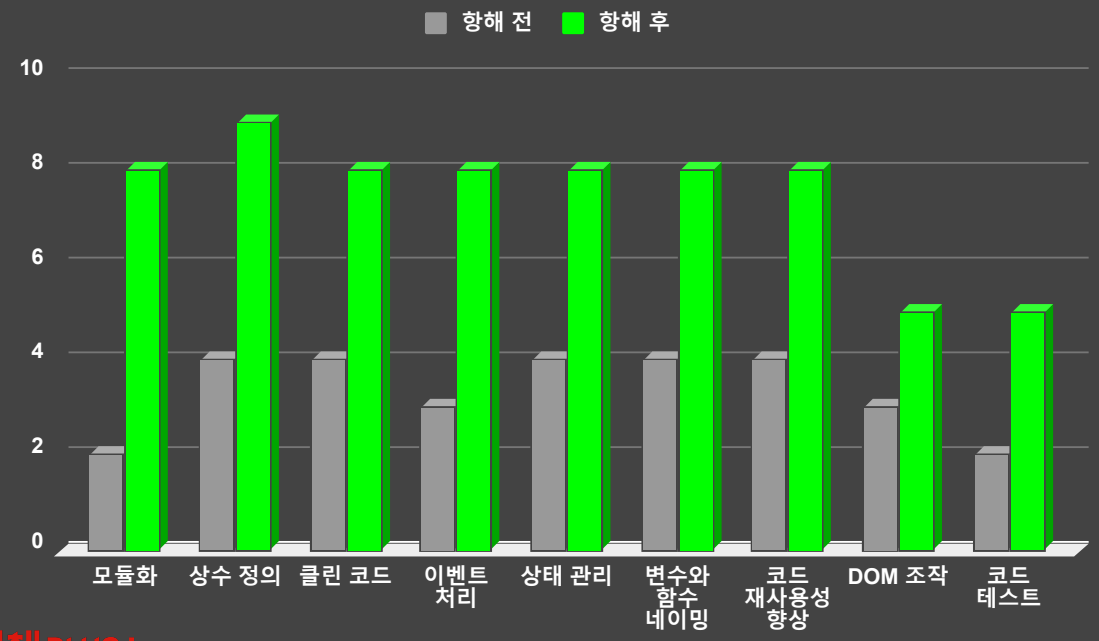
총점 및 성장률			
	항해 전	항해 후	성장률
총점	26	63	59%

가장 이해도가 낮았던 영역		
	키워드	항해 전 점수
1	VirtualDom과 Diff	1
2	VirtualDom 동작 원리	2
3	SPA 이해 및 구현	3

가장 큰 성장률을 보인 영역		
	키워드	성장률
1	VirtualDom과 Diff	80%
2	VirtualDom 동작 원리	67%
3	SPA 이해 및 구현	63%

항해 PLUS+

항해 PLUS+	평가 항목 | 챕터2 : 클린코드_장동진	점수 (● : 항해 전 / ● : 항해 후)									
		1	2	3	4	5	6	7	8	9	10
1	코드를 기능별로 적절하게 모듈화할 수 있는가?		●						●		
2	코드에서 매직 넘버와 반복되는 문자열을 상수로 정의할 수 있는가?				●					●	
3	코드를 읽기 쉽고 이해하기 쉽게 작성할 수 있는가?				●				●		
4	이벤트 처리 로직을 효율적으로 구성할 수 있는가?			●					●		
5	상태 관리 방식을 적절하게 개선할 수 있는가?				●				●		
6	변수와 함수의 이름이 그 용도와 기능을 명확히 나타내도록 표현할 수 있는가?				●				●		
7	중복된 코드를 제거하고 재사용성을 높일 수 있는가?				●				●		
8	DOM 조작을 필요한 최소한으로 줄일 수 있는가?			●		●					
9	코드의 테스트 가능성이 향상되었는가?		●			●					
10	예상 가능한 에러 상황에 대한 처리가 적절하게 이루어졌는가?		●			●					
	총합	●32 / ●72									



항해 PLUS+

총점 및 성장률

	항해 전	항해 후	성장률
총점	32	72	56%

가장 이해도가 낮았던 영역

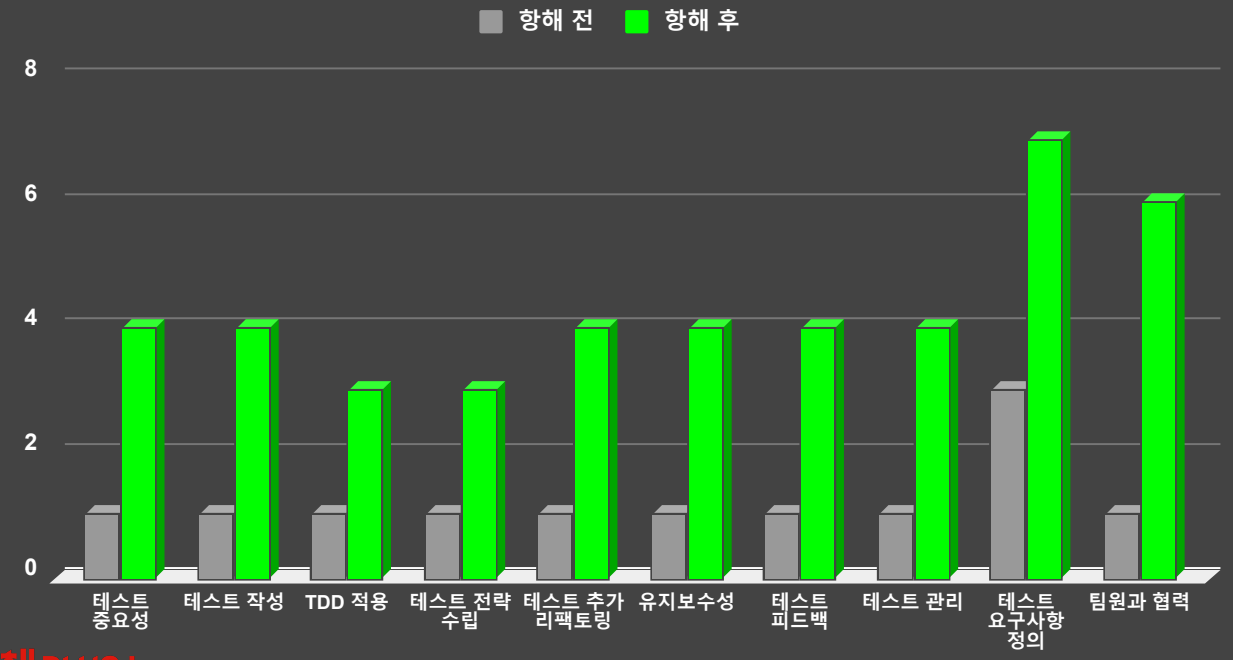
	키워드	항해 전 점수
1	모듈화	2
2	이벤트 처리	3
3	상수 정의	4

가장 큰 성장률을 보인 영역

	키워드	성장률
1	모듈화	75%
2	이벤트 처리	63%
3	상수 정의	56%

항해 PLUS+

항해 PLUS+	평가 항목 | 챕터3 : 프론트엔드 테스트코드_장동진	점수 (● : 항해 전 / ● : 항해 후)									
		1	2	3	4	5	6	7	8	9	10
1	소프트웨어 개발 과정에서 테스트의 중요성과 필요성을 이해하고 설명할 수 있다.	●			●						
2	단위 테스트 및 통합 테스트를 작성하고 실행할 수 있다.	●			●						
3	TDD의 기본 원칙과 프로세스를 이해하고, 실제 개발 과정에서 이를 적용할 수 있다.	●		●							
4	프론트엔드 특성에 맞는 현실적이고 효과적인 테스트 전략을 수립할 수 있다.	●		●							
5	기존의 레거시 코드에 점진적으로 테스트를 추가하고, 안정적인 리팩토링을 수행할 수 있다.	●			●						
6	테스트를 고려한 구조적이고 유지보수성이 높은 코드를 작성할 수 있다.	●			●						
7	테스트 피드백을 통해 기능의 안정성을 높이고, 이를 기반으로 견고한 코드를 작성할 수 있다.	●			●						
8	Jest, React Testing Library 등 프론트엔드 테스트 도구를 사용하여 효율적으로 테스트를 작성하고 관리할 수 있다.	●			●						
9	실제 사용자 시나리오를 바탕으로 테스트 요구사항을 정의하고, 이를 코드로 구현할 수 있다.			●				●			
10	테스트 작성 및 실행을 팀원들과 협력하여 진행하고, 테스트 결과를 바탕으로 기술적 의견을 교환할 수 있다.	●					●				
	총합	● 12 / ● 43									



총점 및 성장률

	항해 전	항해 후	성장률
총점	12	43	72%

가장 이해도가 낮았던 영역

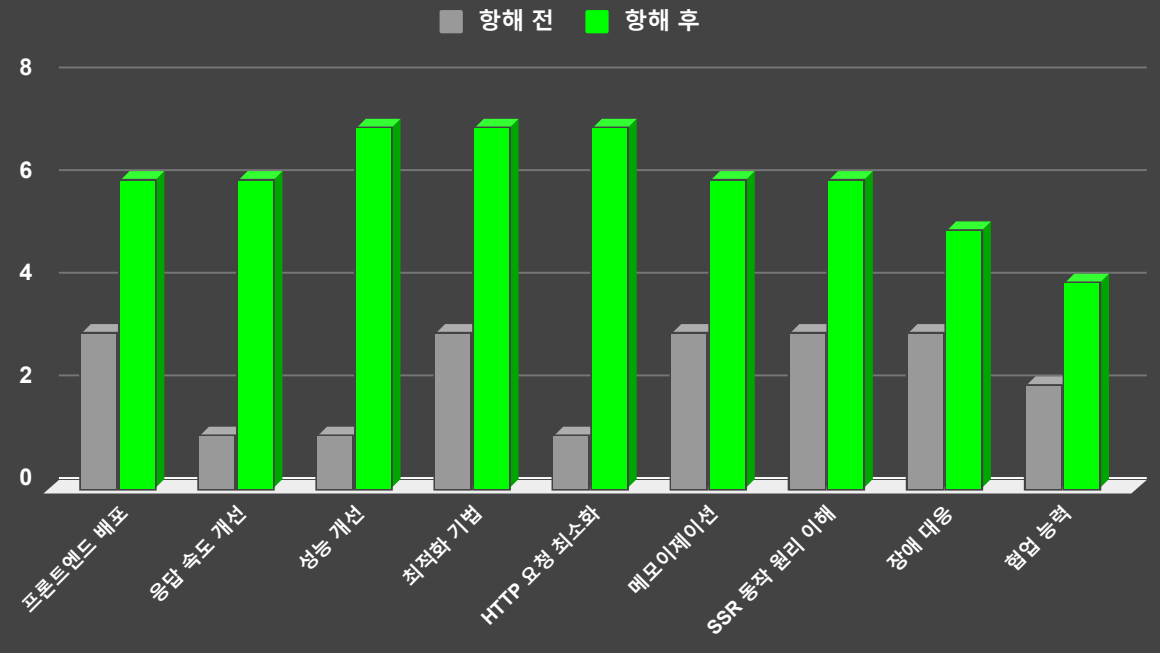
	키워드	항해 전 점수
1	테스트 중요성	1
2	테스트 작성	1
3	테스트 추가 리팩토링	1

가장 큰 성장률을 보인 영역

	키워드	성장률
1	테스트 중요성	75%
2	테스트 작성	75%
3	테스트 추가 리팩토링	75%

항해 PLUS+

항해 PLUS+	평가 항목 | 챕터4 : 성능최적화_장동진	점수 (● : 항해 전 / ● : 항해 후)									
		1	2	3	4	5	6	7	8	9	10
1	프론트엔드 프로젝트의 빌드 및 배포 프로세스를 이해하고, AWS와 같은 클라우드 서비스를 활용하여 배포할 수 있다.			●			●				
2	CDN의 작동 원리를 이해하고, 적절한 캐싱 전략을 수립하여 글로벌 응답 속도를 개선할 수 있다.	●					●				
3	Lighthouse, Chrome DevTools 등 성능 측정 도구를 사용하여 애플리케이션 성능 병목을 파악하고 개선할 수 있다.	●						●			
4	JavaScript와 CSS의 최적화 기법을 이해하고 적용하며, 번들 크기와 코드 로딩 성능을 개선할 수 있다.			●				●			
5	리소스 로딩 성능을 개선하고, HTTP 요청 최소화를 통해 네트워크 성능을 최적화할 수 있다.	●						●			
6	React에서 불필요한 렌더링을 방지하고, 메모이제이션(useMemo, useCallback)을 활용하여 성능을 최적화할 수 있다.			●			●				
7	SSR의 동작 원리를 이해하고, 이를 구현하여 초기 로딩 시간을 개선할 수 있다.			●			●				
8	HTTP, DNS, TCP/IP 등 네트워크 기본 개념을 이해하고, 장애 발생 시 원인을 파악하고 대응할 수 있다.			●		●					
9	애플리케이션의 성능 병목을 진단하고 해결하며, 백엔드 및 DevOps 팀과 원활히 협력할 수 있다.		●		●						
10	코드를 작성할 때 성능 최적화를 고려하고, 이를 개발 과정에서 지속적으로 반영할 수 있다.		●			●					
	총합	● 12 / ● 39									



총점 및 성장률			
	항해 전	항해 후	성장률
총점	12	39	69%

가장 이해도가 낮았던 영역		
	키워드	항해 전 점수
1	성능 개선	1
2	HTTP 요청 최소화	1
3	응답 속도 개선	1

가장 큰 성장률을 보인 영역		
	키워드	성장률
1	성능 개선	86%
2	HTTP 요청 최소화	86%
3	응답 속도 개선	83%

항해 PLUS+

항해 PLUS+