INLÄMNING 1
ANDREAS LÖNNHOLM

Ramverk, biliotek och React

Fråga 1

Vad är ett frontend-bibliotek

och vad skiljer det från ett frontend-ramverk?

Ett bibliotek kan vara en enskild metod eller funktion som ger dig lösningen på ditt problem. Ramverk å andra sidan är en färdig helhetslösning av grundläggande funktioner du kan tänkas behövas, som du sen kan utöka med mer funktionalitet genom att lägga till enskilda bibliotek.

Fråga 1.1

Vad är React.js för något och varför bör man använda det?

ReactJS är ett JavaScript-bibliotek för att skapa gränssnitt för webben.

Vilket du bör använda för att kunna återanvända UI-komponenter, snabba upp prestandan och för att kunna ändra information på din hemsida utan att behöva ladda om hela sidan. Andra alternativ finns som ger liknande lösning, men React är den allra vanligaste och det är därför lättare att få hjälp av dess community.

- **Körs på klienten som en Single Page Application (SPA)**
- **Används för att bygga snabba, responsiva fullstack-appar gentemot Server/API**

Fråga 1.2

Inom React arbetar vi med komponenter, vad är en komponent?

En komponent kan vi se som en del av en hemsida. Exempelvis en meny, footer eller huvudinnehållet. En enskild del, en JavaScript-fil, som visar ett innehåll på något vis, antingen genom att få data från användaren eller genom att statiskt visa det.

Fråga 1.3
Inom React har komponenter props,
vad är en prop och vad används den till?

Props är en egenskap (argumentsbehållare) på en komponent, där du kan överföra information mellan olika komponenter. Information förs bara vidare mellan parent och child och kan inte hoppar flera steg i ett.

Fråga 1.4
Inom React kan komponenter antingen vara klassbaserade eller funktionsbaserade, vad är skillnaden mellan dessa tillvägagångssätt att upprätta komponenter på och är något av sätten att föredra?

En funktionsbaserad komponent är en vanlig funktion som tar emot props och skickar tillbaka ett resultat. Den får sin data via props och renderar till JSX.

En klassbaserad komponent har flera funktioner och kan hantera state, hook och andra ofta använda redskap.

Klassbaserade är föråldrade och används inte så ofta längre, då de är mer besvärliga att arbeta med. Många av de funktioner som förr var exklusiva för det klassbaserade läget är inte längre det.

Fråga 1.5
Inom React arbetar vi med state,
vad är state för något och vad används det till?

State är en sorts ”minne” som hanterar en komponents information (data). Text, siffror eller boolean - precis som övriga variabler.
State har ett basvärde och en manipulator (setter). När dess värde ändras renderas komponenten om och de nya värdena visas upp.

Fråga 1.6
Inom React arbetar vi ofta med hooks,
vad är en hook för något, vad kan den användas till,
var kan den appliceras och hur skapar man sin egen hook?

En hook är en sorts funktion som låter dig påverka state
och lifeycle-funktionerna även inom funktionella komponenter.
Dessa kan återanvändas om och om igen. De vanligaste är useState, useEffect och useContext.

För att använda en hook behöver du först importera den och sen kan den användas som en vanlig funktion. Exempelvis useEffect(() => …

Den kan se till att ladda API-värden innan sidan laddats eller, uppdatera sidan när arrayers värden ändras eller spara/ladda lagrad information från localstorage.

Fråga 1.7

Beskriv kortfattat vad huvudsyftet med hooken useEffect() är och hur man tolkar dess olika parametrar?

UseEffects syfte är att reagera när värden lagrade i useState() ändras och då uppdatera DOM-vyn för användaren. Exempel är om du har en räknare, så vill du att siffran som visas i webbläsaren uppdateras varje gång den ändras av koden.

useEffect(() => { }, [];
Om du har de sista brackets tomma (alltså []) körs innehållet i funktionen bara en gång, alltså vid första uppstart. Om du i stället inuti den anger en state, kommer innehållet att köras varje gång state ändras.

Fråga 1.8

I React så har våra komponenter så kallade livscykler, vad är en livscykel i denna kontext och varför är de viktiga att känna till och att arbeta med?

En komponent göra olika saker vid olike tillfällen i sitt ”liv”.
En sak när den startar, en annan sak vid omstart eller avslut.
.
Detta är olika livscykler och vad som sker vid respektive av dessa tillfällen kan definieras direkt i React. Till exempel kanske du bara vill att din timer-applikation startar vid första tillfället applikationen har laddats klart och lägger därför in den i componentDidMount().

Att förstå i vilken livscykel respektive komponent befinner sig i är viktigt för att förstå vad/varför delar av DOM:en (det du ser utåt i webbläsaren) uppdateras eller inte. Allt för att ha kontroll över din webbapplikation.

Fråga 1.9

I React så talar vi ibland om en Single Source of Truth när vi diskuterar arv, vad innebär detta begrepp?

Det innebär att försöka hålla så att varje state går uppifrån och ner,
så att props skickas neråt till child och inte tvärtom. Detta för att se till att alla child-state är synkroniserade med toppen och inte dubbletter.

Fråga 1.10

I React så arbetar vi ibland med något som kallas för global state management, vad är ett globalt state och vad kan det användas till?

Global state management är ett sätt att skicka data med information genom komponents-trädet utan att behöva stega igenom det med props via varje parent-child. Ett sätt att skicka data direkt dit man vill, alltså.

Många state kan hanteras lokalt, så som ett textfält inuti en komponent, men vissa saker vill vi kunna ha koll på, på en global nivå, och vilken användare som är inloggad kan vara ett exempel på detta.

Node.js

Fråga 2

Vad är Node.js och varför bör man överväga att använda det för sin backend-arkitektur?

NodeJS ger oss möjligheten att exekvera JavaScript-kod på serversidan, en sorts backend-server. Med den är det lätt att snabbt få skalbara webbapplikationer. NodeJS är minneseffektiv och kan köras på alla större operativsystem.
Förenklar underhåll och inlärning då både frontend och backend blir JavaScript. Är optimalt för REST-API:er, chatt- och CRUD-appar.

Fråga 2.1

Vad är en modul inom Node.js?

En är en funktion i form av en JavaScript-fil som kan ladda andra moduler i sin tur och som kan återanvändas om och om igen.

- **Node Core Modules (Path, File System, http, et cetera…)**
- **Tredjepartsmoduler via NPM**
- **Egenskapade funktioner**

Fråga 2.2

Vad är en npm inom Node.js?

Node pakethanterare, vilket gör att du kan installera tillägg till React, Vue, Angular och allt möjligt bara genom korta, enkla kommandon.
Går att jämföra med Ubuntu Linux APT-funktion. Alla paket listas i filen package.json under ’dependencies’.

Fråga 2.3
När bör Node.js användas och inom vilka kontexter är det ett verktyg att föredra för ökad prestanda hos en applikation?

När du vill ha ett system som är lätt att rekrytera arbetare för underhåll och snabb uppsättning är NodeJs att föredra, då det har JavaScript både på klient- och serversida, vilket förenklar uppstartsprocessen, underhållet och förbättrar prestandan på applikationen.

NodeJS används primärt för följande:

- **vid realtids-applikationer för att få bra prestanda vid tillfällen som chatt-tjänster (Facebook, Trello) eller liknande som innebär många korta meddelanden som visas upp för många användare.**
- **Ljud- eller video-streamingsidor, som skickas många små delar av en film/ljud i stället för allt på en gång.**

Fråga 2.4
När bör Node.js inte användas och inom vilka kontexter är det ett verktyg att undvika för att minska risken för prestanda-problem hos en applikation?

Undvik NodeJS om backend behöver ha tunga,
CPU-krävande beräkningar med många stora förfrågningar som sker ofta.
Detta då NodeJS bara stödjer en-trådig hantering hos processorn,
vilket lätt fyller dess stack.

Hjälpverktyg för test och paketering

Fråga 3
Beskriv kortfattat verktyget Gulp och dess användningsområde

- **Det är ett verktyg för att automatisera och optimera saker som behöver göras men som inte alltid är det roligaste att göra om och om manuellt => Kompilera SASS till CSS eller optimera bilder.**
- **Open Source JavaScript-verktyg och taskrunner**
- **Framtaget för att hantera tidskrävande och repetitiva tasks**

- **Tillhandahåller hundratals olika plugin.**

Vanliga tasks i Gulp:

- **Minifiering av script- och style-filer**
- **Concatenering**
- **Cache busting**
- **Testning, lintning och optimering**
- **Upprättande av utvecklingsserver**

Fråga 3.1
Beskriv kortfattat verktyget Webpack och dess användningsområde

Det är ett verktyg för att paketera (slå ihop) och göra om TypeScript, Sass, Hbs och annan kod för att sen göra om dem till fil-standarder som webbläsaren förstår (js, css, jpg, png).

- **En bundler för JS-moduler**
- **Genererar statistiska assets från antingen filer vi skrivit själva eller paket installerade via NPM**
- **Bygger statiska filer av moduler i JS som inte kan direkt tolkas av browsern**
- **Kan utöka sin funktionalitet via plugins och loaders för att hantera style direkt i modulen**

Fråga 3.2
Beskriv kortfattat verktyget JSHint och dess användningsområde

Är ett verktyg för att hålla koll på kodkvalitet, hitta fel och visa på vilket sätt det kan rättas till.

Både ESLint och JSHint är två olika verktyg för samma sak. Den ger oss felmeddelanden i konsolen för icke-optimal kod, så som tomma funktioner, ej använda variabler och andra fel.

Fråga 3.3

Beskriv kortfattat verktyget ESLint och dess användningsområde

Ett verktyg som analyserar och finner buggar i din kod, ser till att du håller samma standard på koden över hela projektet. Om en variabel eller komponent importeras som inte används, kommer den att meddela dig om detta.

Båda verktygen i fråga 3.2 och 3.3 delar mycket funktionsmässigt.
ESLint är dock det populärare alternativet.