Webové aplikace

1. Popište, k čemu slouží uvedené technologie
   1. HTML, CSS, JavaScript

HTML (HyperText Markup Language) je značkovací jazyk určený pro definici obsahu webových aplikací. Pomocí něj se definují prvky, se kterými bude uživatel interagovat a aplikaci ovládat.

CSS (Cascading Style Sheets) je jazyk používaný pro popis vzhledu a formátování dokumentů napsaných v značkovacím jazyce, nejčastěji HTML.

JavaScript je vysokoúrovňový interpretovaný programovací jazyk. Byť se s sním můžete setkat i jinde, nejčastěji je používán pro implementaci logiky klientské části webových aplikací.

* 1. PHP, Node.js, ASP .NET

PHP je opět vysokoúrovňový interpretovaný programovací jazyk, v tomto případě se ale používá pro implementaci serverové části aplikace. K tomuto účelu se jedná o nejčastěji používaný jazyk. Skripty napsané v PHP se spouštějí na webových serverech, jako je například Apache.

Node.js je prostředí pro spouštění JavaScriptového kódu mimo prohlížeč. V kontextu webových aplikací se ve spojení s JavaScriptem jedná o alternativu k PHP a Apache.

ASP .NET je framework pro tvorbu webových aplikací a služeb nejčastěji pomocí jazyka C#. Opět se jedná o další možnou alternativu při volbě nástroje pro tvorbu webových aplikací.

* 1. Frameworky pro tvorbu webových aplikací

Framework je sada nástrojů usnadňující tvorbu aplikací. Frameworků existuje velmi velké množství, přičemž každý framework je určený pro daný typ aplikace – existují frameworky pro tvorbu desktopových aplikací (například WPF), her (Unity) ale i pro tvorbu webových aplikací. K těm se řadí například ASP .NET (primárně C#), Express.js (JavaScript), Spring Boot (Java), Laravel (PHP)

1. Princip fungování webové aplikace
   1. Klientská a serverová část.

Klientská část aplikace se stará o interakci s uživatelem. Jejím hlavním účelem je poskytovat uživatelské rozhraní na základě kterého bude uživatel moci aplikaci ovládat a zobrazovat potřebná data. Klientská část aplikace může být realizována libovolným způsobem – může se jednat o běžnou webovou aplikaci přístupnou přes webový prohlížeč, ale i o aplikaci přímo nainstalovanou v počítači či mobilu.

Serverová část aplikace slouží ke zpracování klientských žádostí. V naprosté většině případů zároveň obstarává autentizaci, autorizaci a správu dat. K serverové části webových aplikací se váží technologie jako je HTTP sever (například Apache), databáze (například MySQL), použitý programovací jazyk (například PHP)

* 1. Komunikace mezi klientem a serverem. – komunikace mezi klientem a serverem funguje na základě posílání a zpracovávání žádostí a odpovědí. Klient nejprve sestaví žádost, která bude odeslána na server. Musí specifikovat URL (identifikátor prováděné operace), nastavit metodu (GET, POST, PUT, DELETE), nastavit záhlaví-headers (zde se dávají dodatečné informace například o typu zasílaného obsahu – atribut content-type), nastavit tělo (to je možné u metody POST a PUT – tělo není součástí URL – díky tomu může být posílaných dat více a data jsou zabezpečenější). Po odeslání je žádost na serveru zpracována a poté server na základě jejího zpracování pošle odpověď klientovi. U odpovědi se specifikují stavové kódy, které říkají, zda byla žádost úspěšně vyřízena.
  2. MVC – návrhový vzor, velmi často používaný při tvorbě webových aplikací. Aplikace je rozdělena do tří částí – Model, View, Controller. Model se stará o práci s daty, která nejčastěji ukládá do databáze. View prezentuje požadovaná data uživateli. Controller propojuje Model s View – na základě požadavků uživatele (klienta) říká, co má provést Model a jaké View bude zobrazeno.

1. REST API
   1. Popis rozhraní

* REST (Representational state transfer) API je architektura webových aplikací umožňující komunikaci mezi klientem a serverem pomocí http metod. Klient může data ze serveru načítat, posílat, aktualizovat a mazat. Specifikace toho, co klient požaduje se provádí pomocí URL.
  1. Princip tvorby aplikace disponující REST API
* REST API je založeno na několika základních principech, které je dobré při tvorbě aplikace dodržet.
* 1. Klientská a serverová část jsou na sobě nezávislé – Aplikace na serveru akorát provádí specifikované operace a nezáleží jí, jakým způsobem je realizována klientská aplikace. Díky tomu může existovat libovolné množství klientských verzí aplikace (webová, desktopová, mobilní …), které komunikují s jednou serverovou.
* 2. Bezstavovost – Server neukládá žádný stav mezi žádostmi klienta => každá žádost musí obsahovat veškerá data potřebná k jejímu zpracování a stav aplikace je dán pouze touto žádostí, a ne stavem předchozím. Aplikuje se princip HATEOAS (Hypermedia as the engine of application state), kde jsou v odpovědi na žádost zahrnuty odkazy na další data či akce, které se k předchozí žádosti vážou.
* 3. Jednotné rozhraní – každý zdroj na serveru má mít pouze jeden URL. Díky tomu je zaručeno, že dotazy na téže data vždy vypadají stejně, nehledě na to, odkud přicházejí.
* 4. Cacheovatelnost – pokud je to možné, zasílané zdroje by měly být uložitelné do mezipaměti z důvodu navýšení výkonu při opakování stejných žádostí. Odpovědi zároveň musí obsahovat informace o tom, zda je jejich ukládání povoleno či ne.
* 5. Aplikace by měla být rozdělena do vrstev, kde každá vrstva se stará o specifické operace. Příkladem rozdělení aplikace do vrstev je návrhový vzor MVC.

1. Git a GitHub – popis nástrojů a jejich použití při vývoji webových aplikací

Git je verzovací software – software pomocí kterého je možné spravovat verze projektu na kterém pracujete. V rámci Gitu se vytvoří takzvaný repozitář, který obsahuje jednak všechny soubory aktuální verze, ale zároveň i informace o verzích ostatních. Mezi verzemi je následně možné přepínat a dále je rozšiřovat. Ukládání verzí se provádí pomocí commitů, které ze změn vložených do staging area (jakýsi meziprostor do kterého se přidávají všechny změny, které by měly být v následujícím commitu zahrnuty) vytvoří nový záznam v historii repozitáře. Git zároveň podporuje vytváření větví, díky čemuž je možné oddělit vývoj jednotlivých částí aplikace. Tyto větve je poté možné spojovat a díky tomu převést změny z větve jedné do druhé. Git je založen na distribuované architektuře – každý vývojář má kompletní kopii repozitáře ve kterém pracuje lokálně a následně převádí změny z vlastní kopie do kopie centrální uložené na serveru.

GitHub je webová služba umožňující hostování repozitářů vytvořených pomocí nástroje Git. Slouží k uložení centrální kopii repozitáře, kterou si mohou vývojáři stáhnout a dále vyvíjet. Jedná se o nástroj výrazně zjednodušující práci většího množství lidí na jednom projektu. V případě statických webových stránek GitHub zároveň disponuje možností jejich bezplatného hostování přímo ze souborů v repozitáře. Tato funkce se nazývá GitHub Pages.

1. Asynchronní programování v jazyce JavaScript – promise, async, await

Asynchronní zpracování je metoda, při které se jednotlivé úlohy zpracovávají nezávisle na sobě a tím jedna neblokuje dlouhodobě druhou. Oproti paralelnímu zpracování, kdy je více úloh vykonáváno v jeden okamžik současně, každá na samostatné výpočetní jednotce (jádře procesoru), dochází při asynchronnímu zpracování k velmi rychlému přepínání mezi jednotlivými úlohami, které jsou ale zpracovány na jádře jednom. Tento koncept je zásadní v prostředí, kdy většina kódu aplikace běží na hlavním vlákně (takto je tomu i u klientských webových aplikací), protože pokud by nebylo asynchronní zpracování využito došlo by k zamrznutí aplikace při vykonávání dlouhotrvajících operací.

V případě JavaScriptu se asynchronní programování provádí pomocí slibů – promise. U nich se specifikují callback metody, které se vykonají při dokončení úlohy, která se ve slibu nachází vykonání slibu se provádí pomocí metody then.

Příklad:

let p = new Promise((resolve) => {

    setTimeout(() => {

        resolve('Done');

    }, 30000);

 });

 p.then((data) => {console.log(data)});

Po provedení tohoto kódu dojde k výpisu Done až po 30 vteřinách, aplikace je ale po celou dobu čekání responzivní a nezamrzne.

Sliby je zároveň možné odmítnout. To se provádí pomocí vytvoření druhého parametru, který v sobě bude mít uloženou callback funkci pro odmítnutí.

Příklad:

function wait(time) {

    return new Promise((resolve, reject) => {

        if (time < 0) reject('Negative time.');

        setTimeout(

            () => {

                resolve('Done');

            },

            time

        );

    });

}

//čekání na dokončení slibu bez specifikování funkce pro odmítnutí => pokud dojde k odmítnutí program vyhodí chybu

wait(10000).then(

    (data) => {

        console.log('Promise succesfuly resolved with output: ' + data);

    }

);

//specifikace funkce pro odmítnutí pomocí metody catch

wait(-10000).then(

    (data) => {

        console.log('Promise succesfuly resolved with output: ' + data);

    }

).catch(

    (error) => {

        console.log('Promise rejected with error: ' + error);

    }

);

//specifikace funkce pro odmítnutí pomocí druhého argumentu při volání metody then

wait(-10000).then(

    (data) => {

        console.log('Promise succesfuly resolved with output: ' + data);

    },

    (error) => {

        console.log('Promise rejected with error: ' + error);

    }

);

Alternativou k vytváření slibů je použití klíčových slov async a await. Pokud je funkce pomocí slova async označena jako asynchronní, pak vždy vrátí slib, u něhož bude možné čekat na jeho splnění či odmítnutí. Čekání na splnění je možné krom metody then provést pomocí klíčového slova await, které je ovšem možné použít pouze v asynchronních funkcích a metodách. Při použití await již není získán sliba, ale data, která slib vygeneroval. V případě odmítnutí dojde při použití await k vyhození výjimky, kterou je potřeby odchytit pomocí bloku try a catch.

Příklad:

//čekání pomocí specifikování callback funkcí

function waitAndDoSomething(time, something) {

    wait(time).then(

        (data) => {

            console.log('Promise succesfuly resolved with output: ' + data);

            something();

        },

        (error) => {

            console.log('Promise rejected with error: ' + error);

        }

    )

}

//čekání pomocí await – zápis je o něco přehlednější a jednodušší

async function waitWithAwaitAndDoSomething(time, something) {

    try {

        let data = await wait(time);

        console.log('Promise succesfuly resolved with output: ' + data);

        something();

    } catch (error) {

        console.log('Promise rejected with error: ' + error);

    }

}

waitAndDoSomething(10000, () => {

    console.log('Hello');

});

waitWithAwaitAndDoSomething(10000, () => {

    console.log('Hello');

});

1. Dynamické a statické typování – výhody, nevýhody a použití při tvorbě webových aplikací

Při statickém typování proměnných je při jejich deklaraci nutné krom jejich názvu uvést také jejich datový typ. Ten se v průběhu vykonávání programu nemění a je nutné jej dodržovat a nesnažit se do proměnných ukládat hodnoty, které do nich nepatří.

Při dynamickém typování je datový typ proměnných určen za běhu programu na základě hodnot, které jsou do proměnných ukládány.

Výhody dynamického typování jsou:

Méně kódu – jelikož i do již vytvořených proměnných je možné uložit cokoliv, je možné je znovu použít pro uložení nových hodnot a není nutné vytvářet proměnné nové

Datové struktury s libovolnými datovými typy – při vytváření datových struktur se nespecifikuje, jaký datový typ v nich bude uložen a díky tomu je do nich možné vždy uložit cokoliv

Výhody statického typování:

Větší robustnost kódu – díky včasné detekci chyb nedochází k chybám způsobeným špatným datovým typem vstupů. Kód je zároveň přehlednější a daleko snáze rozšiřitelnější. Každá funkce má definované datové typy vstupu a výstupu, díky čemuž není nutné dlouze zjišťovat, co vlastně funkci předat a co očekávat jako návratovou hodnotu.

Teoreticky vyšší výkon

Možnost přetěžování funkcí a metod – více funkcí a metod může mít stejný název, ale odlišný vstup a na základě vstupu je vyhodnoceno, která verze bude provedena.

Lepší nápověda IDE – díky definovaným datovým typům IDE (alespoň to dobré) v každém okamžiku ví, čím dané proměnné disponují a mohou díky tomu lépe napovídat při psaní kódu, čímž tento proces výrazně zjednoduší.

Jazyky nejčastěji používané pro vývoj webových aplikací (PHP, JavaScript) jsou dynamicky typované. V poslední době, ale, díky výhodám, které skýtá, dochází k jejich rozšiřování a umožňování typování statického. V případě JavaScriptu existuje jeho nadstavba TypeScript a v případě PHP je možné povolit typovou kontrolu pomocí declare(strict\_types=1).