TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

KATEGORIE M

Výstup RVP

Žák na základě analýzy problému specifikuje zadání pro tvorbu programu, skriptu nebo webové aplikace.

Učivo (RVP)

specifikace a popis řešeného problému; požadavky na řešení

Vysvětlení

V oblasti **tvorby softwaru** podporovat **rozvoj dovedností systematické analýzy, specifikace úkolů a rozhodování o algoritmickém řešení.** Analýzou je myšlena schopnost systematicky analyzovat problémy v oblasti informatiky a identifikovat klíčové prvky a požadavky spojené s daným úkolem nebo projektem (včetně porozumění potřebám uživatelů). Specifikací zadání pak schopnost převést výstupy analýzy do konkrétních požadavků a specifikací pro tvorbu software. Dále je třeba posilovat **dovednosti, jasně a srozumitelně formulovat zadání,** které bude sloužit jako vodítko pro vývojáře. Zároveň je důležité umět **zdůvodnit a komunikovat svá rozhodnutí.** Výstup se zabývá výhradně analýzou návrhu, nikoliv vlastní tvorbou.

Rozklad výsledku vzdělávání

Porozumění a popis problému, který má být řešen

Žák popíše problém včetně identifikace cíle, potřeb uživatelů a specifikace vstupních údajů.

Specifikace a popis řešeného problému

Žák specifikuje výstup projektu. Vymezí funkce a řešení, které software poskytne.

Požadavky na řešení

Žák definuje požadavky na software zahrnující výkon, bezpečnost a použitelnost.

PRAKTICKÝ PŘÍKLAD 1 — FIRMA

Porozumění a popis problému, který má být řešen

Vývoj webové aplikace pro správu úkolů. Začít formulací problému a cíle. S pomocí jednoduché webové aplikace pro efektivní správu denních úkolů a povinností zvýšit produktivitu a snížit stres z nevyřízených úkolů. Umožnit uživatelům přidávat, organizovat a sledovat své úkoly.

Specifikace a popis řešeného problému

Vytvořit jednoduché a intuitivní uživatelské rozhraní s možností přidávat úkoly pomocí textového pole a tlačítka. Cílem je umožnit uživatelům vytvářet seznam úkolů, označovat úkoly jako dokončené a filtrovat úkoly podle stavu (dokončené/neúplné).

Požadavky na řešení

Definovat požadavky, jako například přidání úkolu, odstranění úkolu, označení úkolu jako dokončené, zobrazení všech úkolů v daném časovém intervalu. Neopomenout zajištění rychlé odezvy aplikace, snadné navigace a přístupnosti na různých zařízeních (responzivní design).

PRAKTICKÝ PŘÍKLAD 2 — ZDRAVOTNICTVÍ

Porozumění a popis problému, který má být řešen

Vývoj aplikace pro sledování hydratace. V prvním kroku definovat a popsat problém. Například jako potřebu vytvořit aplikaci, která by uživatelům pomohla monitorovat denní příjem vody, předcházet dehydrataci a souvisejícím zdravotním problémům, motivovat je k optimální hydrataci.

Specifikace a popis řešeného problému

Následně specifikovat funkce a uživatelské rozhraní aplikace (možnost zaznamenávat množství vypité vody, nastavení denního cíle příjmu vody, připomenutí, když je čas pít, zobrazení statistik a pokroku). Jako cíl stanovit vytvoření přehledného a motivujícího uživatelského rozhraní s grafickým zobrazením denního cíle a aktuálního příjmu.

Požadavky na řešení

Určit softwarové požadavky typu přidávání množství vypité vody, zobrazení celkového denního příjmu, nastavení a úprava denního cíle, zasílání upozornění/připomínek. Dále z hlediska výkonu a použitelnosti požadovat jednoduchost a intuitivnost ovládání, vizuálně přitažlivé uživatelské rozhraní, adaptabilitu na různé velikosti obrazovek (smartphony, tablety).

PRAKTICKÝ PŘÍKLAD 3 — FINANCE

Porozumění a popis problému, který má být řešen

Vývoj aplikace pro správu osobních financí. Nejprve definovat problém, například ve znění: řada lidí má potíže s udržením přehledu o svých financích, což vede k neefektivnímu využívání peněz a finančním potížím. Cílem bude vytvořit intuitivní aplikaci, která pomůže uživatelům lépe spravovat své finance. Poskytnout nástroje pro sledování příjmů a výdajů a plánování rozpočtu.

Specifikace a popis řešeného problému

V dalším kroku vymezit funkce aplikace (možnost zaznamenávat příjmy a výdaje, vytvářet kategorie výdajů, nastavovat měsíční rozpočet a sledovat, jak uživatelé vůči tomuto rozpočtu hospodaří). A dále popsat uživatelské rozhraní aplikace (vytvořit přehledné a motivující uživatelské rozhraní s vizualizací dat pomocí grafů a tabulek pro snadnější porozumění finanční situaci).

Požadavky na řešení

Na závěr stanovit požadavky pro software, mezi které patří například přidávání a kategorizace transakcí, vytváření a sledování rozpočtů, generování reportů o příjmech a výdajích, uživatelsky přívětivé rozhraní, rychlá a spolehlivá aplikace, zabezpečení uživatelských dat.

npi Národní pedagogický institut České republiky
Financováno Evropskou unií NextGenerationEU
Národní plán obnovy
MŠMT MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

KATEGORIE M

Výstup RVP

Žák rozdělí zadání nebo problém na menší části, rozhodne, které je vhodné řešit algoritmicky, své rozhodnutí zdůvodní.

Učivo (RVP)

analýza a dekompozice (rozložení) problému

Vysvětlení

Rozdělením problému je myšlena schopnost rozčlenit celkový problém na menší, lépe zvládnutelné části a **identifikovat klíčové úkoly a souvislosti** mezi nimi. Při rozhodování o algoritmickém řešení je podstatná schopnost **odlišit, které části problému jsou vhodné pro řešení pomocí algoritmů.** Přitom je třeba zvažovat efektivitu, složitost a vhodnost různých algoritmů pro daný úkol. A v neposlední řadě je nezbytné rozvíjet schopnost **vysvětlit volbu určitého postupu nebo algoritmu.** To zahrnuje také zdůvodnění, jak dané rozhodnutí přispívá k efektivnímu řešení.

Rozklad výsledku vzdělávání

Identifikace problému

Žák pochopí zadání nebo problém. Určí hlavní cíl, požadavky a očekávané výsledky.

Analýza a dekompozice problému

Žák rozdělí problém na menší, jednodušší části. Rozumí tomu, že tento postup umožňuje lépe zvládat složité úkoly a efektivněji hledat řešení. Dekompozici zaměří na funkční bloky, dílčí úkoly a strukturu uživatelského rozhraní.

Rozhodnutí o algoritmickém řešení

Žák určí, které části jsou vhodné k algoritmickému (programovému) řešení a které je vhodné řešit jinak.

Zdůvodnění rozhodnutí

Žák dokáže rozhodnutí o algoritmickém řešení pečlivě zdůvodnit. Vysvětlí, proč je daný přístup vhodný.

PRAKTICKÝ PŘÍKLAD 1 — OBCHOD

Identifikace problému

Návrh systému pro správu objednávek v internetovém obchodě. Nejprve definovat potřebu obchodu efektivně spravovat objednávky od jejich přijetí až po expedici. Cílem je pak zajistit, aby každá objednávka byla správně zaznamenána, zpracována a expedována včas a zákazníci byli informováni o stavu jejich objednávky.

Analýza a dekompozice problému

Problém správy objednávek rozdělit na klíčové části. (1) Přijetí objednávky: zaznamenat nové objednávky do systému. (2) Zpracování objednávky: zkontrolovat dostupnost produktů a připravit objednávku k expedici. (3) Expedice objednávky: zabalit a odeslat objednávku zákazníkovi. (4) Komunikace se zákazníkem: informovat zákazníka o stavu objednávky.

Rozhodnutí o algoritmickém řešení

Pro každou část problému rozhodnout, zda a jak ji řešit algoritmicky. (1) Přijetí objednávky: algoritmicky zpracovat vstupní data objednávky a zaznamenat je do databáze. (2) Zpracování objednávky: použít algoritmus pro kontrolu skladových zásob a automatické generování seznamu produktů k expedici.

Zdůvodnění rozhodnutí

Použití algoritmického řešení zdůvodnit. (1) Přijetí objednávky: umožňuje rychlé a přesné zaznamenání objednávek do systému bez manuálních chyb. (2) Zpracování objednávky: minimalizuje riziko chyb a zrychluje proces přípravy objednávek. (3) Expedice objednávky: zvyšuje efektivitu a zkracuje čas potřebný k expedici. (4) Komunikace se zákazníkem: zvyšuje spokojenost zákazníků a snižuje potřebu jejich dotazů na zákaznickou podporu.

PRAKTICKÝ PŘÍKLAD 2 ZDRAVOTNICTVÍ

Identifikace problému

Systém pro správu lékařských záznamů v nemocnici. Identifikovat potřebu efektivního systému pro správu lékařských záznamů, který zajistí snadný přístup, aktualizaci a ochranu pacientových dat. Určit cíl, tedy zlepšit správu informací o pacientech v nemocnici.

Analýza a dekompozice problému

Správu lékařských záznamů rozdělit na klíčové části. (1) Přijetí pacienta: zaznamenat základní informace o nově přijatém pacientovi. (2) Aktualizace lékařského záznamu: zaznamenat diagnostické výsledky, předepsané léky a provedené procedury. (3) Uchování a ochrana dat: zajistit bezpečnost a soukromí lékařských záznamů. (4) Přístup k záznamům: umožnit rychlý a snadný přístup pro autorizovaný personál.

Rozhodnutí o algoritmickém řešení

Pro každou část problému určit způsob algoritmického řešení. (1) Přijetí pacienta: algoritmicky zpracovat a uložit základní informace o pacientovi do databáze. (2) Aktualizace lékařského záznamu: použít algoritmy pro automatické zaznamenávání a aktualizaci dat v záznamech pacientů. (3) Uchování a ochrana dat: implementovat algoritmy pro šifrování dat a kontrolu přístupu k zajištění bezpečnosti informací. (4) Přístup k záznamům: zaměřit se na vývoj algoritmu pro efektivní vyhledávání a získávání informací z databáze lékařských záznamů.

Zdůvodnění rozhodnutí

Zdůvodnit algoritmické řešení. (1) Přijetí pacienta: zajistí, že všechny základní informace budou správně zaznamenány a rychle dostupné. (2) Aktualizace lékařského záznamu: minimalizuje riziko lidských chyb a zvyšuje efektivitu zpracování pacientových dat. (3) Uchování a ochrana dat: chrání citlivé informace před neoprávněným přístupem. (4) Přístup k záznamům: umožňuje rychlý přístup k potřebným informacím, což je zásadní pro rychlou lékařskou péči.

PRAKTICKÝ PŘÍKLAD 3 MĚSTSKÉ PLÁNOVÁNÍ

Identifikace problému

Systém pro správu dopravního zatížení ve městě. Nejprve určit problém, tj. město čelí výzvě v podobě rostoucího dopravního zatížení, které vede k dopravním zácpám a zvyšuje emise. Za cíl stanovit optimalizaci průtoku dopravy tak, aby se snížil výskyt dopravních komplikací a zlepšila kvalita ovzduší.

Analýza a dekompozice problému

Problém dopravního zatížení rozdělit na klíčové části. (1) Monitorování dopravy: sbírat data o aktuálním dopravním zatížení na hlavních tazích. (2) Analýza dopravních toků: určit kritické body s největším dopravním zatížením. (3) Plánování dopravních opatření: navrhnout řešení pro zlepšení průtoku dopravy. (4) Implementace a hodnocení: zavést dopravní opatření a sledovat jejich efekt.

Rozhodnutí o algoritmickém řešení

Pro každou část problému určit algoritmické řešení. (1) Monitorování dopravy: použít algoritmy pro zpracování dat z dopravních senzorů a kamer. (2) Analýza dopravních toků: algoritmicky analyzovat data pro identifikaci kritických bodů. (3) Plánování dopravních opatření: vyvinout algoritmy pro simulaci dopravních toků a navrhování opatření. (4) Implementace a hodnocení: vytvořit algoritmy pro monitorování výsledků zavedených opatření, upravit podle potřeby.

Zdůvodnění rozhodnutí

Zvolené algoritmické řešení zdůvodnit. (1) Monitorování dopravy: umožňuje real-time sledování dopravního zatížení, což je základ pro další analýzu. (2) Analýza dopravních toků: umožňuje cílené plánování opatření. (3) Plánování dopravních opatření: pomáhá předvídat dopady různých opatření na dopravní tok, což umožňuje efektivní plánování. (4) Implementace a hodnocení: umožňuje rychlé zjištění úspěšnosti a potřebných úprav.

npi Národní pedagogický institut České republiky
Financováno Evropskou unií NextGenerationEU
Národní plán obnovy
MŠMT MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

KATEGORIE
M

TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

Výstup RVP

Žák navrhne algoritmy a datové struktury podle specifikace zadání a zapíše je vhodnou formou.

Učivo (RVP)

základní koncepce tvorby programů (například proměnná a datový typ, řídící příkazy, cykly); návrh algoritmů a datových struktur; zápis algoritmu vhodnou formou (například blokové schéma, přirozené a formální jazyky, skriptovací a programovací jazyk); využívání hotových komponent

Vysvětlení

Je třeba rozvíjet schopnosti žáků v oblasti návrhu algoritmů, datových struktur a implementace jednoduchých programů či aplikací. Správná datová struktura může výrazně zlepšit efektivitu algoritmů při práci s daty, ale i **zefektivnit proces tvorby programu.** Orientace v problematice je důležitá rovněž pro **komunikaci s vývojáři** a implementaci samotnou.

Rozklad výsledku vzdělávání

Identifikace klíčových konceptů a operací

Žák rozpozná a definuje základní prvky, které budou v algoritmu využity. Vybere vhodné datové typy pro uložení informací. Definuje proměnné. Vytváří logické struktury programu.

Návrh datových struktur

Žák určí nejvhodnější způsob uložení a manipulace s daty pro efektivní provedení algoritmu. Vybere mezi základními datovými strukturami (proměnná, pole, seznamy, zásobníky) podle typu dat a operací, které se na nich budou provádět.

Návrh algoritmů

Žák na základě dekompozice problému vytvoří posloupnost kroků pro dosažení výsledků definovaných v zadání. Rozdělí problém na menší, řešitelné části. Vytvoří logický postup pro každou část. Využívá pseudokódy nebo diagramů pro vizualizaci toku dat a operací s použitím řídících příkazů (řídící struktura například cyklus, podmiňovací příkaz aj.).

Zápis algoritmů a datových struktur vhodnou formou

Žák převede návrhy do čitelné a pochopitelné formy, která může být sdílena nebo implementována. Používá pseudokódy, diagramy toku nebo jiné nástroje pro zápis algoritmů.

PRAKTICKÝ PŘÍKLAD 1 — INTERNETOVÝ OBCHOD

Identifikace klíčových konceptů a operací

Návrh jednoduchého systému pro správu objednávek v internetovém obchodě. Nejprve je potřeba rozpoznat, jaké informace bude systém potřebovat a jak budou data zpracována. Dále definovat proměnné objednávky (ID objednávky, seznam produktů, cena, datum, stav). Rozhodnout o datových typech (čísla, řetězce, datum). Naplánovat řídící struktury, tedy podmínky (například pokud je objednávka již zpracována, neumožnit její změnu) a cykly (pro procházení seznamu objednávek).

Návrh datových struktur

Následně určit, jak budou data o objednávkách uložena a organizována. Použít seznam nebo pole pro ukládání objednávek. Vytvořit strukturu objednávky obsahující všechny potřebné informace (ID, produkty, cena atd.). Rozhodnout o využití hash tabulky pro rychlé vyhledávání objednávek podle ID.

Návrh algoritmů

Vytvořit postupy pro každou klíčovou funkci systému. (1) Vytvoření objednávky: sběr dat o produktu, výpočet celkové ceny, přidělení ID a data objednávky, ukládání objednávky. (2) Zobrazení objednávek: procházení seznamu všech objednávek a jejich zobrazení uživateli. (3) Úprava objednávky: vyhledání objednávky podle ID, umožnění změny produktů nebo množství, pokud to stav objednávky dovoluje. (4) Mazání objednávky: vyhledání objednávky podle ID a její odstranění ze systému.

Zápis algoritmů a datových struktur vhodnou formou

Formulovat návrhy algoritmů a struktur tak, aby byly připraveny pro implementaci. Použít pseudokód pro zápis postupů.

PRAKTICKÝ PŘÍKLAD 2 — ŠKOLA / ŠKOLNÍ KNIHOVNA

Identifikace klíčových konceptů a operací

Vytvoření systému pro správu knižního fondu školní knihovny. Nejprve stanovit proměnné a datové struktury potřebné pro správu dat. Definovat proměnné pro knihy a výpůjčky. Rozhodnout o vhodných datových typech: řetězce pro název knihy, autora a žánr; číslo nebo datum pro záznamy o půjčkách. Vybrat řídicí struktury pro operace, jako je vyhledávání knih nebo sledování stavu výpůjček.

Návrh datových struktur

Určit, jak budou informace o knihách a výpůjčkách organizovány a uloženy. Použít seznamy nebo pole pro ukládání informací o knihách. Vytvořit struktury nebo třídy pro knihy obsahující všechny potřebné informace. Implementovat seznam nebo jinou datovou strukturu pro sledování výpůjček (umožní snadné vyhledávání a aktualizace).

Návrh algoritmů

Vytvořit logické postupy pro realizaci klíčových funkcí systému. (1) Vyhledávání knih: umožnit uživatelům vyhledávat knihy podle různých kritérií. (2) Půjčování knih: zaznamenat datum a uživatele, který si knihu vypůjčil, a nastavit stav knihy na „vypůjčeno“. (3) Vracení knih: akualizovat stav knihy na „dostupné“ a zaznamenat datumu vrácení. (4) Zobrazení historie výpůjček: poskytnout přehled o tom, kdo a kdy si knihu půjčil a kdy ji vrátil.

Zápis algoritmů a datových struktur vhodnou formou

Připravit návrhy algoritmů a struktur pro další kroky vývoje. Použít pseudokódy nebo diagramy toku pro vizualizaci algoritmů.

npi Národní pedagogický institut České republiky
Financováno Evropskou unií NextGenerationEU
Národní plán obnovy
MŠMT MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

KATEGORIE M

Výstup RVP

Ve vztahu k charakteru a velikosti vstupu žák hodnotí algoritmy a datové struktury podle různých hledisek, porovná a vybere pro řešený problém ty nejvhodnější; vylepší algoritmus podle daného hlediska.

Učivo (RVP)

návrh algoritmů a datových struktur

Vysvětlení

Pro rozvíjení schopností v oblasti návrhu algoritmů, datových struktur a implementace jednoduchých programů či aplikací je žádoucí učit se hodnotit **algoritmy a datové struktury v závislosti na charakteru a velikosti vstupu.** Vhodné je zařazovat například úlohy vyžadující **porovnání efektivity algoritmů pro vyhledávání v seznamu při různých velikostech dat.** Dále je žádoucí posilovat **schopnosti vylepšit existující algoritmus podle daného hlediska** (časová složitost nebo paměťová efektivita). Příkladem může být optimalizace algoritmu pro hledání nejkratší cesty v grafu.

Rozklad výsledku vzdělávání

Hodnocení algoritmů a datových struktur

Žák hodnotí algoritmy a datové struktury podle různých kritérií, jako jsou časová a prostorová složitost nebo vhodnost pro specifické problémy. Rozumí tomu, že tento postup je klíčový pro výběr nejefektivnějšího řešení.

Optimalizace a vylepšení

Žák optimalizuje a vylepšuje vybrané algoritmy a datové struktury tak, aby co nejlépe vyhovovaly konkrétním potřebám a omezením daného projektu. Při optimalizaci bere v úvahu možnosti dalšího vývoje produktu.

PRAKTICKÝ PŘÍKLAD 1 INTERNETOVÝ OBCHOD

Hodnocení algoritmů a datových struktur

Vývoj systému pro doporučování produktů. Nejprve je třeba posoudit, jak efektivně algoritmus pracuje v různých situacích (například má-li uživatel malou historii nákupů, či má zájem o velmi specifické produkty). Při hodnocení porovnat různé algoritmy doporučování, jako jsou kolaborativní filtrace a obsahově založené filtrace. Vyhodnotit, který je pro dané potřeby nejvhodnější (rychlost a relevance doporučení).

Optimalizace a vylepšení

Provést optimalizaci, tedy pravidelně vyhodnocovat výkon systému sledováním reakcí uživatelů na doporučené produkty. Na základě analýzy upravit váhy v algoritmu pro zlepšení relevance doporučení. Experimentovat s různými metodami doporučování pro zvýšení přesnosti (včetně zohlednění aktuálních trendů a změn v chování uživatelů). Zvyšovat úroveň personalizace rozvíjením sofistikovanějších profilů uživatelů (pro lepší odraz jejich preference a chování). Optimalizovat technické aspekty systému (pro zajištění rychlosti a škálovatelnosti). Provét testování (pro ověření účinnosti provedených změn, například A/B testování).

PRAKTICKÝ PŘÍKLAD 2 DOMÁCNOST

Hodnocení algoritmů a datových struktur

Vývoj chytrého domácího termostatu, který se učí z uživatelských návyků a optimalizuje nastavení teploty pro energetickou účinnost a pohodlí uživatelů. Úkolem je posoudit účinnost algoritmu v různých domácnostech a za různých klimatických podmínek. Při hodnocení srovnat různé přístupy strojového učení (regresní modely a neuronové sítě) pro predikci optimálních teplotních nastavení.

Optimalizace a vylepšení

Upravit algoritmus na základě zpětné vazby od uživatelů a analýzy účinnosti v různých podmínkách. Při optimalizaci se zaměřit na zlepšení energetické účinnosti a pohodlí na základě shromážděných dat a uživatelské zpětné vazby. Pro vylepšení implementovat funkce rozpoznání specifických situací (dovolené, zvláštní události) k dalšímu snížení spotřeby energie bez kompromisu v pohodlí.

PRAKTICKÝ PŘÍKLAD 3 ZDRAVOTNICTVÍ

Hodnocení algoritmů a datových struktur

Vývoj systému pro analýzu zdravotních dat. Nejprve posoudit efektivitu algoritmů pro rychlou a přesnou analýzu zdravotních dat.

Optimalizace a vylepšení

Následně zlepšit algoritmus a aplikace na základě zpětné vazby od uživatelů a nových zdravotnických poznatků.

TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

KATEGORIE M

Výstup RVP

Žák vytvoří jednoduchý spustitelný program, skript, nebo webovou aplikaci.

Učivo (RVP)

zápis algoritmu vhodnou formou (například blokové schéma, přirozené a formální jazyky, skriptovací a programovací jazyk); využívání hotových komponent

Vysvětlení

Rozvíjet schopnosti žáků **v oblasti implementace** jednoduchých programů či aplikací. Důraz by měl být kladen na **jazyky vhodné pro začátečníky.** Cílem je umožnit efektivní implementaci algoritmů do konkrétního programovacího kódu. Seznámit žáky s praktickými **příklady integrace komponent do vlastního projektu.** Tím se zvýší efektivita vývoje a umožní se využít již existující řešení.

Rozklad výsledku vzdělávání

Seznámení se s programovacími a skriptovacími jazyky

Žák se seznámí s různými programovacími a skriptovacími jazyky. Vysvětlí jejich základní rozdíly a oblasti použití. Rozumí základní konstrukci a syntaxi vybraného programovacího nebo skriptovacího jazyka.

Práce s hotovými komponentami

Žák využívá knihoven a frameworků. Rozumí významu a možnostem použití těchto hotových komponent pro zjednodušení vývoje a zlepšení funkcionalit vytvářených programů. Má praktickou zkušenost s jejich integrací do vlastního projektu.

PRAKTICKÝ PŘÍKLAD 1 MALOOBCHOD

Seznámení se s programovacími a skriptovacími jazyky

Vývoj jednoduchého skladového systému. Cílem je vytvořit systém, který umožní maloobchodnímu podniku sledovat zásoby produktů a usnadnit proces inventury. Zvolit jazyk Python, který je přátelský k začátečníkům a disponuje knihovnami usnadňujícími práci s daty. Seznámit se se základy Pythonu, tj. základní syntaxí včetně proměnných, podmínek (if-else), cyklů (for, while) a základními funkcemi pro práci se seznamy a soubory.

Práce s hotovými komponentami

Využít knihovnu Pythonu (například Pandas) pro práci s daty, která umožní snadné spravování a analýzu dat skladu ve formě tabulek. Knihovnu integrovat do projektu. Tedy importovat knihovnu, vytvořit DataFrame (tabulku) pro evidenci zásob a provést základní operace, jako přidávání nových záznamů o zboží nebo aktualizace stávajících záznamů.

PRAKTICKÝ PŘÍKLAD 2 HOTELNICTVÍ

Seznámení se s programovacími a skriptovacími jazyky

Vývoj jednoduchého rezervačního systému pro malý hotel. Úkolem je vytvořit systém, který umožní malému hotelu spravovat rezervace pokojů, zaznamenávat údaje o hostech a sledovat dostupnost pokojů. Vybrat vhodný jazyk pro projekt. Doporučit lze PHP spolu s databází MySQL (často používaný pro vývoj webových aplikací, včetně rezervačních systémů). Seznámit se základy PHP pro zpracování požadavků na serveru a MySQL pro správu databáze, která bude obsahovat informace o rezervacích a dostupnosti pokojů.

Práce s hotovými komponentami

Vytvořit webové formuláře pro zadávání informací o rezervaci a PHP skripty pro zpracování těchto informací a komunikaci s databází.

PRAKTICKÝ PŘÍKLAD 3 ZEMĚDĚLSTVÍ

Seznámení se s programovacími a skriptovacími jazyky

Vývoj aplikace pro sledování a predikci výnosů z polí. V prvním kroku se seznámit se s Pythonem a jeho knihovnami pro efektivní práci s daty a analýzu.

Práce s hotovými komponentami

Vytvořit jednoduchý skript v Pythonu, který zpracuje zadaná data, provede potřebné výpočty a vizualizuje výsledky vhodným způsobem.

TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

KATEGORIE M

Výstup RVP

Žák testuje spustitelný program, skript nebo webovou aplikaci; najde, specifikuje a opraví případnou chybu.

Učivo (RVP)

druhy chyb, chybové hlášky, neočekávané ukončení a zamrznutí; způsoby a druhy testování softwaru; spotřeba výpočetních a jiných zdrojů

Vysvětlení

Žádoucí je zaměřit se na rozvíjení následujících schopností. (1) Provádět testování výsledného softwaru s cílem odhalit případné chyby nebo nedostatky. **(2) Identifikovat a lokalizovat chyby** ve spustitelném programu, skriptu nebo webové aplikaci. **(3) Detailně popsat chybu a podniknout** potřebné **kroky k nápravě.**

Rozklad výsledku vzdělávání

Seznámení s chybami v softwaru

Žák rozliší druhy chyb (syntaktické, chyby v logice programu, runtime chyby). Dokáže interpretovat chybové hlášky poskytované vývojovými nástroji nebo prostředím, ve kterém je software spuštěn. Orientuje se v příčinách, které mohou vést k neočekávanému ukončení programu nebo jeho zamrznutí. Objasní, jak tyto situace řešit.

Způsoby a druhy testování softwaru

Žák rozumí rozdílům mezi manuálním testováním (testy prováděné ručně) a automatickým testováním (využívající skripty a nástroje pro automatizaci testů). Rozlišuje funkční a nefunkční testování – testování funkcí, které má software vykonávat, a aspektů jako výkon, bezpečnost a použitelnost.

Spotřeba výpočetních a jiných zdrojů

Žák si osvojí způsoby, jak monitorovat a analyzovat spotřebu zdrojů (CPU, paměť RAM, úložiště) během spuštění programu nebo aplikace. Diskutuje o strategiích pro optimalizaci využití zdrojů, které zvyšují efektivitu softwaru na různém hardwaru.

PRAKTICKÝ PŘÍKLAD 1 GASTRONOMIE

Seznámení s chybami v softwaru

Vývoj webové aplikace pro správu objednávek v restauraci. Úkolem je danou webovou aplikaci otestovat. První krok proto vyžaduje seznámení se s možnými typy chyb, které mohou v aplikaci nastat (například chyby v logice programu by mohly způsobit, že se objednávky nesprávně přiřazují k stolům, nebo že systém nesprávně vypočítává celkovou cenu objednávky). Při identifikaci chyb vyzkoušet různé scénáře použití aplikace, například zadání objednávky, změnu objednávky, zrušení objednávky. Sledovat, jak aplikace reaguje, události zaznamenat (chybové hlášky, neočekávané ukončení).

Způsoby a druhy testování softwaru

Aplikaci manuálně otestovat (přihlášení do aplikace, zadávání a správa objednávek, kontrola správnosti výpočtů, ověřování, zda všechny funkce aplikace pracují podle očekávání). Vytvořit seznam testovacích scénářů, které pokryjí všechny základní funkce aplikace, jako je přidání nové objednávky, úpravu stávající objednávky, platbu a generování reportů. Pro každý scénář specifikovat očekávaný výsledek.

Spotřeba výpočetních a jiných zdrojů

Pro monitorování výkonu webového serveru, na kterém aplikace běží, použít dostupné nástroje. Sledovat, jak aplikace ovlivňuje využití procesoru, paměti a diskového prostoru, zvláště při zpracování velkého množství objednávek. Pokud je například zjištěno, že aplikace spotřebovává příliš mnoho zdrojů, může to znamenat, že některé části kódu nejsou efektivní. Takové části identifikovat a diskutovat o možných řešeních s vývojářem, například o optimalizaci databázových dotazů.

PRAKTICKÝ PŘÍKLAD 2 DOMÁCNOST

Seznámení s chybami v softwaru

Vývoj a testování softwaru pro inteligentní domácnost – aplikace pro řízení teploty v domácnosti. Vybrat aplikaci, která umožňuje uživatelům nastavovat a regulovat teplotu v různých místnostech domu prostřednictvím mobilního zařízení. Seznámit s typickými chybami, které se mohou objevit (chyby v komunikaci s termostaty, nesprávné čtení teplotních dat, selhání v algoritmech pro regulaci teploty). Experimentovat s aplikací (nastavit teploty pro různé místnosti; vytvořit časové plány pro automatickou regulaci teploty a sledovat, jak aplikace na příkazy reaguje). Jakékoli nesrovnalosti nebo selhání v očekávaném chování zaznamenat.

Způsoby a druhy testování softwaru

Provést manuální testy ovládání teploty (například nastavením teploty v jednotlivých místnostech a ověřením, že teplota se reguluje podle nastavení). Taktéž otestovat reakci systému na rychlé změny nastavení a jeho schopnost udržovat stabilní teplotu. Vytvořit scénáře, které simulují běžné uživatelské situace (snížení teploty v noci nebo automatické vyhřívání místnosti před příchodem domů). Pro každý scénář specifikovat správnou funkčnost. Sledovat, zda je dosahováno očekávaných výsledků.

Spotřeba výpočetních a jiných zdrojů

Použít nástroje pro monitorování spotřeby energie a výpočetního výkonu potřebného pro spuštění aplikace. Zjistit, zda aplikace efektivně komunikuje s termostaty a zda nezpůsobuje zbytečné zatížení domácí sítě nebo spotřebu energie. Objeví-li se problémy s výkonem nebo nadměrnou spotřebou zdrojů, hledat způsoby optimalizace kódu (zvýšit efektivitu komunikace s termostaty, snížit energetickou spotřebu aplikace).

PRAKTICKÝ PŘÍKLAD 3 ZDRAVÍ, SPORT

Seznámení s chybami v softwaru

Vývoj a testování mobilní aplikace pro sledování fitness aktivity. Jedná se o aplikaci, která uživatelům umožňuje sledovat jejich denní kroky, běžecké trasy, spálené kalorie a monitorovat další fitness aktivity. Nejprve je třeba pochopit potenciální problémy a chyby (nepřesnosti ve sledování aktivity, chyby v synchronizaci dat, problémy s uživatelským rozhraním apod.). Pro identifikaci chyb provést různé aktivity (chůze, běh, cvičení) s cílem ověřit, zda je aplikace správně sleduje a zaznamenává. Nesrovnalosti nebo selhání aplikace při sledování nebo zobrazování dat zapsat.

Způsoby a druhy testování softwaru

Zaměřit se na manuální testování uživatelského rozhraní aplikace, funkčnost sledování aktivity a přesnost zaznamenávaných dat. Vytvořit testovací scénáře, které simulují reálné podmínky užití aplikace, například sledování aktivity v různých prostředích (venku, uvnitř, na různých površích) a při různých rychlostech. Každý scénář by měl mít specifikované očekávané výsledky pro množství kroků, vzdálenost a kalorie.

Spotřeba výpočetních a jiných zdrojů

Při analýze spotřeby sledovat, jak aplikace ovlivňuje výdrž baterie mobilního zařízení a využití dat. Důležité je, aby aplikace efektivně využívala systémové zdroje, nevyčerpávala baterii zařízení příliš rychle nebo nevyužívala příliš mnoho mobilních dat. Jsou-li zjištěny problémy s výkonem nebo spotřebou, zaměřit se na identifikaci možných řešení, jako je optimalizace kódu pro lepší výkon a efektivnější využití baterie a dat.

TVORBA, TESTOVÁNÍ A PROVOZ SOFTWARU

VYSVĚTLENÍ A PŘÍKLADY K NOVÉMU RVP PRO SOV

KATEGORIE M

Výstup RVP

Žák spolupracuje při tvorbě programu s další osobou, popisuje strukturu programu další osobě.

Učivo (RVP)

verze programu, instalace a aktualizace programu; hlášení a evidence závad, logování a sledování provozu; nápověda a licence programu

Vysvětlení

Při vývoji softwaru je potřebné umět efektivně spolupracovat, včetně schopnosti detailně a pro druhého srozumitelně **popsat strukturu** programu, včetně organizace kódu, modulů a vzájemného propojení. Nezbytné je dále **rozumět konceptu verzí softwaru** a být schopen identifikovat, jaké změny byly provedeny v jednotlivých verzích programu. Dále je třeba umět **instalovat program na cílovém zařízení a provést aktualizace** na novější verzi. V neposlední řadě efektivně **hlásit a evidovat závady** v programu, rozumět významu **logování a jeho nastavení, vytvořit** uživatelsky přívětivou **nápovědu k programu** a orientovat se **v licenčních podmínkách** a právních aspektech používání programu.

Rozklad výsledku vzdělávání

Spolupráce při tvorbě programu

Žák má zkušenost s plánováním programu a diskusí o jeho cílech, funkcích, uživatelském rozhraní a podmínkách použití. Uvědomuje si důležitost konsensu, efektivní komunikace a týmové práce. Používá verzovací nástroje pro sledování změn a usnadnění spolupráce. Po vytvoření první verze programu provádí testování (manuální i automatizované). Dokumentuje struktury programu, použité algoritmy a další klíčové informace. Vytváří uživatelskou dokumentaci (nápověda, FAQ). Rozumí licenčním podmínkám pro software (vč. omezení použití, distribuce a modifikace softwaru).

Běh a provoz programu

Žák spravuje různé verze programu. Zajišťuje bezpečnou distribuci, instalaci a aktualizace softwaru. Rozumí základům verzování, správně aplikuje aktualizace a opravy. Zná techniky pro hlášení chyb, logování a monitorování softwaru. Orientuje se v nastavení logovacích nástrojů, interpretaci logů a využití monitorovacích nástrojů pro sledování výkonu softwaru v reálném čase.

PRAKTICKÝ PŘÍKLAD 1 GASTRONOMIE

Spolupráce při tvorbě programu

Vývoj webové aplikace pro online objednávky pizzy. Nejprve je třeba rozdělit si úkoly mezi členy týmu (design, backend, databáze). Ve vývoji spolupracovat na funkcích aplikace, jako je výběr pizzy a platební systém. Pravidelně komunikovat. Použít verzovací systém (například Git) pro sdílení kódu.

Běh a provoz programu

Týmově testovat aplikaci na chyby a uživatelskou přívětivost. Po spuštění aplikace monitorovat její provoz. Řešit technické problémy a sledovat výkon. Provést aktualizace na základě zpětné vazby uživatelů.