VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

ROZŠÍŘENÍ PRO REAL-TIME SDÍLENÍ OBSAHU

EXTENSION FOR REAL-TIME CONTENT SHARING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PAVEL PODLUŽANSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. ZDENĚK VAŠÍČEK, Ph.D.
SUPERVISOR

BRNO 2021

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

24083

Student: **Podlužanský Pavel**
Program: Informační technologie
Název: **Rozšíření pro real-time sdílení obsahu**
Real-time Content Sharing Extension
Kategorie: Web

Zadání:

1. Seznamte se s editorem Visual Studio Code a způsobem tvorby modulů umožňujících rozšířit funkcionalitu tohoto editoru. Prostudujte tzv. Notebook API pro práci s poznámkovými bloky. Dále se seznamte s vhodnými technologiemi podporujícími kontinuální přenos dat nebo-li streamování (např. HTTP chunking nebo Websocket).
2. Navrhněte systém umožňující online streaming obsahu poznámkového bloku skrze internetové připojení tak, aby bylo možné vhodně podpořit výuku programovacích jazyků, tj. dovolit studentům mít na svých zařízeních aktuální kopii bloku učitele.
3. Navržený systém implementujte formou rozšíření do editoru Visual Studio Code a tenkého serveru implementovaného ve vhodném jazyce. Pro komunikaci s klienty zvolte vhodnou asynchronní technologii a pokuste se minimalizovat požadavky na šířku přenosového kanálu.
4. Vyhodnoťte parametry navrženého řešení.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Vašíček Zdeněk, doc. Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 30. července 2021
Datum schválení: 30. října 2020

Abstrakt
Cieľom tejto bakalárskej práce je navrhnúť a implemetovať rozšírenie pre editor Visual Studio Code, ktoré umožní zdieľať obsah poznámkových blokov. Cieľová skupina sú najmä učitelia a študenti, kde učiteľ zdieľa obsah svojho poznámkového bloku študentom a na základe tohto zdieľania ich učí programovanie.

Abstract
The aim of this thesis is to design and implement an extension for Visual Studio Code editor, which allow to share the content of notebooks. The target group is teachers and students, where teacher shares content of his notebook to students and he teach them how to programming

Kľúčové slová
Visual Studio Code, rozšírenie, TypeScript, WebSocket, node.js, poznámkový blok

Keywords
Visual Studio Code, extension, TypeScript, WebSocket, node.js, notebook

Citácia
PODLUŽANSKÝ, Pavel. *Rozšíření pro real-time sdílení obsahu.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Ing. Zdeněk Vašíček, Ph.D.

Rozšíření pro real-time sdílení obsahu

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Doc. Ing. Zdenka Vašíčka, Ph.D. Uviedol som všetky literárne pramene, publikácie a dalšie zdroje, z ktorých som čerpal.

. . . .
Pavel Podlužanský
29. júla 2021

Poďakovanie

Touto formou by som rád poďakoval vedúcemu mojej práce, pánovi Doc. Ing. Zdeňkovi Vašíčkovi, Ph.D. za odborné vedenie, pomoc a všetky poskytnuté užitočné rady v rámci riešenia tejto práce.

Obsah

Kapitola 1

Úvod

Trávenie času na internete je v súčasnosti veľmi populárne. Ľudia každodenne navštevujú rôzne internetové stránky, sledujú seriály, či iné druhy videí, prípadne hrajú online hry. Skrátka používajú rôzne internetové aplikácie. Z veľkej časti týchto aplikácií, či hier sa v nich nachádza spoločný menovateľ a to komunikácia, alebo zdieľanie dát v reálnom čase. Aj vďaka tomutu ľudia tak radi používajú tieto aplikácie. Povedal by som, že v poslednom roku zaznamenali aplikácie, ktoré sa využívajú napríklad na online hovory v reálnom čase veľkú popularitu. Dôsledkom môže byť totiž pandémia COVID-19. Takéto aplikácie boli využívané na komunikáciu v rámci mnohých firiem, rodín, či učiteľov so žiakmi. Technológie by mali slúžiť aj na zlepšenie edukácie. Dalo by sa povedať, že sa ukázalo, aké potrebné je mať dostupných viacero aplikácii na podporu edukačných činností a to nie len počas pandémie.

Tým, ako sa rýchlo vyvýjajú technológie, je vhodné aplikovať tento technologický pokrok aj v oblasti vzdelávania. V dnešnej dobe totiž technológie dokážu častokrát uľahčiť mnoho vecí, či už z pohodlia domova, alebo aj v rámci pracovného prostredia. Je preto vhodné implementovať aplikáciu na jednoduché zdieľanie notebookov v rámci Visual Studio Code. Takáto aplikácia dokáže študentom a takisto aj vyučujúcim uľahčiť výučbu a možno aj pomôcť efektívnejšie využiť čas.

Táto práca sa zameriava na vytvorenie rozšírenia pre takú podporu edukácie, aby bolo možné vhodne podporiť výuku programovacích jazykov. Aplikácia by mala dovoliť študentom zobraziť si poznámkový blok v rámci editoru Visual Studio Code, ktorý im je zdieľaný ich učiteľom. Pred začatím písania práce to bol unikátny projekt. Donedávna totiž ešte v spomínanom editore nebolo možné spolupracovať na poznámkových blokoch, ale to sa zmenilo. V priebehu písania tejto práce bolo známe rozšírenie LiveShare od spoločnosti Microsoft obohatené o možnosť spolupracovať na poznámkových blokoch v reálnom čase.

Práca je členená nasledovne. Prvá kapitola sa venuje zdieľaniu obsahu v reálnom čase. V nej sú popísané jednotlivé protokoly a metódy ako zdieľanie obsahu v reálnom čase dosiahnuť. Druhá kapitola sa zameriava na samotný editor Visual Studio Code, kde je popísaný tento editor, jeho funkcie a spôsob tvorby rozšírení v tomto editore. V tejto kapitole je potom predstavený aj samotný jazyk TypeScript, ktorý sa využíva na tvorbu rozšírení pre Visual Studio Code. Tretia kapitola popisuje návrh danej aplikácie. Konkrétne popisuje návrh prehliadača poznámkových blokov a spôsob komunikácie medzi serverom a klientmi. Nasledujúca kapitola s názvom implementácia popisuje štruktúru projektu a spôsoby, akými sú jednotlivé časti implementované. V závere je nakoniec popísané zhrnutie celej problematiky.

Kapitola 2

Zdieľanie obsahu v reálnom čase

Táto kapitola popisuje oblasti, ktoré sú dôležité a nutné pre vypracovanie samotného riešenia. Najskôr je popísané zdieľanie a synchronizácia obsahu v reálnom čase. Ďalej sú popísané jednotlivé internetové protokoly dôležité pri zdieľaní obsahu v reálnom čase. Detailnejší popis je potom venovaný najmä protokolu **Websocket**, ako aj knižnici **socket.io**. Nakoniec sú predstavené podobné aplikácie, ktoré sa zaoberajú rovnakou úlohou a to zdieľaním poznámkových blokov medzi rôznymi používateľmi a takisto aplikácie, ktoré sú určené na podporu výuky.

2.1 Synchronizácia obsahu

Synchronizácia obsahu je čoraz častejšie využívaná pri rôznych aplikáciach. Veľmi populárne aplikácie, ktoré slúžia na synchronizáciu súborov sú napríklad Google dokumenty[1]. Na jeden dokument zároveň môže byť pripojených naraz viacero používateľov, ktorí ho môžu v reálnom čase upravovať, prípadne pozorovať zmeny vykonané inými užívateľmi.

2.1.1 Základná metodika

Okamžite po pripojení k zdieľanému súboru môže byť pokojne stiahnutý celý súbor. Avšak pri úpravach je nutné riešiť **synchronizáciu súborov**. Tá má za úlohu zosynchronizovať dáta v súboroch u zdieľateľa a prijímateľov, tak aby mal každý užívateľ v tom istom čase na svojom zariadení rovnaký súbor. Teda v prípade jednosmerného zdieľania to znamená, že odosielateľ odosiela zmeny, ktoré v súbore aktuálne robí a prijímateľ tieto zmeny môže pozorovať následne potom, keď ich obdrží.

Dôležitým faktorom pri synchronizácii súborov je ich porovnanie. Vždy je nutné zistiť, v čom sa súbory odlišujú. Existujú rôzne pohľady na porovnavánie dvoch súborov. Jedným zo spôsobov, akým je možné uvažovať o rozdieloch medzi súbormi je séria riadkov, ktoré boli odstránené, vložené do súboru, alebo zmenené v súbore[5]. V takomto prípade existuje algoritmus, ktorý porovnáva dva súbory po riadkoch a vyhľadáva skupiny riadkov, ktoré nie sú totožné v rámci súborov. Výsledkom je, že algoritmus vráti tieto odlišné skupiny. Po zistení, v čom sa odlišujú súbory je nutné staršie súbory zmeniť na súbor, ktorý bol aktuálne zmenený. V citovanom dokumente je uvedený aj iný pohľad na porovnávanie dvoch rôznych súborov a to za pomoci sekvencie bytov. Pohľad, ktorý je však pre potreby tejto aplikácie

[1] https://docs.google.com/

dôležitejší je ten prvý. Na základe tohto pohľadu totiž funguje algoritmus, ktorý popisuje Neil Fraser.

Neil Fraser[4] v citovanom dokumente uvádza, že diferenciálnu synchronizáciu je možné chápať ako komplexný problém. Vôbec teda nie je jednoduché mať viacero kópií súborov neustále správne zosynchronizovaných. V tomto dokumente ďalej popisuje aj najbežnejšie prístupy k synchronizácii. Stanovil, že týmito prístupmi sú vlastníctvo, odovzdávanie udalostí a trojcestné zlúčenie. Tieto prístupy sú považované za koncepčne jednoduché, ale majú aj svoje nevýhody.

2.1.2 Základné prístupy

Zamykanie

Je jednou z najjednoduchších techník. V najjednoduchšej forme je možné zdieľaný dokument upravovať práve jedným užívateľom v danej chvíli. Možnosť zamykania teda neponúka možnosť kolaborácie v reálnom čase medzi viacerými užívateľmi.

Predávanie udalostí

Patrí takisto k jednoduchým technikám. Spolieha sa na to, že zachytí všetky akcie používateľov a následne ich dokáže odzrkadlovať po sieti k ostatným užívateľom. Praktickou výzvou pri synchronizácii predávania udalostí je, že je nutné zachytiť všetky akcie používateľa. Samozrejmosťou je písanie, ale musia sa zachytiť aj úpravy ako napríklad vystrihnutie, vloženie, ťahanie a pustenie, nahradenie či automatická korekcia.

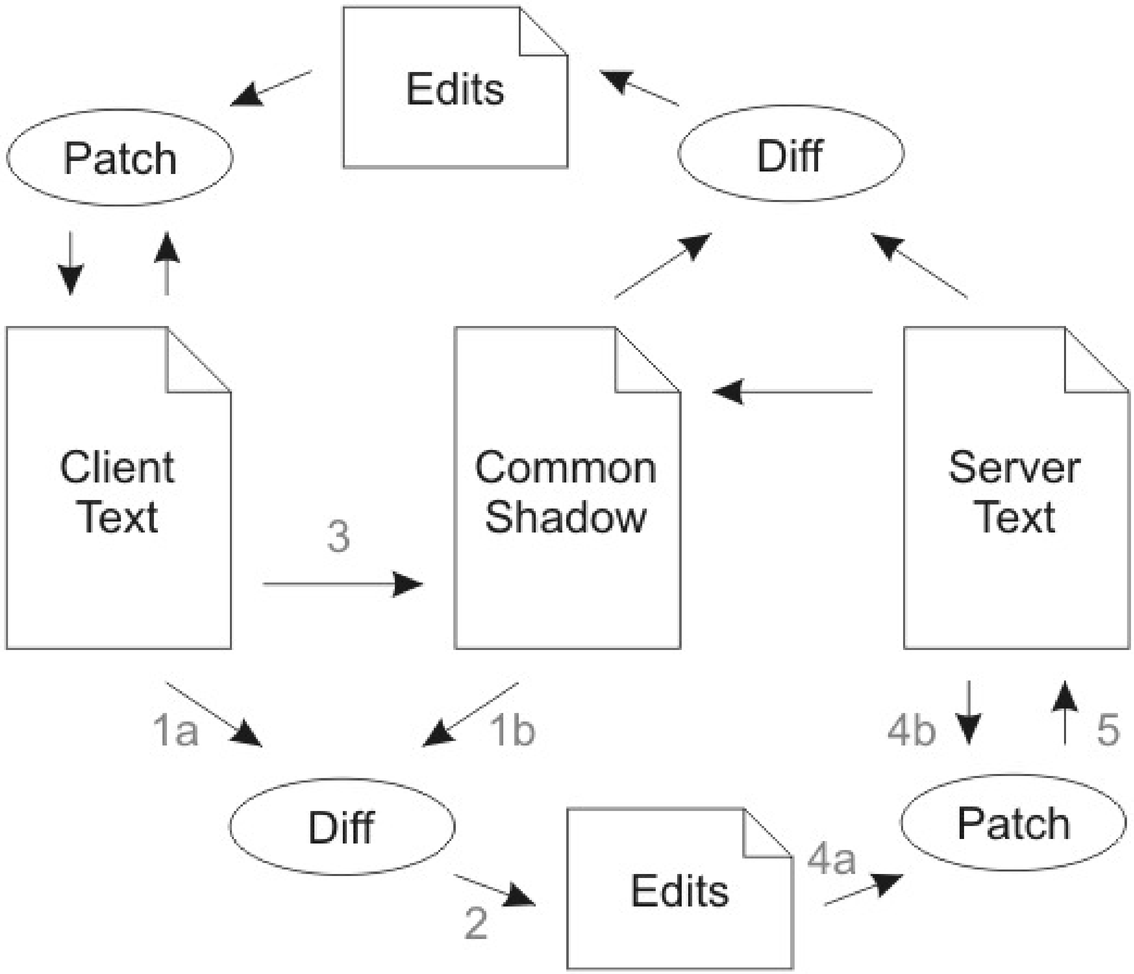
Pri tejto metóde môže pomerne ľahko nastať problém. Akékoľvek zlyhanie počas odovzdávania úprav má totiž za následok fork. Jedna stratená úprava môže spôsobiť nesprávne použitie iných úprav. Následne dochádza k problému, pretože sa zvýši rozdiel medzi dvomi verziami súborov. Vo chvíli, keď sa paket stratí, alebo oneskorí je potrebné aby sa systém dokázal rýchlo a efektívne zotaviť.

Trojcestné zlúčenie

Funguje na takomto princípe. Prvotne klient pošle obsah súboru na server. Potom server vykoná trojcestné zlúčenie, aby aplikoval zmeny užívateľa spolu so zmenami od iných užívateľov. Server odošle klientovi novú kópiu súboru. Ak používateľ vykonal v dokumente nejakú zmenu v čase, kedy prebiehala synchronizácia je klient nútený zahodiť novo prijatú zmenu a skúsiť to znovu neskôr. Tento systém autor označuje ako polovične duplexný. To znamená, že pokiaľ niekto píše, žiadne zmeny neprichádzajú. Krátko po ukončení písania sa však zobrazí buď vstup od ostatných užívateľov, alebo dialógové okno s informáciou o kolízii. Autor ďalej prirovnáva takýto systém k čelnému sklu automobilu, ktoré sa stáva nepriehľadným počas jazdy. Ak by mal každý na ceste totiž takéto nepriehľadné sklo, tak na cestách by sa kolízie takisto stávali bežnou záležitosťou. Trojcestné zlúčenie aj preto nie je vhodným riešením pre spoluprácu v reálnom čase po sieti s časovou odozvou.

2.1.3 Diferenciálna synchronizácia

V prípadoch, kedy dochádza k obojsmernému zdieľaniu je nutné, aby klientovi, ktorý zmenil súbor bol vytvorený rozdiel "diff"(z anglického slova difference). Jedná sa o rozdiel medzi tieňovým súborom, ktorý má klient dostupný pred úpravou so súborom, ktorý aktuálne upravuje. Následne je nutné upraviť tieňový súbor na aktuálny. Tento rozdiel potom klient posiela na server, kde je takisto nutné zmeniť súbor, aby platilo, že na každom zariadení v jednom momente musí byť dostupný aktuálny súbor.



Obr. 2.1: Ukážka idealizovaného data flow diagramu diferenciálnej synchronizácie. Predpokladá dva dokumenty uložené na jednom počítači bez využitia siete. (prevzaté z [4])

Všetky podobné príklady, akým je obrázok 2.1, ktoré popisuje Neil Fraser v dokumente, prezentujú synchronizáciu textu. Diferenciálna synchronizácia však dokáže spracovať akýkoľvek obsah (obyčajný text, formátovaný text, bitmapy, vektorovú grafiku a podobne). To platí vtedy, ak je pre daný obsah dostupný diferenčný algoritmus a fuzzy patch algoritmus.

Algoritmy tohto typu sú výpočtovo náročné. Pravdou však je, že zlepšovaním ich efektívnosti sa vo veľkej miere znižuje odozva systému. Zlepšením ich presnosti sa docieli menší počet kolízii a závažnosť kolízii.

Na účely diferenciálnej synchronizácie v rámci manažérov balíčkov npm, existuje napríklad veľmi dobre spracovaný diff-match-patch[2], ktorý je veľmi obľúbený. Obsahuje jednoduchú dokumentáciu a takisto je aj pomerne jednoduchý na použitie. Zároveň tento modul je implementovaný samotným Neilom Fraserom z Googlu, ktorého teória je popisovaná v tejto sekcii.

[2] https://www.npmjs.com/package/diff-match-patch

2.2 Protokoly pre komunikáciu v reálnom čase

Ako bolo spomínané v úvode, v súčasnej dobe internet zažíva stále väčšiu a väčšiu popularitu, a to v zastúpení všetkých vekových kategórií. Odpoveďou, prečo to tak je, môže byť aj to, že zdieľanie dát prebieha často v reálnom čase. Tie najpopulárnejšie webové aplikácie spája často práve komunikácia v reálnom čase. Ľudia vo veľkom navštevujú rôzne aplikácie slúžiace na textovú, či multimediálnu komunikáciu. V hojnom počte hrávajú online hry, ale na obrovskom vzostupe je aj veľké množstvo streamovacích služieb. Zo spomínaných vekových kategórií však najväčšie percento ľudí využívajucich internet patrí práve do kategórie mladých a mladistvých. Podľa štatistík z USA[6] dokonca 99% dospelých vo veku 18-29 sú užívateľmi internetu. Vďaka tomuto faktu je vhodné edukovať študentov aj prostredníctvom dostupných internetových aplikácii, keďže sú pravidelnými návštevníkmi webu.

Na webových stránkach alebo vo webových aplikáciách sú používatelia samozrejme zvyknutí, že pokiaľ na danej stránke, alebo v aplikácii urobia nejakú akciu, vyvolajú tým nejakú reakciu. Napríklad po kliknutí na tlačidlo odoslať je možné odoslať nejaký vyplnený formulár. Na mnohých stránkach alebo aplikáciách však dochádza často k situáciám, kedy nie je nutné vykonať žiadnu akciu, ale bez nej dostaneme nejakú odpoveď. Napríklad pokiaľ sa používateľ nachádza na stránke ako twitter či facebook a nebude dlhšie vykonávať na stránke žiadne akcie je možné, že sa mu zobrazí nejaké nové upozornenie. Tento príklad ukazuje, čo vo všeobecnosti znamená ak hovoríme o “real-time webe”.[13] Takýchto príkladov však môže byť oveľa viac. Všetko sa to deje preto, aby užívateľ nemusel neustále obnovovať stránku, prípadne odosielať, či sťahovať nejaké dáta manuálne pomocou nejakého tlačítka, či príkazu. Tým sa zaistí, že užívateľ vníma to, čo vykoná iný užívateľ v danej aplikácii okamžite aj na svojom zariadení.

Pre predstavenie protokolu Websocket je dobré vysvetliť aj protokol HTTP, pretože ako sa internet vyvíjal, tak sa postupne časom vyvíjali aj jednotlivé internetové protokoly. O vznik Websocket protokolu sa zaslúžil Ajax a jeho popularita v rámci aktualizácii v reálnom čase. Pri HTTP protokole klient požaduje od serveru zdroje, ktoré mu server buď poskytne, alebo ak nastane nejaká chyba pri komunikácii odošle príslušný chybový kód. Jednosmernosť HTTP protokolu bola vyriešená za pomocí technológií akými sú Comet, či long polling, ale výpočetné nároky pri týchto technikách boli vysoké[7]. Popis protokolov v tejto častí čerpá z RFC 2616[3], ktorý popisuje HTTP protokol, ďalej z RFC 6202[8], ktorý popisuje obojsmerné HTTP pomocou pollingu, či streamingu a RFC 6455[2], ktorý sa venuju protokolu Websocket.

2.2.1 Protokol HTTP

Hypertextový prenosový protokol (HTTP) je protokol aplikačnej vrstvy, nazývaný aj bezstavový, keďže je založený na princípe žiadostí a odpovedí. Spojenie vždy ustanovuje klient a to tak, že odošle serveru žiadosť. Server prijíma tieto spojenia a následne posiela odpovede. HTTP komunikácia prebieha zvyčajne prostredníctvom TCP spojenia. Predvoleným portom je port 80, ale je možné použiť aj iný port.

Komunikačný model HTTP polling

V štandardnom HTTP modeli server nemôže nadviazať spojenie s klientom, ani posielať nevyžiadanú odpoveď. Tým pádom, server nedokáže odosielať asynchrónne udalosti klientovi. Možnosťou, ako prijímať asynchrónne udalosti je, že sa server dopytuje v určitých časových intervaloch, aby získal nové dáta. Takéto neustále dopytovanie však vedie k zaťažaniu šírky pásma, posielaním požiadoviek vo chvíli, aj keď nie sú dostupné žiadne dáta. Takisto to môže byť neefektívne, keďže dáta ostávaju v poradí na odoslanie, pokiaľ server nedostane ďalšiu žiadosť od klienta.

Kvôli spomínaným problémom boli neskôr vytvorené takzvané "server push"mechanizmy, ktoré by mali byť efektívnejšie. Tieto mechanizmy fungujú na princípe, že povoľujú webovému serveru posielať aktualizácie klientovi bez toho, aby čakali na dopytovaciu požiadavku. Často sa súhrnne označujú názvom Comet. Comet mechanizmy majú nižšiu odozvu ako HTTP polling a takisto aktualizácie zo servera môžu prichádzať klientovi v častejšom rozsahu.

HTTP Long Polling

Jednou z techník, pomocou ktorej je možné zlepšiť efektívnosť a nároky na záťaž je aj HTTP Long Polling. Táto technika pracuje na princípe, že server sa snaží držať otvorený. Nesnaží sa teda o to, aby na každú HTTP požiadavku okamžite odpovedal, ale jeho snahou je odpovedať iba keď existujú udalosti, ktoré majú byť doručené. Tým pádom, vždy existuje nejaká čakajúca požiadavka, na ktorú server môže odpovedať za účelom doručenia udalostí, ak sa nejaké vyskytnú. Takýmto spôsobom dochádza k zníženiu oneskorenia pri doručovaní správy, ako aj k zníženiu využívania procesných/ sieťových zdrojov.

Streamovanie dát

Mechanizmus HTTP streaming nechává stále otvorené spojenie a nikdy neukončuje požiadavku. Tým pádom dochádza k veľkej redukcii odozvy, keďže klient so serverom nemusia otvárať a zatvárať spojenie medzi sebou.

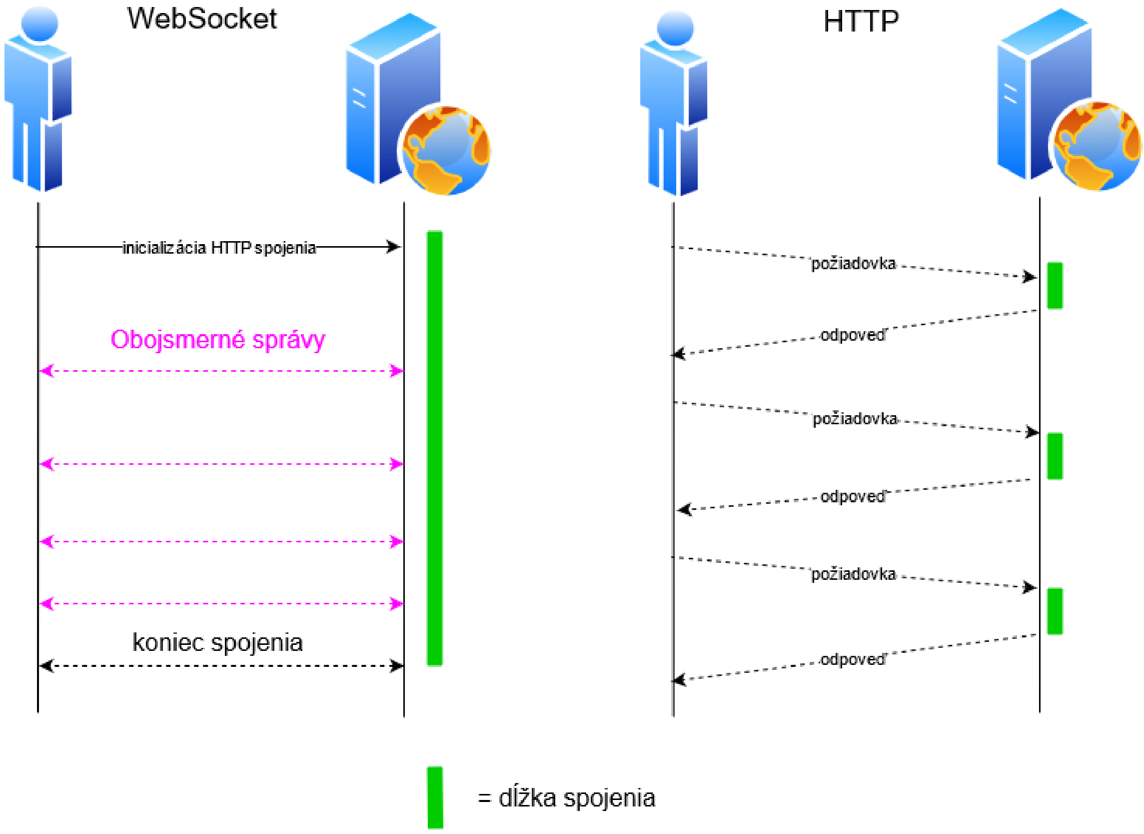
HTTP Streaming je možné popísať jednoduchým životným cyklom, kedy klient pošle úvodnu požiadavku. Server odloží odpoveď na túto požiadavku a čaká, pokým nedôjde k aktualizácii, alebo pokiaľ nenastane nejaký časový limit, prípadne určitý stav. Ak je k dispozícii nejaká aktualizácia, potom server posiela túto aktualizáciu späť klientovi, ako časť jeho odpovede. Dáta poslané serverom neukončujú spojenie, ani požiadavku.

2.2.2 WebSocket

Protokol Websocket je internetový komunikačný protokol, ktorý povoľuje obojsmernú, plne duplexnú komunikáciu medzi klientom a serverom. Historicky vznikol z dôvodu, že HTTP protokol nebol dostatočne dobrý z hľadiska réžie internetových aplikácií, ktoré potrebovali obojsmernú komunikáciu. Websocket teda ponúka oveľa jednoduchšie riešenie a používa jedno TCP spojenie pre posielanie dát v oboch smeroch. Websocket je nezávislým TCP protokolom. Jeho **jediným** vzťahom k **HTTP** je taký, že prvotný handshake je interpretovaný pomocou HTTP serverov ako aktualizačná požiadavka.

Zloženie protokolu pozostáva z dvoch častí: počiatočného podania rúk (handshake) a prenosu dát. V prípade, že obe strany klient a aj server odošlú svoj počiatočný handshake, ktorý bude úspešný, môže začať fáza posielania dát. Jedná sa o obojsmerný komunikačný kanál, v ktorom každá strana môže posielať dáta na želanie. Dizajn protokolu je postavený na princípe, že by mal obsahovať čo najmenej dátových rámcov. Konceptuálne je Websocket akousi vrstvou nad TCP.

Na obrázku 2.2, je možné si všimnúť krásne viditeľný rozdiel medzi jednotlivými dĺžkami spojenia. HTTP spojenie je otvorené iba chvíľkovo, kde naozaj odpovedá iba na požiadavku a potom sa zavrie. Opačne pri WebSocketoch je vidieť, že spojenie je perzistentné a môže trvať dovtedy, dokým sa jedna zo strán neodpojí.



Obr. 2.2: Porovnanie komunikácií HTTP a WebSocket

Socket.io

Socket.io je pomerne známa a populárna knižnica, ktorá povoľuje plne duplexnú obojsmernú komunikáciu medzi klientom a serverom v reálnom čase. Mnoho aplikácií využíva práve túto knižnicu pre účely komunikácie v reálnom čase. Princíp Socket.io je založený na protokole Websocket, ktorý je popísaný vyššie. Fungovanie je založené na naviazaní spojenia, kde sa klient pokúsi nadviazať spojenie cez WebSocket. Ak sa nadviazanie spojenia prostredníctvom WebSocket nepodarí, tak sa Socket.io snaží nadviazať spojenie pomocou iných prenosových mechanizmov, ako napríklad HTTP long polling[13]. Rovnako ako pre klientskú časť, tak aj pre serverovú časť obsahuje API. Obe API sú však takmer identické. Funkcie, ktoré poskytuje Socket.io navyše oproti bežným WebSocketom[14]:

- spoľahlivosť (spomínaný návrat k HTTP long pollingu v prípade, že nie je možné nadviazať spojenie z WebSocketom),
- automatické opätovné pripojenie,
- ukladanie paketov do vyrovnávacej pamäte (packet buffering),
- vysielanie pre všetkých klientov, alebo pre skupinu klientov. Takúto skupinu nazývame **miestnosť (room).**

Z vypísaných funkcií je vhodné doplniť nejaké informácie o miestnostiach. Podľa oficiálnej dokumentácie je miestnosť ľubovoľný kanál, ku ktorému sa jednotlivé sockety môžu pripojiť, alebo sa odpojiť. Miestnosti sú iba serverovým konceptom a teda klient nemá prístup k informáciam, ku akým miestnostiam je pripojený.

Po popise **Socket.io** je možné vidieť, že Socket.io nie je iba akousi implementáciou WebSocket protokolu, ale používa **WebSocket** pokiaľ ho **potrebuje**. Navyše pri komunikácii Socket.io pridáva do paketov dodatočné metadáta a preto, pokiaľ by sa niekto snažil pripojiť na Socket.io server prostredníctvom WebSocket klienta, nebolo by mu to umožnené. Takisto by to nefungovalo ani v opačnom prípade. Teda, že by sa niekto pokúšal pripojiť prostredníctov Socket.io klienta k čistému WebSocket serveru.

2.3 Existujúce riešenia

Cieľom riešenia práce je vytvoriť aplikáciu na zdieľanie obsahu poznámkových blokov, ktorá by dokázala podporiť výuku programovacích jazykov. V tejto sekcii sú preto predstavené podobné riešenia, či už na podporu výuky, alebo na zdieľanie poznámkových blokov. Pri aplikáciách určených na zdieľanie poznámkových blokoch je väčšia časť venovaná rozšíreniu LiveShare, keďže sa jedná o rozšírenie, ktoré je možné nainštalovať do Visual Studio Code.

Aplikácie na podporu výuky

Aplikácie na podporu výuky naberajú v posledných rokoch na popularite. Dovolím si dokonca tvrdiť, že v minulom roku využívanie takýchto aplikácií zažilo obrovský nárast na popularite a ich využívanie rapídne rástlo. Podľa môjho názoru totiž počas pandémie narástol počet využívania takýchto aplikácií, keďže prezenčná výučba nebola možná vo väčšine krajín sveta.

Tieto aplikácie však nemusia pomáhať iba pri dištančnej výučbe. Existuje možnosť si predstaviť, ako sa využívajú aj v prezenčných vyučovacích procesoch. Príkladom môžu byť aplikácie, ktoré momentálne podporujú zdieľanie zdrojových kódov v reálnom čase. Použitie by mohlo vyzerať tak, že učiteľ začne zdieľať kód, ktorý práve píše. Prístup k tomuto kódu by mal potom každý študent, či už v danej učebni, alebo z pohodlia domova. Napríklad aj v prípade, že by bol daný študent chorý, či nedostupný z iných dôvodov. Ak by navyše bol spolu so zdieľaným súborom streamovaný učiteľov hlas, vysvetľujúci daný kód, mal by kdekoľvek študent prístup k vyučovaniu s pocitom, akoby na danom vyučovaní bol prezenčne.

Samozrejme existuje možnosť zdieľania kompletného videa učiteľa ako vyučuje daný programovací jazyk. Možnosť, ktorá je spomínaná v predchádzajúcom odstavci je však v rámci technického vybavenia jednoduchšia na realizáciu. Navyše učiteľa, školu, či vzdelávací inštitút nestojí takmer žiadne peniaze, okrem samotného počítača a prípadnej ceny danej aplikácie.

Ďalšia vec, ktorú je vhodné spomenúť je, že toto nie je jediné využitie takýchto aplikácií. Existuje veľké množstvo aplikácií, ktoré úplne odlišným spôsobom podporujú výuku a sú populárne. Možno najznámejšou aplikáciou na podporu edukácie je Duolingo[3]. Jedná sa o veľmi populárnu aplikáciu a to najmä pre svoju zaujímavú, dalo by sa povedať, až hravú formu. Obsahuje aj totiž viaceré gamifikačné prvky. Takisto existuje Duolingo for Schools[4],

[3]Duolingo: https://www.duolingo.com/

[4]Duolingo for schools: https://schools.duolingo.com/

ktoré ponúka možnosť využívať duolingo v rámci jednotlivých tried. Učiteľovi tak ponúka väčšie množstvo možností ako sledovať progres jednotlivých žiakov.

Aplikácie určené na zdieľanie poznámkových blokov

Samotný poznámkový blok je popísaný nižšie. Tento typ súborov je však veľmi obľúbený, či už pri data-science odboroch, alebo aj pri učení programovacích jazykov. Existujú viaceré aplikácie, ktoré ponúkajú možnosť zdieľať obsah poznámkových blokov prostredníctvom internetu v reálnom čase. Jednou zo známych aplikácii je napríklad CoCalc[5], ktorá obsahuje aj rôzne pomôcky pre učiteľa.

Spomínaná aplikácia CoCalc má mnoho výhod a je určite vhodné ju používať. Je však nutné podotknúť, že sa jedná možno zbytočne o ďalšiu aplikáciu využívanú ako učiteľmi, tak aj žiakmi. Ľudia študujúci počítačové odbory a najmä teda programovanie obvykle používajú na písanie zdrojových kódov rôzne editory zdrojových súborov. Visual Studio Code je podľa viacerých zdrojov najpoužívanejší editor zdrojových súborov v súčasnosti. Spomínaný editor poskytuje momentálne možnosť zdieľať poznámkové bloky. Túto možnosť ponúka doplnok v rámci programu Visual Studio Code nazývaný LiveShare, ktorý je určený na zdieľanie obsahu. Je dôležité podotknúť, že LiveShare[6] takisto začal ponúkať možnosť zdieľať poznámkové bloky až v dobe písania tejto práce. Spomínaný LiveShare navyše síce túto možnosť poskytuje, ale táto funkcionalita nie je úplne hotová, vzhľadom na stále prebiehajúci vývoj Notebook API pre Visual Studio Code. Nutné **prerekvizity** pre zdieľanie poznámkových blokov rozšírením LiveShare sú:

- nutnosť využívania Visual Studio Code Insiders,
- nainšlavoné rozšírenie LiveShare v určitej verzii,
- u zdieľateľa rovnako aj nainštalované rozšírenie Jupyter.

LiveShare má stále aj známe problémy, kedy napríklad občasne dochádza k tomu, že pridávanie, mazanie, či presúvanie buniek nie sú synchronizované. Ďalej LiveShare nie je až tak jednoduchý na používanie, ako by si možno študenti predstavovali. Mojím názorom je, že takáto aplikácia by mala byť jednoduchá a dostupná každému bez nutnosti registrácie, či prihlasovania sa.

[5] Cocalc: https://cocalc.com/

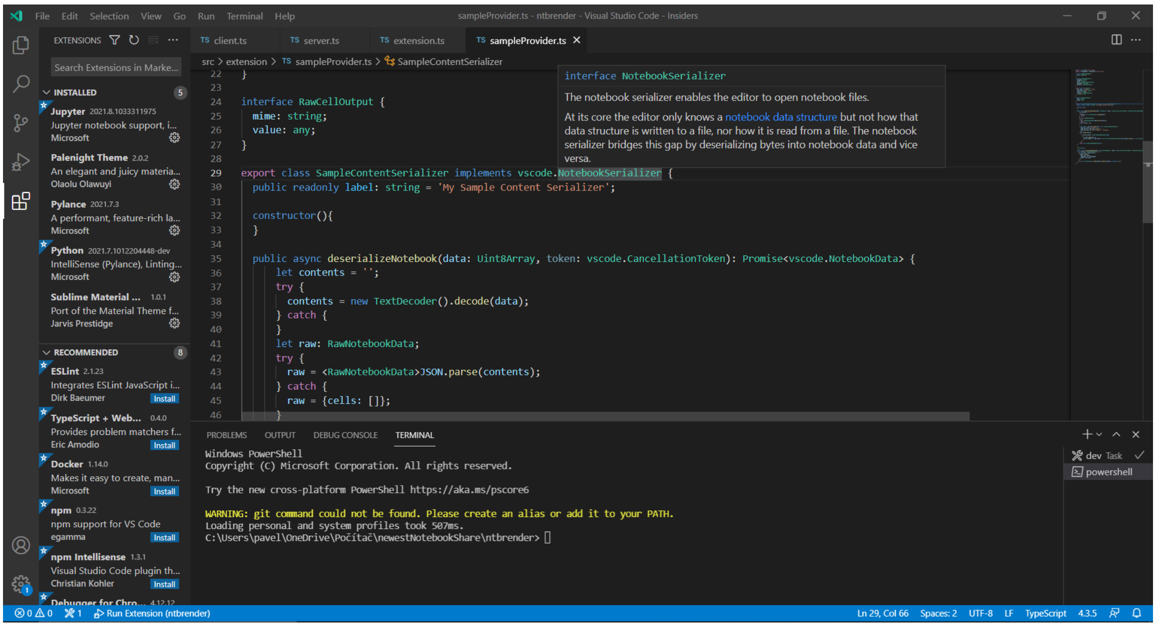
[6] LiveShare notebooks: https://docs.microsoft.com/en-us/visualstudio/liveshare/reference/notebooks

Kapitola 3

Visual Studio Code

V tejto kapitole je predstavený samotný editor **Visual Studio Code**, jeho základné funkcie a spôsob tvorby rozšírení v rámci tohto editoru. Pri spôsobe tvorby rozšírení je väčšia časť venovaná **Notebook API**. Toto API povoľuje v rámci tvorby VS Code rozšírení využívať funkcie poznámkových blokov. V tejto časti bolo dôležité predstaviť aj samotný poznámkový blok. Posledná časť sa zaoberá popisom jazykov **JavaScript** a **TypeScript**. Tieto jazyky totiž môžu byť využívané na tvorbu rozšírení v editore VS Code.

3.1 Editor Visual Studio Code



Obr. 3.1: Editor Visual Studio Code, konkrétne otvorená Insiders verzia.

Freewarový editor zdrojových kódov Visual Studio Code (ďalej VS Code), ktorý bol vyvinutý spoločnosťou Microsoft je jedným z najpoužívanejších editorov v súčasnosti. Samotný VS Code je pomerne novým editorom, keďže jeho prvé vydanie vyšlo až koncom roka 2015. Je však veľmi populárny, čoho znakom je aj výsledok prieskumu Stack Overflow Developer Survey[15] z roku 2019. Už po štyroch rokoch od svojho prvého vydania bol v tomto prieskume vyhodnotený za **najpopulárnejšie** vývojárske prostredie. VS Code je dostupný ako pre Windows, tak aj pre MacOS a Linux. Tento editor ponúka zabudovanú podporu jazykov JavaScript, TypeScript a Node.js a má bohatý ekosystém rozšírení pre iné jazyky ako napríklad (C#, C++, JAVA, Python)[9].

VS Code kombinuje jednoduchosť editora zdrojového kódu s výkonnými vývojárskymi nástrojmi, ako napríklad IntelliSense, teda doplnenie a ladenie kódu. VS Code ponúka samozrejme aj funkcie podporujúce zvýrazňovanie syntaxu, párovanie zátvoriek, automatické odsadenie, výber políčok a mnohé ďalšie[11].

VS Code je teda ideálne vývojové prostredie pre implementáciu rozšírenia na zdieľanie obsahu poznámkových blokov v reálnom čase.

3.2 Tvorba rozšírení pre VS Code

Základnou myšlienkou pri tvorbe VS Code bolo, aby bol čo možno najviac rozšíriteľný[10]. Vývojárom sa to nakoniec podarilo zrealizovať. VS Code tak ponúka možnosť vytvoriť vlastné rozšírenie prakticky každému prostredníctvom ich Extension API. Pomocou tohto API je možné upravovať, alebo prispôsobiť si VS Code rôznymi spôsobmi. Podľa samotného VS Code tímu je v rámci tohto editoru mnoho základných funkcií vytvorených práve za pomoci Extension API. To je možné potvrdiť, keďže mnoho základných funkcií nemusí byť dostupných pokiaľ ich nepotrebujeme, ale je nutné ich nainštalovať prostredníctvom rozšírení priamo vo VS Code. Dôkazom je aj to, že medzi najpopulárnejšie rozšírenia patrí aj rozšírenie s názvom Python, ktoré rozširuje VS Code o podporu tohto jazyka. Inštalácia je však veľmi jednoduchá a užívateľa vôbec neobmedzuje. Po inštalácii rozšírenia je možné rozšírenie kedykoľvek aktivovať alebo deaktivovať. Rozšírenia ponúkajú viacero možností využitia.

K samotnému vygenerovaniu projektu pre tvorbu rozšírenia je potrebné, aby programátor mal nainštalovaný Node.js, Git, Yeoman a samotný VS Code Extension Generator, ktorý je možné takýmto spôsobom nainštalovať, *npm install -g yo generator-code*. Generátorom je potom možné vytvárať projekty a to zadaním yo code do terminálu. Po zadaní tohto príkazu je ďalej potrebné vyplniť rôzne políčka, ako napríklad výber programovacieho jazyka (TypeScript, JavaScript), názov projektu, identifikátor rozšírenia, či chceme inicializovať git repozitár, alebo akého správcu balíčkov používať pre daný projekt.

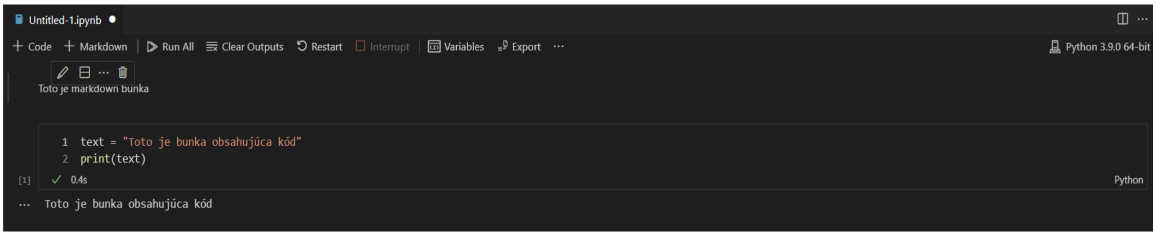
Po vytvorení rozšírenia je možné pracovať na rozšírení a využívať popritom funkcie programovacieho jazyka, dostupné knižnice, ale aj rôzne schopnosti, ktoré ponúka samotné Extension API. Spomínaných schopností, ktoré sú ponúkané je veľa. Základnými sú napríklad registrovanie nových príkazov, konfigurácií, klávesových skratiek, či položiek kontextového menu.

Extension Host(hostiteľ rozšírenia)[10] je správcom všetkých aktúalne bežiacich rozšírení. Jedná sa o node.js proces, ktorý je vo VS Code zodpovedný za načítanie a spustenie rozšírení. **Stabilita a výkon** je hlavným cieľom editora VS Code. Nesprávne fungujúce rozšírenia by nemali mať vplyv na užívateľskú skúsenosť. Hostiteľ rozšírení zabraňuje rozšíreniam v oplyvnení výkonu pri spustení, spomalení operácií užívateľského rozhrania, alebo pri úpravách používateľského rozhrania. VS Code navyše umožňuje rozšíreniam zadefinovať ich aktivačné udalosti a načítava ich lenivo. Napríklad v našom prípade sa rozšírenie aktivuje na príkaz o zdieľanie, alebo na príkaz o pripojenie. Týmto pádom je zaistené, že rozšírenia nespotrebúvajú zbytočné CPU, či pamäť.

3.3 API pre podporu natívnych poznámkových blokov

Pred predstavením samotného notebook API je vhodné zadefinovať, čo je vlastne notebook (ďalej poznámkovy blok).

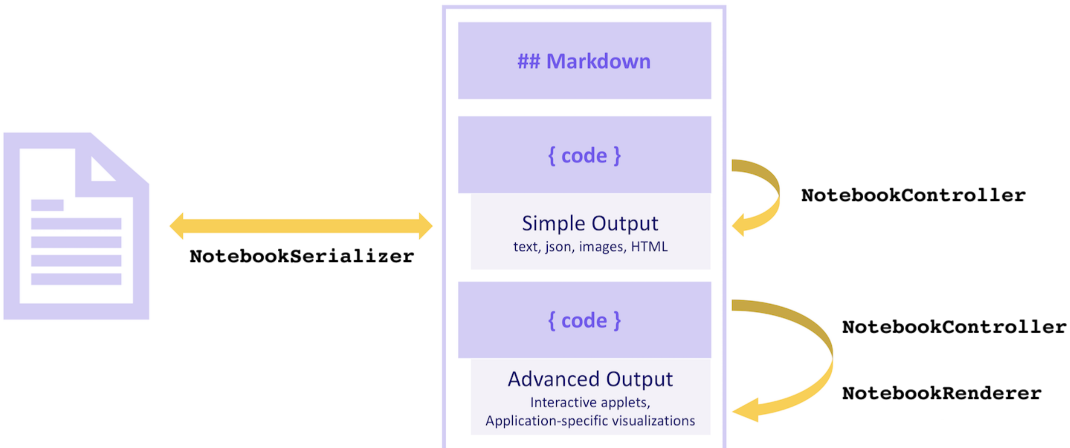
Poznámkový blok



Obr. 3.2: Príklad jednoduchého poznámkového bloku otvoreného vo VS Code

Poznámkový blok v rámci VS Code je v podstate **jupyter notebook**. Ten je definovaný ako open-source interaktívny webový nástroj, známy aj ako výpočtový, alebo poznámkový blok. Jupyter je vlastne dokument definovaný formátom JSON[1] a je ukladaný s príponou .ipynb. Spomínaný JSON súbor definuje poznámkový blok ako sekvenciu buniek a ich výstupov. V rámci poznámkového bloku sú bunky poznámkového bloku buď typu Markdown, alebo bunky obsahujúce zdrojový kód nejakého programovacieho jazyka.

Takéto typy blokov boli dostupné dlhé desaťročia. Jupyter však nabral v posledných rokoch veľkú obľubu. Jeho popularita doslova explodovala. Podľa dostupného zdroja[12] je možné tento nárast krásne vidieť. Zdroj totiž tvrdí, že podľa analýzy známej stránky na zdieľanie kódov GitHub, počet nazdieľaných súborov s príponou .ipynb na tomto portáli stúpol z okolo 200 000 v roku 2015 na 2,5 milióna do septembra roku 2018.



Obr. 3.3: Vizuálna ukážka Notebook API vo Visual Studio Code (prevzaté z https://code.visualstudio.com/api/extension-guides/notebook)

[1] Detaily JSON formátu pre poznámkový blok: https://nbformat.readthedocs.io/en/latest/format_description.html

3.3.1 Notebook API

Notebook API vo VS Code povoľuje rozšíreniam otvárať poznámkové bloky, ako aj vykonávať príkazy obsiahnuté v poznámkovom bloku a renderovať ich výstupy v rôznych formátoch. Notebook API dodáva do VS Code podobné využitia, rovnako ako mnoho iných rozhraní schopných pracovať s poznámkovými blokmi. Z takýchto rozhraní sú známe napríklad Jupyter notebook, či Google Colab. V rámci VS Code je Notebook API stále vo vývoji, takže je dostupné iba vo VS Code Insiders, čo je v podstate beta verzia Vs Code editoru. Tým pádom bola aj dokumentácia na oficiálnych stránkach mierne zastaralá a často neobsahovala najaktuálnejšie informácie.

Bunky poznámkového bloku sú čítané ako aj zapisované súborovým systémom pomocou NotebookSerializeru, ktorý riadi čítanie dát zo súborového systému a následné konvertovanie na popis daných buniek. Bunky poznámkového bloku je možné vykonať prostredníctvom NotebookControlleru, ktorý podľa obsahu jednotlivých buniek produkuje výstupy v rôznych formátoch od čistého textu až po formátované dokumenty, či interaktívne applety. Aplikačne špecifické formáty výstupov a interaktívne applety sú renderované za pomoci NotebookOutputRenderer. Tieto jednotlivé časti vizualizuje aj obrázok 3.3

NotebookSerializer

NotebookSerializer je svojim spôsobom veľmi prirovnateľný ku NotebookContentProvider. Dokonca, by sa dalo povedať, že sa jedná o akúsi jeho novú vylepšenú verziu. NotebookSerializer ponúka editoru možnosť otvárať súbory poznámkového bloku. Je definovaný ako rozhranie a ponúka dve funkcie. Jednou z nich je *deserializeNotebook*, ktorá konvertuje súbor poznámkového bloku na dátovú štruktúru s názvom NotebookData. S tou vie editor ďalej pracovať. Opačným prípadom je druhá funkcia *serializeNotebook*, ktorá otvorený poznámkový blok serializuje a dáta ukladá do súboru.

Spomínaná dátová štruktúra NotebookData je tvorená z buniek dát poznámkového bloku. Bunky dát sú tvorené pomocou triedy NotebookCellData, kde povinnými parametrami sú:

- **kind**: vyjadruje druh bunky, teda či sa jedná o bunku so zdrojovým kódom nejakého podporovaného programovacieho jazyka, alebo sa v bunke nachádza markdown text,
- **value**: označuje zdrojové hodnoty dát, teda buď sa jedná o zdrojový kód, alebo o markdown text,
- **langueageId**: identifikuje jazyk daných zdrojov dát v rámci bunky.

Nepovinné parametri sú:

- **outputs**: výstupy danej bunky. Bunka žiadny výstup mať nemusí, prípadne bunka ani po spustení nevráti žiadny výstup,
- **metadata**: jedná sa o metadáta danej bunky,
- **executionSummary**: jedná sa o sumár vykonania danej bunky.

NotebookController

NotebookController v rámci Notebook API má zodpovednosť pri spúšťaní daných buniek. Od tejto časti Notebook API závisí, či bunka má, prípadne nemá výstupy. NotebookController môže byť priamo spojený s NotebookSerializer. V tejto aplikácii však nie je nutnosť používať NotebookController, keďže už vyprodukované výstupy daných buniek budú prichádzať od zdieľateľa poznámkových blokov.

3.4 Jazyk TypeScript

Jazykom TypeScript rozumieme open-source programovací jazyk, ktorý bol vytvorený spoločnosťou Microsoft. Microsoft je rovnako tvorcom TypeScriptu a aj VS Code editoru. Je tak takmer jasné, že TypeScript má excelentnú podporu v rámci VS Code, ale aj v editore Visual studio, ktorý je takisto produktom Microsoftu. Rovnako tak je však tento jazyk podporovaný aj inými dostupnými editormi. Kód jazyka JavaScript je takmer vždy validným zdrojovým kódom jazyka TypeScript, až na veľmi málo výnimiek. Ak je do TypeScript súboru vložený kód jazyka JavaScript, všetky príkazy v danom súbore budú validné. Avšak môže nastať situácia, že kód bude plne funkčný, ale kompilátor jazyka TypeScript môže ukazovať rôzne varovania[1].

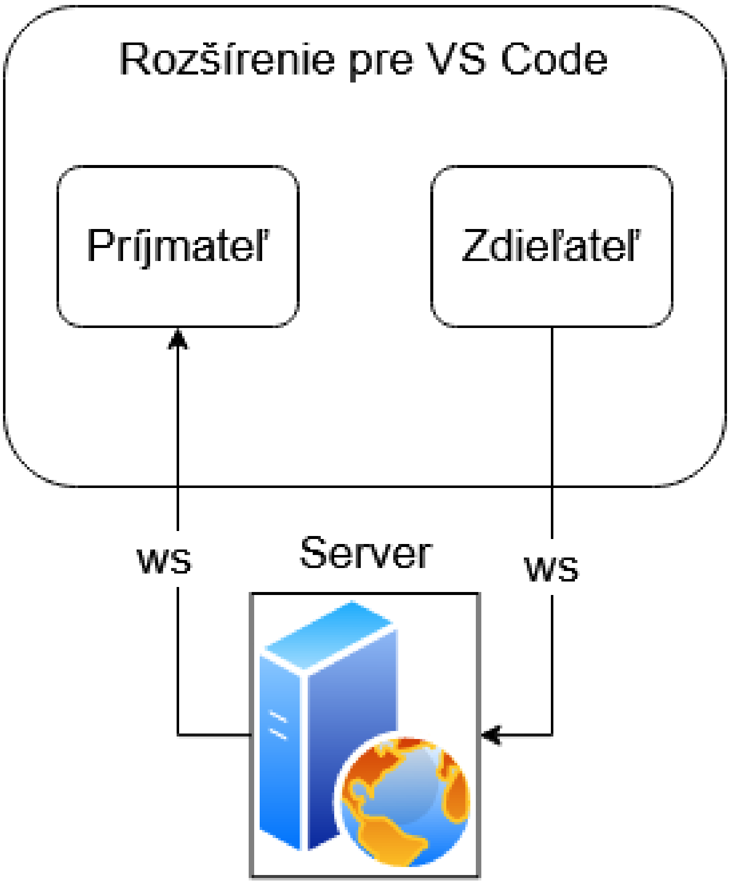
TypeScript je nadstavbou jazyka JavaScript. To znamená, že jazyk TypeScript poskytuje všetko, čo poskytuje JavaScript, s dodatočnými užitočnými funkciami navyše. Dôležitá vec z tohoto pohľadu je, že všetky štandardné kontrolné mechanizmy, ktoré ponúka JavaScript sú okamžite dostupné v TypeScript programe. Konkrétne dátove typy, operátory, riadiace štruktúry, podprogramy. Základné bloky programov, ako podmienky if, switch, cykly, funkcie, aritmetika a logické testy, teda pochádzajú z JavaScriptu a sú dobre známe širokej komunite programátorov. Podľa prieskumu, ktorý každoročne vydáva GitHub získava jazyk TypeScript čoraz väčšiu popularitu, aj keď najpopulárnejším programovacím jazykom zatiaľ ostáva stále JavaScript.[2]

[2] Octoverse, prieskum webovej služby GitHub https://octoverse.github.com/

Kapitola 4

Návrh riešenia

V tejto kapitole je popísaný návrh celého riešenia práce. Návrh je skomponovaný s úmyslom dodržať pravidlá, ktoré majú platiť pri zdieľaní obsahu v reálnom čase. Ako už bolo spomínané, tak jasným cieľom riešenia je dovoliť študentom mať na svojich zariadeniach aktuálnu kópiu poznámkového bloku učiteľa. Riešenie by mali využívať študenti a učitelia. Tým pádom je vhodné riešenie implementovať v rámci rozšírenia editoru VS Code, vďaka všetkým jeho výhodam.



Obr. 4.1: Diagram zobrazujúci Server a rozšírenie

Riešenie je vhodné implementovať ako rozšírenie pre VS Code aj z hľadiska toho, že VS Code je veľmi jednoduché nainštalovať, ako aj z hľadiska, že je naozaj extrémne populárnym editorom zdrojových súborov. Pravdepodobne je používaný veľkým percentom ľudí z cieľovej skupiny. Implementované riešenie pomocou rozšírenia, by zároveň malo byť jednoduché na inštaláciu. Rozšírenie bude možné vyhľadať v dostupných rozšíreniach v rámci VS Code, akonáhle bude Notebook Api finálne pridané do VS Code. Pokiaľ nebude dostupné, je jednoduché nainštalovať dané rozšírenie pomocou súboru s príponou .vsix, do ktorého je takéto rozšírenie "zabalené". Pokým však nebude Notebook API dostupné aj pre klasickú verziu VS Code, bude rozšírenie fungovať iba v jej beta verzii.

Návrh možno rozdeliť do troch skupín, ktorými sú server, rozšírenie pre VS Code a komunikácia medzi serverom a rozšírením. Tieto jednotlivé časti návrhu je možné vidieť na obrázku 4.1. Prvým bodom návrhu je samotná tvorba rozšírenia na zobrazovanie poznámkových blokov v rámci VS Code, teda dalo by sa povedať zjednodušený editor poznámkových blokov, prípadne prehliadač. Druhým bodom bude implementácia serveru, ktorý bude komunikovať spolu s rozšírením a spracovávať dáta prichádzajúce od učiteľa (zdieľajúceho) a posielať ich študentom (prijímateľom). Tretím bodom je vlastne klientská časť, ktorá je súčasťou rozšírenia a riadi posielanie a prijímanie dát zo servera a následnú prácu s týmito dátami. Celkový diagram zobrazujúci dátový tok medzi zdieľateľom, serverom a prijímateľom je zobrazený na obrázku 4.2

4.1 Jednoduchý prehliadač poznámkových blokov

Rozšírenie vo forme zjednodušeného prehliadača poznámkových blokov je jedným z bodov práce. Tento prehliadač vizualizuje práve zdieľaný obsah v reálnom čase. Prijímateľ totiž bez akejkoľvek jeho akcie dostáva stále nové a nové dáta, ktoré je nutné spracovať a vizualizovať na jeho zariadení. Táto časť návrhu práce je zameraná práve na návrh prehliadača poznámkových blokov pre prijímateľa.

Pomocou tohto rozšírenia je možné zobrazovať poznámkové bloky, ktoré s používateľom niekto zdieľa. Nejedná sa teda o kompletne ucelený editor, ale iba o nejaké časti. Spomínané časti serializujú daný poznámkový blok, ktorý je odosielaný ku prijímateľom od zdieľateľa. Prijímateľ tým pádom nepotrebuje mať na svojom zariadení Jupyter, prípadne iné rozšírenie pre prácu s poznámkovými blokmi. Treba však podotknúť, že naopak je žiadúce, aby zdieľateľ mal nainštalované vo svojom VS Code rozšírenie na prácu s poznámkovými blokmi. Možno teda spomenúť buď veľmi známy Jupyter, ktorý je zahrnutý priamo v rozšírení jazyka Python, alebo nejaké iné rozšírenia. Takýto prístup je z môjho osobného pohľadu vhodné zvoliť, keďže samotný Jupyter má nainštalovaný veľké množstvo užívateľov VS Code a preto je jednoduchšie zdieľať súbor otvorený za pomoci tohto rozšírenia.

Do veľkej miery funguje otváranie prichádzajúcich zdieľaných súborov aj za pomoci komponentu, ktorý ponúka priamo Notebook API. Jedná sa o Notebook Serializer, ktorému síce treba zadefinovať, ako súbor výzerá, čo obsahuje, prípadne môže obsahovať, ale na druhej strane po tomto zadefinovaní sa už o všetko stará Notebook API. Keďže súbory poznámkového bloku typu Jupyter sú vlastne súbory formátu **JSON**, akurát so špeciálnymi pravidlami, tak práve tieto pravidlá je nutné definovať.

Ako bolo spomínané tento prehliadač neplní plnohodnotnú funkciu editora. Prijímateľ je teda v pozícii, kedy mu iba prichádza nový súbor a jeho aktualizácie. Otvorený dokument v tomto prehliadači sa následne automaticky mení.

4.1.1 Operácie s poznámkovými blokmi

Keďže cieľom riešenia je dovoliť študentom mať na svojich zariadeniach aktuálnu kópiu poznámkového bloku svojho učiteľa, tak je určite vhodné, aby riešenie spĺňalo minimálne tieto požiadavky. Je potrebné danú kópiu poznámkového bloku dokázať otvoriť. Editácia smerom od študenta k učiteľovi nie je vôbec potrebná, keďže sa nejedná o tvorbu spolupráce na notebookoch. Určite by aj takúto funkciu v budúcnosti nebolo na škodu implementovať. Ďalej nemenej dôležitým je aj spúšťanie daných buniek kódu na zariadení študenta. Je určite vhodné, aby bolo dostupné v rámci riešenia, keby si študent mohol spustiť daný kód a overiť, či funguje správne. Keďže hlavná cieľová skupina je učiteľ a žiaci, tak táto možnosť nie je až tak potrebná. Predpokladá sa totiž, že učiteľ pri programovaní prednáša určitý výklad a v prípade nutnosti daný program sám spustí. Stačí teda, aby bolo možné zobrazovať výstupy smerujúce od učiteľa k študentovi.

Čo však určite stojí za pridanie do návrhu, je ukladanie samotných poznámkových blokov, aby si ich študenti mohli uložiť. Po uložení bude možné otvoriť tento súbor aj za pomoci Jupyter rozšírenia a tým pádom neskôr po skončení výučby bude žiakovi umožnené vidieť výstupy na základe vlastného spustenia buniek.

Na túto časť sú využívané technológie spomenuté v prvej kapitole a to konkrétne samotný VS Code, extension API, notebook API, jazyk TypeScript a potrebné knižnice tohto jazyka.

4.1.2 Editácia poznámkových blokov

Z hľadiska toho, že riešenie má byť určené najmä ako jednosmerné zdieľanie notebookov tak, aby bolo vhodné podporiť výuku, nie je potrebné podporovať obojsmerné zdieľanie. Naopak je práveže žiadúce aby jediný, kto môže upravovať poznámkový blok, bol učiteľ, keďže ho zdieľa s ostatnými a jeho hlavným cieľom je žiakom vysvetliť, čo možno najviac detailov.

Editácia otvorenej kópie poznámkového bloku podľa vlastnej potreby teda nie je až tak potrebná, keďže sa očakáva, že pri zdieľaní poznámkového bloku učiteľ so žiakmi komunikuje cez nejakú externú aplikáciu a komentuje, čo sa momentálne deje. Z tohto hľadiska nepovažujem túto možnosť za veľmi dôležitú a navyše študent si bude môcť poznámkový blok stiahnuť prakticky kedykoľvek a dopisovať si v ňom tak, aby bola réžia čo najmenšia.

4.1.3 Výstupy poznámkových blokov

Výstupy v poznámkových blokoch nie sú vždy k dispozícii. Výstup vznikne po spustení bunky, ktorá nejaký výstup vyprodukuje. V prípadoch, keď ešte bunka nebola spustená, alebo aj po spustení nemá výstup, výstupy neexistujú. V otvorenej kópii študenta, by mala byť možnosť vidieť jednotlivé výstupy buniek. Študent by tak samozrejme videl okamžitý výsledok daného problému prednášaného učiteľom. Preto je žiadúce opätovne využiť komponent Notebook API pre VS Code, ako aj knižnicu socket.io na posielania daných výstupov. Navrhnuté riešenie u študenta by malo samozrejme ponúkať rovnaké formáty výstupov, aké budú vytvorené na zariadení učiteľa. Pokiaľ nejakú bunku spustí učiteľ, ktorá má určitý výstup, tak by mal tento výstup byť prezentovaný rovnako ako učiteľovi, tak aj žiakovi. Keďže študentovi príde výstup po sieti od učiteľa, tak opäť nie je potrebné pre študenta mať k dispozícii spúšťanie jednotlivých buniek.

4.2 Zdieľanie dát

Predchádzajúca sekcia spomínaná v návrhu sa venovala najmä časti, kedy už sú dáta zdieľané a následne prijímané. Práve prijímanie a spracovanie dát boli popísané v predchádzajúcej sekcii. Táto sekcia sa venuje zdieľaniu dát a v podstate celkovému návrhu aplikácie z pohľadu zdieľateľa.

4.2.1 Začiatok zdieľania

Začiatok zdieľania začína užívateľ. Dôležité v rámci zdieľania je, aby mal zdieľateľ otvorený práve daný poznámkový blok. Začať zdieľanie môže užívateľ v akejkoľvek situácii pri otvorenom poznámkovom bloku. Je úplne jedno, či má užívateľ zatiaľ dokument prázdny, alebo už niečo daný dokument poznámkového bloku obsahuje. Aktuálnosť daného súboru rieši server. Server by mal mať neustále dostupný aktuálny súbor poznámkového bloku, aby vedel vždy pri novom pripojení na danú miestnosť poskytnúť poznámkový blok s rovnakým obsahom, aký práve má na svojom zariadení zdieľateľ daného poznámkového bloku.

Pri zdieľaní užívateľ zadáva takisto aj názov miestnosti, do ktorej daný poznámkový blok zdieľa. Pokiaľ miestnosť už existuje, znamená to, že už niekto zdieľa súbor v rámci danej miestnosti a je preto na mieste, aby bola zvolená iná miestnosť.

4.2.2 Zasielanie aktualizácií

Navrhnutie posielania aktualizácií v poznámkovom bloku nie je príliš komplikované. V prípadoch textových zmien, či už vloženia textu, alebo vymazania textu je pomerne jednoduché. Riešenie totiž spočíva v tom, že za pomoci diff-match-patch modulu spomínaného v prvej kapitole, v sekcí Synchronizácia obsahu, je vytvorený patch, ktorý je následne posielaný na server.

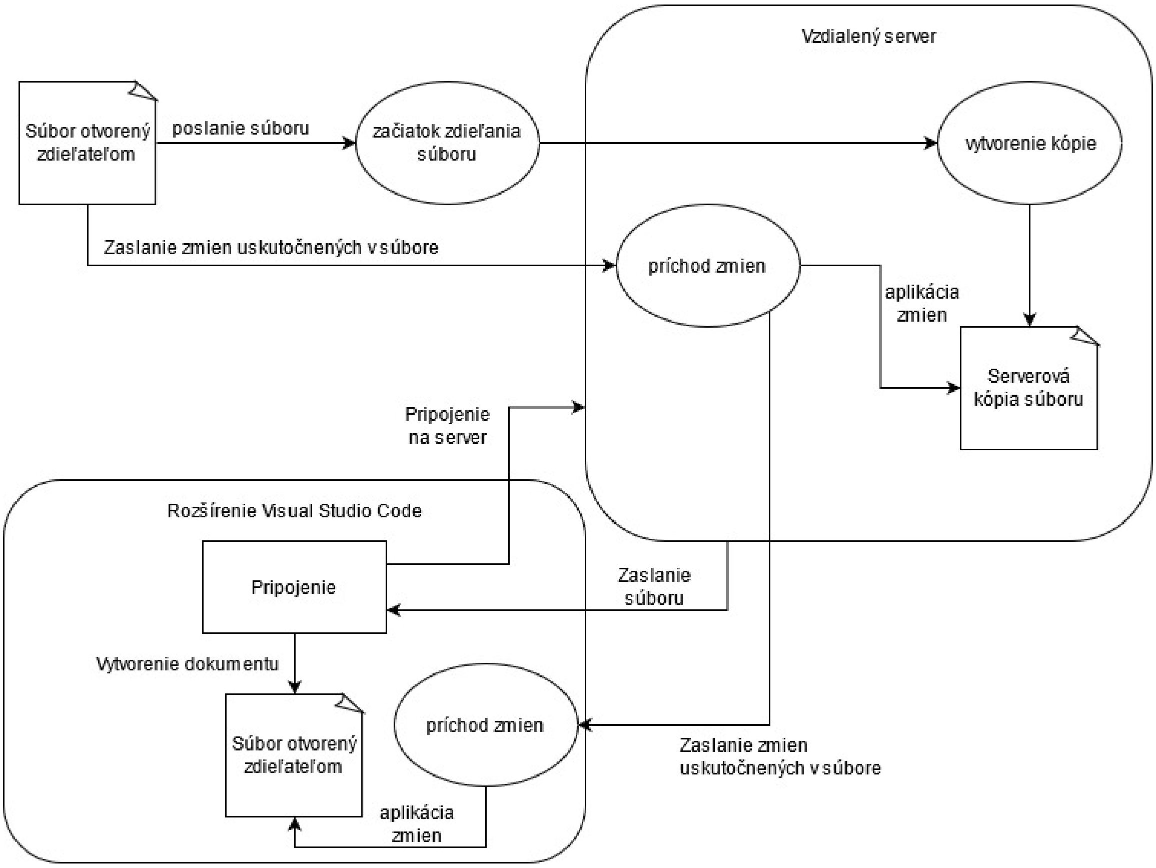
Zmeny však nie sú iba textového typu. V poznámkovom bloku je možné identifikovať dva typy buniek. Tieto typy buniek je možné odstrániť, pridať, či pohybovať s nimi. Preto aj pri týchto zmenách je nutné, posielať tieto zmeny na server. Server následne zmeny spracuje a pošle ich pripojeným klientom v danej miestnosti.

Všetky zmeny sú posielané, či už na server, alebo zo serveru za pomoci Socket.io knižnice.

4.3 Server a riešenie komunikácie

Server a komunikácia medzi klientami je veľmi dôležitou súčasťou riešenia, pretože sa zameriava na veľmi podstatnú časť zadania. Požiadavky na návrh sú jasné. Je nutné navrhnúť jednoduchý server s asynchrónnym pripojením tak, aby jeden odosielateľ zdieľal svoj poznámkový blok medzi určitú skupinu ľudí. Akonáhle je poznámkový blok zdieľaný a nejaká osoba chce tento poznámkový blok vidieť, tak po pripojení na server, by malo okamžite otvoriť zdieľaný poznámkový blok. Ako bolo spomínané vyššie, keďže sa jedná o jednosmerné zdieľanie tak v prípade, že majiteľ poznámkového bloku urobí nejakú zmenu, okamžite by táto zmena mala byť dostupná a viditeľná na všetkých zariadeniach, ktoré tento poznámkový blok prijímajú.

Táto požiadavka je riešená za pomoci technológií, akými sú **Socket.io** a **Node.js**. Node.js server čaká, pokiaľ mu nepríde socket od klienta. V ňom potvrdí, že chce zdielať dáta. Tento klient si však navyše zvolí názov miestnosti. Treba podotknúť, že názov miestnosti nikdy nie je náhodne vygenerovaný, ale vždy ho zadáva zdieľateľ. Od momentu, kedy klient úspešne zadal názov miestnosti je pripojený a začína zdieľať dáta. Tento zdieľateľ potom môže názov danej miestnosti poskytnúť svojím študentom, priateľom, skrátka každému, komu by chcel zdieľať daný poznámkový blok. Pokiaľ sa potom napojí na server klient, ktorý zadá rovnaký názov miestnosti, aký bol zadaný nejakým zdieľateľom, je pripojený a automaticky mu bude otvorený súbor. Pokiaľ by však niekto chcel začať zdieľať poznámkový blok do miestnosti, ktorá už existuje, tak mu to nebude umožnené. Pri takejto chybe bude nutné, aby došlo k zmene názvu danej miestnosti. Toto nastáva z dôvodu toho, že miestnosť je už obsadená a tak je potrebné vytvoriť novú miestnosť.



Obr. 4.2: Diagram zobrazujúci prepojenie a funkcionalitu popisovaných častí návrhu

Kapitola 5

Implementácia

Táto kapitola má za úlohu popísať implementáciu všetkých častí popísaných v návrhu. V tejto kapitole sú implementačné detaily jednotlivých častí navrhnutého riešenia popísané vo väčšej miere. Pri každej z jednotlivých častí je popísaný výber jazyka, knižníc a aj popis toho, čo sa nachádza v jednotlivých zdrojových súboroch. Projekt má dve implementačné časti. Rozšírenie pre VS Code a server, s ktorým rozšírenie komunikuje. Server využívajúci websockety je implementovaný pomocou node.js a socket.io knižnice.

5.1 Rozšírenie VS Code

Pre túto časť práce bola možnosť zvoliť buď jazyk JavaScript, alebo jazyk TypeScript. Táto časť práce je implementovaná za pomoci jazyka **TypeScript**, vďaka jeho výhodám. Výhody jazyka TypeScript oproti jazyku JavaScript sú spomínané v sekcí jazyk Typescript. TypeScript je navyše stále viac a viac populárnym jazykom. Jeho použitie je preto logickým krokom aj pre prípadné ďalšie rozširovanie aplikácie. Dôležitým prvkom pri implementácii je aj správca balíčkov **npm** a knižnica **socket.io**.

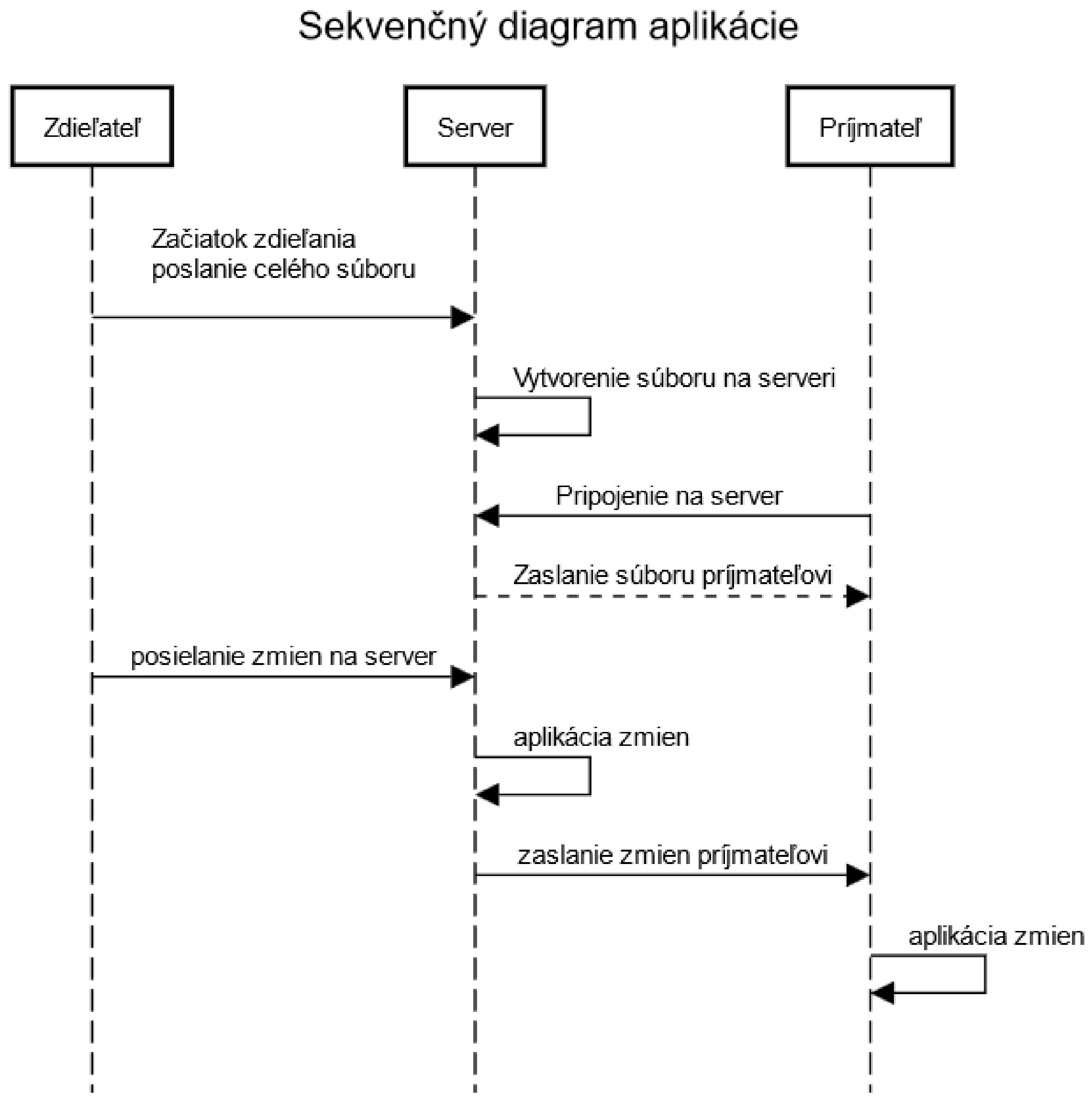
5.1.1 Štruktúra rozšírenia

Rozšírenia v rámci VS Code majú už vopred definovanú určitú štruktúru projektu. Tá vlastne definuje, v akých zložkách majú byť uložené jednotlivé súbory. Dôležitým súborom v hlavnej zložke projektu je súbor **package.json**, v ktorom sú definované závislosti na moduloch, skripty, rôzne príkazy a aktivačné udalosti. Ďalej je v tomto súbore uložený aj názov daného rozšírenia, verzia rozšírenia, verzia VS Code a podobne. Dôležité je aj povolenie v danom rozšírení využívať proposed API. Teda skupinu API, ktoré ešte neboli vydané do používania v hlavnom programe VS Code. A to najmä kvoli nestabilite a stálemu vývoju daného API. V hlavnej zložke je ešte možné nájsť tieto zložky:

- **.vscode**: táto zložka obsahuje súbory slúžiace na konfiguráciu projektu,
- **`node_modules`**: zložka, v ktorej sa nachádzajú jednotlivé moduly v rámci node.js, po akejkoľvek inštalácii modulu pomocou npm manažéra balíčkov sú tieto moduly pridané do tejto zložky,
- **src**: v tejto zložke sa nachádzajú jednotlivé zdrojové kódy. Najdôležitejším súborom v rámci tejto zložky, je súbor **extension.ts**, ktorý obsahuje aktivačné a deaktivačné metódy celého rozšírenia a teda dá sa povedať, že je vstupným súborom aplikácie,

- **out**: jednotlivé TypeScript súbory zo zložky src, sú kompilované na JavaScript súbory. Tieto JavaScript súbory sú ukladané práve do zložky out.

Extension.ts, nachádzajúci sa v zložke src obsahuje aktivačné udalosti, kedy sa má rozšírenie spustiť. K aktivovaniu daného rozšírenia dochádza pri zadaniach príkazov. Dôležitou časťou súboru extension.ts je v tomto projekte aj registrácia serializátora poznámkových blokov (notebookSerializer). Celkovo najdôležitejšou zložkou je zložka **src**. Práve do tejto zložky sú totiž ukladané všetky zdrojové kódy potrebné na vytvorenie rozšírenia.



Obr. 5.1: Sekvenčný diagram popisujúci flow medzi zdieľateľom, serverom a prijímateľom

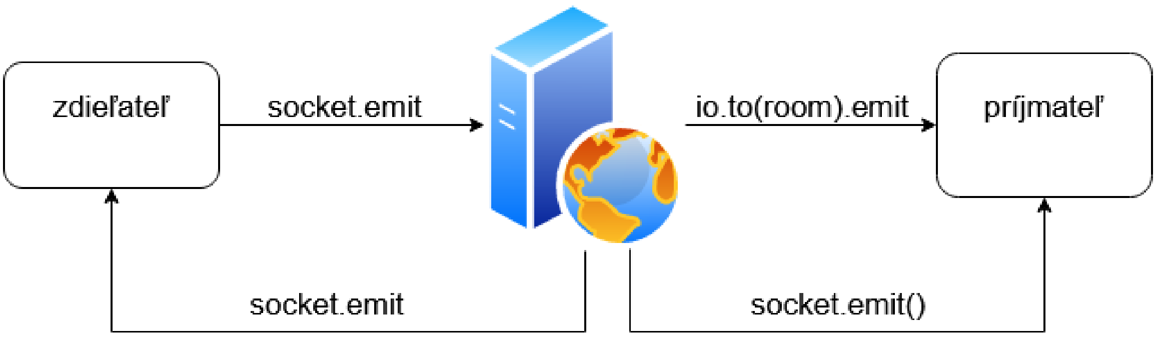
5.1.2 Prehliadač poznámkových blokov

Prehliadač poznámkových blokov slúži na zobrazovanie poznámkových blokov u prijímateľa. Jeho podstatou je teda prenášané zmeny od zdieľateľa aplikovať v podobe, aby vyzerali tak, akoby ich vykonával užívateľ priamo na svojom zariadení. Pre tento prehliadač je teda dôležité, aby mu boli doručené vždy všetky potrebné zmeny, ktoré vykonáva zdieľateľ. Všetky tieto zmeny a ich spôsob získavania, či aplikácie sú popísané v sekcii komunikácia medzi klientmi a serverom.

Dôležitou súčasťou prehliadača poznámkových blokov je **NotebookSerializer**. Ten definuje ako otvárať súbory poznámkového bloku. Vďaka využitiu tohto komponentu je potom jednoduchšie pracovať v rámci projektu s danými bunkami súborov. Ďalším benefitom je aj následne využívanie iných komponentov zo samotného Notebook API.

Ďalším nemenej dôležitým bodom, pri ktorom je využívaný **NotebookSerializer**, je ukladanie súborov poznámkového bloku. Takisto je možné zadefinovať, ako bude výsledný uložený súbor vyzerať. V rámci implementácie tohto projektu je však samozrejme potrebné, aby sa výsledný súbor čo najviac približoval originálu. Originálnym formátom je formát súboru ukladaný pomocou rozšírenia **Jupyter**. Pre zachovanie tohto formátu je využívaná metóda *serializeNotebook*, ktorá je súčasťou triedy NotebookSerializer. Pomocou tejto metódy sa určí, akým spôsobom bude daný JSON súbor uložený. Výsledný súbor obsahuje kľúč s názvom cells, ktorý má v sebe pole obsahujúce informácie o jednotlivých bunkách. Každá bunka obsahuje informácie o texte bunky, jej type a jej prípadných výstupoch.

Dôležitosť, čo možno najviac podobného formátu je potrebná z pohľadu znovuotvorenia daného súboru, za pomoci Jupyter rozšírenia. Jupyter je totiž plnohodnotný editor, ktorý je prepracovanejší a s dokumentom tak môže užívateľ v rámci Jupyter editoru robiť akékoľvek zmeny v dokumente poznámkového bloku.



Obr. 5.2: Štruktúra socket.io správ

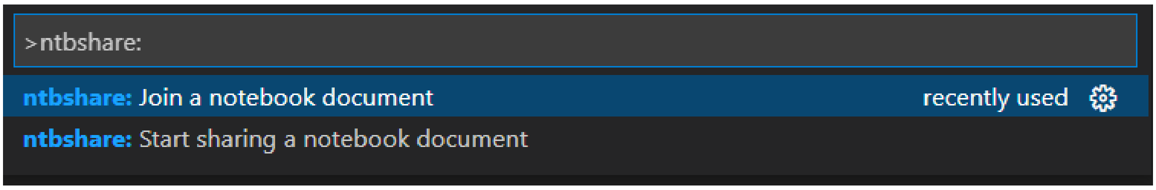
5.2 Komunikácia medzi klientmi a serverom

V tejto sekcii je popísaná komunikácia medzi zdieľateľom a prijímateľmi za pomoci serveru. Komunikácia je popísaná postupne, od začiatku zdieľania súboru a jeho prijatí, až po koniec zdieľania, tak ako to zobrazuje obrázok 5.1.

5.2.1 Začiatok zdieľania

Jednou z najdôležitejších častí implementácie je klient-server časť aplikácie. Pojmu klient môžeme rozumieť ako pripojené zariadenia na server, kde server spracováva komunikáciu medzi jednotlivými klientami. V prípade tejto aplikácie server spracováva a udržuje jednotlivé súbory poznámkových blokov stále aktuálnymi.

Na spustenie zdieľania je nutné, aby užívateľ zadal príkaz "ntbshare: Start sharing a notebook document", ktorý je možné zadať vo VS Code tak, ako to zobrazuje obrázok 5.3.



Obr. 5.3: Aktivačné príkazy daného rozšírenia

Po tomto príkaze sa otvorí užívateľovi okno, v ktorom zadá názov miestnosti, do ktorej chce zdieľať daný dokument. Po úspešnom zadaní je užívateľ pripojený na server a začne tak zdieľať svoj poznámkový blok do vytvorenej miestnosti s názvom, ktorý práve zadal. Na obrázku 5.2 je možné vidieť, že sa socket posiela aj zo servera zdieľateľovi. Jediný krát, kedy toto nastáva je práve pri pripájaní. Pri pripájaní sa totiž spätne posiela informácia, či už na serveri existuje daná miestnosť.

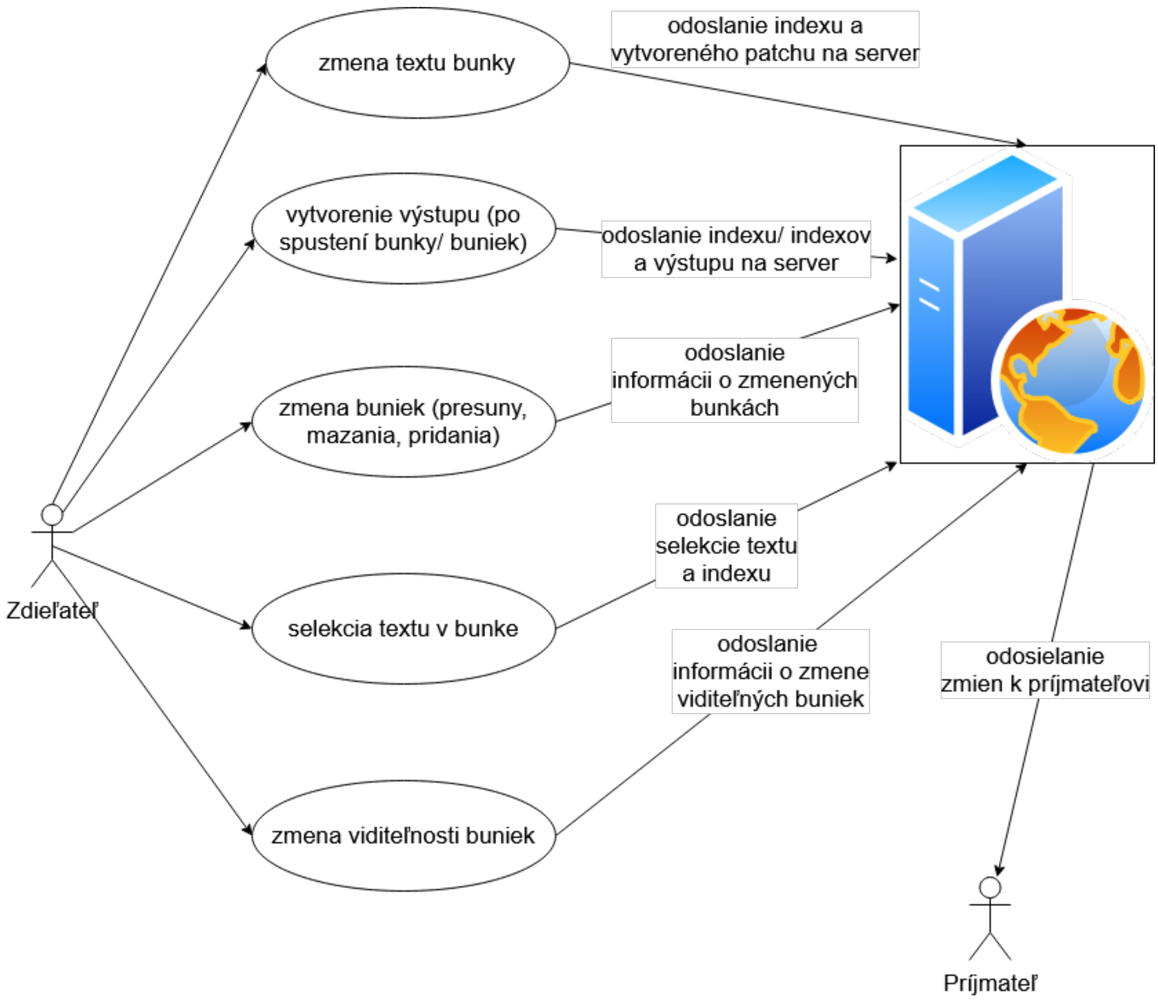
Začiatok celej komunikácie a následného zdieľania poznámkového bloku je implementovaný tak, že za pomoci forEach cyklu je prechádzaný celý aktuálne otvorený editor poznámkových blokov. Počas prechádzania jednotlivých buniek poznámkového bloku sú bunky postupne pridávané do premennej file, ktorá sa následne odošle na server. Ak by nastala situácia, kedy nie je aktúalne otvorený žiadny editor poznámkových blokov v prostredí VS Code, alebo je v rámci VS Code otvorených viac priečinkov, bude užívateľovi zobrazený error. Na začatie zdieľania je nutné mať otvorený práve daný súbor s poznámkovým blokom a najlepšie v samostatnom okne.

Server slúži v podstate iba ako správca súborov medzi zdieľateľom a prijímateľmi. Nároky na server v rámci tejto aplikácie nie sú vôbec vysoké. Server iba spracováva obsah a odosiela ho ďalej. Server spracováva obsah jednotlivých súborov v daných miestnostiach. Tieto súbory sú ukladané na serveri ako **JSON**. V ňom je kľúčom názov danej miestnosti. Hodnota pre daný kľúč je celý súbor, ktorý je uložený podobne ako **jupyter notebook**, teda takisto vo formáte JSON. Ten obsahuje dáta o jednotlivých bunkách. Najväčší dôraz pri implementácii serveru je v zachovaní aktuálnosti dokumentu na serveri. To znamená, že server musí byť schopný vždy urobiť takú zmenu v súbore, akú práve urobil zdieľateľ daného poznámkoveho bloku.

5.2.2 Pripojenie a otvorenie dokumentu u prijímateľa

Priebeh pripojenia na daný zdieľaný blok je tiež jednoduchý. Pripojenie prebieha po zadaní príkazu "ntbshare: Join a notebook document". Po zadaní príkazu sa užívateľovi zobrazí okno, kde zadá názov miestnosti, do ktorej sa chce pripojiť. Pri pripájaní na server si pomocou Socket.io, knižnice vymenia informáciu o tom, či daná miestnosť na serveri existuje. To je jediný krát, kedy server prijímateľovi posiela správu pomocou *socket.emit()*. Pokiaľ totiž miestnosť na serveri existuje, zavolá sa metóda *socket.join()*. Tá je využívaná iba na strane servera. Slúži na pridanie daného socketu do danej miestnosti. V praxi to znamená, že od tohto momentu nie je využívané volanie metódy *socket.emit*, ale *io.to("názov miestnosti").emit()*, kde sú dáta posielané iba socketom pripojeným do danej miestnosti. Týmto prístupom je vlastne vyriešené, že na jednom serveri sa môže nachádzať a byť obsluhovaných viacero súborov súčasne.

Po príchode dát zo serveru je daný dokument automaticky uložený do otvorenej zložky užívateľa. Súbor je vždy ukladaný s **názvom**, ktorý **vznikne spojením názvu miestnosti s anglickým slovom file**. Je teda málo pravdepodobné, že by nejaký užívateľ mal vo



Obr. 5.4: Odosielanie jednotlivých zmien na server vizuálne

svojej zložke súbor poznámkového bloku s takýmto názvom. Dáta, ako už bolo spomínané, prichádzajú vo formáte JSON. Dáta je potom veľmi ľahko možné spracovať pre otvorenie daného dokumentu.

Po úspešnom prijatí a uložení dát je možné otvárať dokument. Pri otváraní dokumentov na strane prijímateľa je využitý aj spomínaný prehliadač poznámkových blokov. Dáta, ktoré prichádzajú zo servera, je nutné za pomoci cyklu pretvoriť na JSON dokument, ktorý sa uloží a za pomoci prehliadača poznámkových blokov bude VS Code schopný otvoriť tento dokument. Daný JSON súbor je nutné uložiť tak, aby boli všetky bunky daného poznámkového bloku v rovnakom poradí, ako na zariadení zdieľateľa, alebo v súbore uloženom na serveri. Vďaka takémuto usporiadaniu je totiž jednoduchšie pracovať s týmito bunkami neskôr, či už pri príchode nejakej textovej zmeny do bunky, alebo pri pridávaní, mazaní, či presúvaní buniek.

5.2.3 Posielanie zmien urobených v súbore

Zmeny vykonané na poznámkovom bloku môžu byť dvoch typov. Buď sa jedná o textové zmeny v rámci danej bunky, alebo o iné zmeny týkajúce sa buniek a dokumentu poznámkového bloku. Pri každej zmene, ktorá sa uskutoční na strane prijímateľa je vždy využitá funkcia *vscode.window.showNotebookDocument()* s potrebnými parametrami, ktorá vždy presunie zameranie na tú bunku, pri ktorej došlo aktúalne k nejakej zmene. O tom, že má byť nejaká zmena odoslaná, informujú udalosti, ktoré sú súčasťou VS Code API. Pohľad na tieto zmeny je z hľadiska implementácie iný a jednotlivé pohľady sú popísané nižšie.

Popis protokolu

Pri počiatku zdieľania zdieľateľ odosiela prvotne celý súbor. Následne už odosiela iba zmeny urobené v dokumente. Z obrázku 5.4, je možné vidieť aké dáta sú posielané na server. Rovnaké udalosti potom server odosiela aj na pripojené zariadenia. Odosielaný obsah závisí vždy na udalosti, ktorú zdieľateľ vykonal. Zdieľateľ ešte navyše pri každej jednej zmene odosiela aj názov miestnosti. Server tak vie, do akej miestnosti má dané zmeny poslať.

- **zmena textu bunky**: pri zmene textu bunky je odosielaný na server patch, obsahujúci informácie o zmenách textu a index,
- **zmena výstupu bunky**: po zmene výstupu bunky sa na server odosiela index, a informácie o danej bunke. Jedná sa o informácie: druh, výstup a text bunky.
- **zmena buniek (mazania, pridania, presuvánia)**: odosielanie informácii o bunkách a prípadne počtu vymazaných buniek,
- **selekcia textu v bunke**: odosiela súradnice označeného textu v bunke,
- **zmena viditeľnosti buniek**: odosiela sa rozmedzie, ktoré bunky vidí na zariadení zdieľateľ.

Textové zmeny v rámci bunky

V súčasnosti, kedy je v rámci programovacích jazykov dostupných mnoho nápomocných knižníc, či modulov, je častokrát jednoduchšie nájsť riešenie na implementáciu, medzi dostupnými riešeniami. Vymýšľať totiž vlastnú implementáciu nejakého riešenia, ktoré už funguje správne a ktoré by inak bolo veľmi komplikované na implementáciu je v mnohých prípadoch zbytočné. Príkladom v rámci takejto pomoci je určite aj algoritmus Diff-Match-Patch spomínaný v kapitole Synchronizácia súborov. Algoritmus je používaný, aby vytvoril patch. K vytvorenie takéhoto patchu dochádza vždy pri zmene textu bunky na strane zdieľateľa. Informácia, kedy patch vytvoriť je zachytená pomocou udalosti *onDidChangeTextDocument()*. Táto udalosť informuje o akejkoľvek zmene obsahu v textovom dokumente. Bunky poznámkového bloku v rámci VS Code sú totiž reprezentované ako textové dokumenty. Následne po vytvorení patchu je tento patch posielaný na server, kde sa patch aplikuje na daný zdieľaný dokument. Je totiž dôležité mať na serveri vždy aktuálny dokument. Ak by sa chcel totiž pripojiť ďalší užívateľ s iným zariadením, dožaduje sa opäť servera na celý dokument. Dokument teda musí byť aktuálny, aby ďalej bolo jednoduchšie posielať aj tieto zmeny. Po zmene dokumentu na serveri je takisto patch poslaný aj pripojeným zariadeniam, ktoré taktiež následne využijú patch na aplikovanie zmien.

Metóda aplikovania týchto zmien na serveri a na konkrétnom zariadení klienta je však svojim spôsobom odlišná. Keďže na serveri je reprezentovaný tento súbor ako JSON, tak sa iba aplikuje zmena. Takáto zmena prebieha tak, že sa v JSON upraví daná bunka, ktorá bola zmenená. Lenže na zariadení u prijímateľa je potrebné túto zmenu aj vizualizovať. Vizualizovanie zmien je riešené pomocou API pre VS Code obsahujúceho edit metódu nad editorom textového bloku. Za pomoci tejto metódy je aplikovaná metóda insert, alebo delete, v závislosti od toho, či je text do danej bunky pridávaný, alebo odstraňovaný. Pri zmenách textu sa navyše automaticky užívateľovi ukáže práve tá bunka, ktorú v danom momente zmenil zdieľateľ. Pre prípadné editovanie textu bunky, by bolo možné editovať aj za pomoci replaceCells metódy nad poznámkovým blokom. Avšak pokiaľ je takto upravovaná bunka s výstupmi, sú aj výstupy opätovne generované a vizuálne to nepôsobí dobre.

Prístup implementácie zmien na základe patchu je oveľa jednoduchší na réžiu, ako prístup, kde by sa pri každej textovej zmene súboru posielal napríklad celý súbor, alebo celý obsah bunky.

Pri textových zmenách je však dobré spomunúť aj posielanie označení daného textu. Pokiaľ zdieľateľ označí nejaký text v rámci danej bunky, tak toto označenie sa zobrazí aj na zariadeniach prijímateľov. Z hľadiska prínosu do tejto aplikácie je to veľmi dobrá funkcia. Uplatnenie určite nájde v momente, kedy učiteľ vysvetľuje nejakú časť poznámkového bloku. Túto časť môže študentom označiť a prednášať im výklad o tom, ako daná časť poznámkového bloku funguje.

Zmeny buniek

V prípade, že dochádza ku zmenám v rámci buniek poznámkového bloku, je nutné takisto túto zmenu odoslať na server, spracovať ju a poslať opäť na všetky pripojené zariadenia. O zmenách buniek informuje udalosť *vscode.notebooks.onDidChangeNotebookCells()*. V rámci tejto udalosti je potom možné rozlišovať, či zmena nastala v podobe pridania, vymazania, či presunutia bunky. Po vyhodnotení o akú zmenu sa jedná, sú informácie o tejto zmene posielané na server. Ten túto zmenu takisto spracováva a následne odosiela ďalej na pripojené zariadenia.

V prípade pridania bunky je bunka pridaná do JSON na serveri. Následne je takisto pridaná aj na strane pripojených zariadení a to za pomoci edit metódy nad poznámkovým blokom. Konkrétne sa jedná o metódu replaceCells, ktorá nahradí dané bunky. Naopak, ak je bunka odstránená, tak je odstránená ako aj z JSON na serveri, tak takisto aj za pomoci replaceCells metódy, ktorá dokáže aj vymazávať bunky. V prípadoch zlúčení alebo rozdelení buniek sa jedná o rovnaký princíp. Zlúčenie je totiž iba vymazanie bunky a prípadne následné pridanie textu z vymazanej bunky do bunky, do ktorej sa zlučuje. Na druhej strane rozdelenie bunky vytvára novú bunku, do ktorej sa prípadne pridá text z predchádzajúcej bunky.

Bunky je možné aj presúvať. Pri presúvaní buniek je využívaný ten istý princíp ako pri odstraňovaní, respektíve pridávaní buniek. Presuny buniek totiž fungujú tak, že presúvaná bunka je najprv vymazaná zo svojej danej pozície, na ktorej sa pôvodne nachádzala. Následne je znovu pridaná na miesto, kam bola presunutá.

Čo však zatiaľ nefunguje, je funkcia "drag and drop", teda funkcia ťahaj a pusti. Táto funkcia nefunguje iba pre presúvanie viacerých buniek. VS Code API totiž zatiaľ v natívnych poznámkových blokoch pri tejto funkcii nemá informáciu o tom, že sa presúva viac buniek. Preto je odporúčané zatiaľ túto funkciu nevyužívať pri používaní.

O zmenách v rámci viditeľnosti buniek sa dá hovoriť vtedy, keď je buniek v poznámkovom bloku viac ako ich je práve zobrazených. Pri zmenách v rámci vizualizácii buniek sa táto zmena takisto odosiela a editor sa u prijímateľa automaticky presúva na rozmedzie tých buniek, ktoré vidí aj prijímateľ.

Zmeny výstupov

Výstupy bývajú takisto rôznych typov. Výstup je vytvorený potom, čo zdieľateľ spustí nejakú bunku, respektíve viacero buniek. Vytvorenie výstupu je teda stále kompletne riadené rozšírením Jupyter. Pri vytváraní výstupu, je však výstup možné zachytiť pomocou udalosti *vscode.notebooks.onDidChangeCellOutputs()*. Táto udalosť obsahuje informácie o danom výstupe, ktorými sú najmä jeho typ, index a dáta, ale aj mnoho ďalších. Od momentu, kedy nastane udalosť, je výstup zachytený naším rozšírením a obsah tejto udalosti je posielaný ďalej po sieti.

V prípade, že zdieľateľ bunku, či viacero buniek spustí, bývaju informácie o vytvorenom výstupe odosielané na server, ktorý si ich uloží do súboru uloženého na serveri. Následne tieto informácie o zmene výstupu posiela aj na pripojené zariadania v danej miestnosti. Po príchode dát zo servera je následne výstup pridaný a zobrazený aj na všetkých pripojených zariadeniach.

Výstup je teda posielaný na pripojené zariadenia. Takýto prístup implementácie zabezpečí, že rovnaký výstup má na svojom zariadení, ako učiteľ, tak aj študent. Popritom, ako učiteľ vyučuje, tak študent ihneď uvidí výsledok akcie učiteľa. Predpokladá sa komunikácia medzi učiteľom a žiakmi. Učiteľ tak môže okamžite aj okomentovať to, čo práve nastalo.

Na pripojenom zariadení bolo však potrebné vhodne implementovať priradenie jednotlivých výstupov ku konkrétným bunkám. Nakoniec je najlepším riešením zvoliť opäť metódu replaceCells. Tá dokáže zmeniť rovnako, ako usporiadanie a obsah buniek, tak aj ich výstupy.

Zhodnotenie výsledku implementácie

Vývoj výsledného riešenia bol z veľkej časti založený na API pre VS Code. Toto API ponúka viacero možností pre prácu s rôznymi časťami editora VS Code. Pre potrebu tejto aplikácie však bolo treba použiť konkrétne aj Notebook API. Notebook API je však označený ako proposed API. To znamená, že síce môže byť takéto API využívané v rámci rozšírení, ale samotné API bolo vo vývoji a dochádzalo často k zmenám v rámci jeho funkcií. Príkladom môže byť aj využívaný NotebookSerializer, ktorý bol do tohto API pridaný v apríli tohto roka. Výrazne uľahčuje prácu s danými bunkami jednotlivých poznámkových blokov.

Spolu s vývojom daného API občasne dochádzalo aj k neočakávaným chybám, kedy nejaká funkcionalita daného API nefungovala správne. Príkladom môže byť aj bunka typu markdown, ktorú editor nevie momentálne vygenerovať pomocou metódy replaceCells.

Aj keď dochádzalo počas vývoja k týmto rôznym problémom, tak výsledná aplikácia sa podarila skomponovať do pomerne dobrej podoby. Zatiaľ však funguje podobne ako LiveShare iba vo VS Code Insiders, kvôli Notebook API. Aplikácia je založená na jednoduchom princípe. Existujú dve základné možnosti použitia aplikácie a tými sú zdieľanie poznámkového bloku, alebo prípadne prijímanie poznámkového bloku. Pri zdieľaní sa jedná o odosielanie dát na server. Opačne pri prijímaní sa jedná o prijímanie dát zo servera, ale aj k následnej vizualizácii daných dát na zariadení prijímateľa. Aplikácia má už vopred stanovené nejaké obmedzenia pred samotným spustením. Nasledujúce obmedzenia sú definované pre zdieľateľa:

- nutnosť mať otvorený daný poznámkový blok, ktorý chce užívateľ zdieľať s ostatnými užívateľmi,
- nemožnosť zdieľať do dvoch miestností naraz,
- nemožnosť zdieľať poznámkový blok v prípade, že je užívateľ na nejaký súbor pripojený ako prijímateľ.

Pre prijímateľa sa jedná o tieto obmedzania:

- nutnosť mať otvorenú nejakú zložku, do ktorej bude uložený zdieľaný súbor,
- nemožnosť prijímať poznámkový blok z dvoch miestností naraz,
- nemožnosť prijímať poznámkový blok v prípade, že už užívateľ nejaký súbor zdieľa,

Pre obe strany platia obmedzenia:

- v rámci editora nemôže byť otvorených viac zložiek ako jedna,
- užívatelia musia zadať nejaký názov miestnosti. Pokiaľ miestnosť s daným názvom existuje, je následne zdieľateľ nútený vytvoriť miestnosť s iným názvom. Naopak u prijímateľa je potrebné, aby miestnosť existovala.

Požiadavky na server

Bežnému užívateľovi sa môže zdať, že dáta sú prenášané priamo medzi počítačmi zdieľateľa a prijímateľa. V rámci tohto projektu je server aplikovaný ako spojovateľ pri komunikovaní medzi zdieľateľom a prijímateľom. Pri momentálnej implementácii serveru, by mal byť server schopný zvládnuť stovky až tisícky pripojení. Ak by náhodou dochádzalo k väčšiemu využitiu aplikácie, kedy by už server nebol schopný zvládať daný nápor zo strany klientov, je možné server rozdeliť. Vtedy by bol spustený v rôznych procesoch. Dokázal by tak zvládnuť väčšie množstvo pripojení. V prípade masívneho využívania aplikácie, by bolo lepšie využiť ukladanie dát do databázy namiesto JSON. To by vo výsledku pomohlo znížiť réžiu servera.

Do budúcnosti by bolo rozumné aplikáciu rozšíriť aj o možnosť ukladania súboru na server. Na tento súbor by sa následne pri nejakej rozpracovanej práci mohli ľudia napojiť. Potom by bolo dôležité zamerať sa aj na väčšiu bezpečnosť v rámci pripojenia na server. Na serveri by bolo vhodné užívateľov autentifikovať na základe nejakých prihlasovacích údajov.

Kapitola 6

Záver

Hlavným cieľom práce bolo vytvoriť rozšírenie editora Visual Studio Code, ktoré umožňuje zdieľať obsah poznámkových blokov. V tejto práci boli predstavené rôzne technológie a možnosti, ktoré je možné, alebo potrebné využiť na tvorbu takéhoto rozšírenia. Predstavené boli technológie zdieľania dát v reálnom čase, rovnako tak, ako boli predstavené aj spôsoby synchronizácie takéhoto obsahu. Jednalo sa najmä o protokol **Websocket** a knižnicu *Socket.io*. Ďalej bol predstavený aj editor Visual Studio Code a tvorba rozšírení v tomto editore. Následne bol popísaný aj jazyk TypeScript, ktorý je využívaný pri tvorbe rozšírení pre editor Visual Studio Code.

Počas návrhu a implementácie aplikácie som sa stretol s viacerými problémami, ktoré bolo treba nutne riešiť. Problémom bol občas aj samotný editor Visual Studio Code a konkrétnejšie notebook API v rámci daného editora. Notebook API bolo neustále počas vývoja aplikácie vo vývoji a tak ho vývojári neustále aktualizovávali. Občasne bolo treba upravovať súbory len kvôli aktualizácii daného API. Je však vhodné zmieniť, že s každou aktualizáciou väčšinou vždy prišla aj nejaká pozitívna zmena v rámci funkcionality, ktorá viac uľahčila prácu so samotným poznámkovým blokom.

V práci sú použité viaceré technológie. Jazyk TypeScript, knižnica socket.io, či node.js. Možno spomenúť aj samotnú tvorbu daného rozšírenia. Práce s danými API vo Visual Studio Code neboli vždy jednoduché a často meniace sa API nebolo vždy úplne dobre dokumentované. Väčšina vývoja tak prebiehala v aplikácii Visual Studio Code Insiders, ktorá umožňuje prístup k nedokončeným verziám VS Code API, respektíve notebook API.

V práci okrem rozšírenia editora, ktorý je vlastne prehliadačom poznámkových blokov aplikujúcim prichádzajúce zmeny do poznámkového bloku, bolo nutné implementovať aj server. Server slúži ako správca obsahu jednotlivých poznámkových blokov medzi zdieľateľom a prijímateľom. Cez server prechádza každá jedna zmena, ktorú v poznámkovom bloku vykoná zdieľateľ. Táto zmena je následne aplikovaná na serveri, ako aj na strane pripojených klientov. Na serveri sa zmena aplikuje kvôli nutnosti mať neustále dostupný aktuálny súbor v prípade pripojenia nového zariadenia.

Podľa môjho názoru sa podarilo dosiahnuť všetky stanovené ciele. Rozšírenie pre realtime zdieľanie obsahu poznámkových blokov vo Visual Studio Code nájde dokonca uplatnenie nielen medzi učiteľmi a študentami. Využiť ho môžu aj bežní programátori. Jedná sa o jednosmerné zdieľanie. V prípade, že by chcel niekto ukázať svoj poznámkový blok iným, je možné využiť funkcie tohto rozšírenia.

Literatúra

[1] Fenton, S. *Pro TypeScript: Application-Scale JavaScript Development.* 1. vyd. Apress, september 2014. The expert's voice in TypeScript. ISBN 978-1-4302-6790-4.

[2] Fette, I. a Melnikov, A. *The WebSocket Protocol* [online]. RFC 6455. December 2011 [cit. 2021-20-04]. Dostupné z: https://www.rfc-editor.org/rfc/rfc6455.html.

[3] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L. et al. *Hypertext Transfer Protocol – HTTP/1.1* [online]. RFC 6455. Jún 1999 [cit. 2021-20-04]. Dostupné z: https://www.rfc-editor.org/rfc/rfc6455.html.

[4] Fraser, N. *Differential Synchronization* [online]. 2009 [cit. 2021-29-06]. Dostupné z: https://neil.fraser.name/writing/sync/.

[5] GNU. *Comparing and Merging Files* [online]. November 2008 [cit. 2021-29-06]. Dostupné z: https://www.gnu.org/software/diffutils/manual/diffutils.html.

[6] Johnson, J. *Share of adults in the United States who use the internet in 2021, by age group* [online]. 2021 [cit. 2021-29-06]. Dostupné z: https://www.statista.com/statistics/266587/percentage-of-internet-users-by-age-groups-in-the-us/.

[7] Lombardi, A. *WebSocket: Lightweight Client-Server Communications.* 1. vyd. O'Reilly, september 2015. ISBN 978-1-4493-6927-9.

[8] Loreto, S., Saint Andre, P., Salsano, S. a Wilkins, G. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP* [online]. RFC 6202. Apríl 2011 [cit. 2021-20-04]. Dostupné z: https://www.rfc-editor.org/rfc/rfc6202.html.

[9] Microsoft. *Documentation for Visual Studio Code* [online]. 2021 [cit. 2021-02-07]. Dostupné z: https://code.visualstudio.com/docs.

[10] Microsoft. *Extension API* [online]. 2021 [cit. 2021-20-07]. Dostupné z: https://code.visualstudio.com/api.

[11] Microsoft. *Why did we build Visual Studio Code?* [online]. Júl 2021 [cit. 2021-02-07]. Dostupné z: https://code.visualstudio.com/docs/editor/whyvscode.

[12] Perkel, J. M. *Why Jupyter is data scientists' computational notebook of choice* [online]. 2018 [cit. 2021-20-04]. Dostupné z: https://www.nature.com/articles/d41586-018-07196-1.

[13] Rai, R. a (Firm), P. *Socket.io Real-time Web Application Development.* 1. vyd. Packt Pub., 2013. Community experience distilled. ISBN 978-1-78216-078-6.

[14] Socket.io. *Introduction: What Socket.IO is* [online]. 2021 [cit. 2021-07-02]. Dostupné z: https://socket.io/docs/v4/.

[15] Stackoverflow. *Developer Survey Results 2019* [online]. 2019 [cit. 2021-20-04]. Dostupné z: https://insights.stackoverflow.com/survey/2019.