دانشکده مهندسی کامپیوتر و فناوری اطلاعات

گروه فناوری اطلاعات

عنوان

طراحی و پیاده سازی ویرایشگر کد مشارکتی همزمان برای توسعه وب

نگارش

محمد ناصری

استاد راهنما

دکتر پیروز شمسی نژاد

مهرماه ۱۳۹۹


بسم الله الرحمن الرحيم

چکیده

با توجه به افزایش روزافزون توسعه دهندگان وب و همینطور علاقه مندان به این حوزه، نیازهای عملیاتی توسعه و آموزش در این حوزه نیاز به برداشتن قدم‌هایی رو به جلو جهت افزایش سرعت، کارایی و راحتی احساس میشود.

با اینکه تکنولوژی‌های موجود پاسخگوی برخی نیازهای کاربران هستند، ولی در راستای پیشرفت توسعه در این زمینه همچنان پیشنهادات و مسائل فراوانی پیش روی توسعه دهندگان وجود دارد؛ یکی از این مسائل که چندین سال اخیر مورد بحث و توجه توسط توسعه دهندگان بوده، مفهوم ویرایش مشارکتی و همزمان است. در این پروژه قصد داریم تا یک پلتفرم ویرایش کد اشتراکی مخصوص زبان های تحت وب با قابلیت‌های منحصر به فرد نسبت به محصولات مشابه طراحی و پیاده‌سازی نماییم.

در این متن ابتدا به تعریفی خلاصه از مفهوم ویرایشگرهای اشتراکی پرداخته و سپس به بررسی و نیازسنجی استفاده از پلتفرم‌های ویرایشگر کد اشتراکی و نقاط تمایز این پلتفرم‌ها با دیگر روش‌ها مانند پلتفرم‌های کنترل نسخه [۱] پرداخته شده است .

پس از آن به بررسی راهکارهای پیاده‌سازی این پلتفرم‌ها و موانع و مشکلات پیش روی پیاده‌سازی و راه حل‌های موجود برای این موانع میپردازیم.

در آخر با توجه به نتایج بدست آمده به طراحی و پیاده سازی یک پلتفرم ویرایش اشتراکی کد با نام Code۲Gether، که یک پلتفرم ویرایش اشتراکی کد برای زبان‌ها تحت وب میباشد، پرداخته و تکتولوژی‌های مورد استفاده و نحوه پیاده‌سازی، راه‌اندازی و کارکرد این پلتفرم را مورد بررسی قرار خواهیم داد.

[۱] Version Control

فصل اول : مقدمه

۱- ۱ مقدمه

نوشتن مشارکتی[۲] نوشتاری است که توسط بیش از یک نفر انجام میشود. افراد ممکن است قبل از شروع در مورد آنچه قرار است بنویسند بحث کنند و بعد از اتمام هر پیش نویس که می نویسند درباره آنچه که نوشته اند به گفتگو بپردازند. در عین حال تایپ کردن مطالب بصورت گروهی به همراه تقسیم نوشتار به کارهای کوچکتر و اختصاص هر کار به هر یک از اعضای گروه کار را بسیار ساده‌تر خواهد کرد. در این روش میبایست نوشتن متن ابتدا برنامه ریزی شده، سپس نوشته شده و در آخر نیز اصلاح شود و بطور معمول در این روند، بیش از یک نفر حداقل در یکی از مراحل مذکور دخیل است.

ویرایش مشارکتی معمولاً روی اسناد متنی یا کد منبع[۳] برنامه اعمال می شود. چنین مشارکتهای همزمان[۴] (غیرهمزمان[۵]) از نظر زمانی بسیار کارآمد هستند، زیرا اعضای گروه برای کار با یکدیگر نیازی به جمع شدن ندارند. به طور کلی، مدیریت چنین کارهایی به یک پلتفرم احتیاج دارد؛ متداول ترین ابزار برای ویرایش اسناد ویکی[۶] و سایر موارد کاربردی مانند برنامه نویسی ، سیستم های کنترل نسخه هستند.

پانزده سال پیش، بیشتر همکاری های آنلاین و کار از راه دور شامل روش‌های همچون ایمیل، تماس های اسکایپ و بی‌شمار فشرده‌سازی پروژه و اشتراک پروژه بر بستر اینترنت بعد از هر تغییر بود. این امر امروزه دستخوش تحولات بسیار زیادی شده و با ظهور پلتفرم های جدید، منصفانه است که بگوییم از اصطکاک‌های کار از راه دور بسیار کاسته شده است؛ ولی امروزه افراد ممکن است در یک تیم یا شرکت به صورت از راه دور مشغول به کار باشند و به ابزارهای برنامه نویسی مشترک برای همکاری و هماهنگی نیاز پیدا کنند. همچنین امکان دارد که به لطف COVID-۱۹، به طور ناگهانی و به اجبار، بیشتر به سمت دورکاری رفته باشند و یا شاید فقط برای یک جلسه جهت حل مسئله با یک دوست یا همکار و یا حتی استفاده در موارد آموزشی، به پلتفرمی برای ارتباط و ویرایش همزمان منابع خود احتیاج داشته باشند. این در حالی است که با وجود اینکه ابزارهای دیگر به جلو حرکت کرده‌اند، همکاری هنگام برنامه نویسی در Real-Time همچنان یک نقطه بحث است و راه حل های مناسب این موضوع به تازگی و اخیراً به صحنه اجرا رسیده اند و توسعه دهندگان ویرایشگرهای کد برتر دنیا راه حل های خود را پیرامون آن ارائه داده اند.

---

[۲] Collaborative Editing
[۳] Source Code
[۴] Synchronous
[۵] Asynchronous
[۶] Wiki

ویرایشگر مشارکتی در لحظه[۷]، نوعی نرم افزار مشارکتی یا برنامه تحت وب است که امکان ویرایش مشارکتی، ویرایش همزمان یا ویرایش مستقیم یک سند دیجیتال واحد، پرونده رایانه یا داده های ذخیره شده ابری[۸] مانند word را فراهم می کند. پردازش سند، پایگاه داده یا ارائه همزمان توسط کاربران مختلف در رایانه ها یا دستگاه های تلفن همراه مختلف، با ادغام خودکار و تقریباً فوری ویرایش های آنها صورت میپذیرد.

برخلاف ویرایش های همزمان ناهمگام[۹] (تأخیری یا آفلاین[۱۰]) ، مانند آنچه در سیستم های کنترل نسخه مانند Git یا Subversion رخ می دهد، در ویرایش همزمان، همسان سازی خودکار در حالی که کاربران در حال ویرایش سند روی دستگاه خودشان هستند بصورت دوره ای یا اغلب تقریباً فوری انجام می دهند. این روش برای جلوگیری یا به حداقل رساندن تعارضات ویرایش طراحی شده است. در ویرایش مشترک غیر همزمان ، هر کاربر معمولاً باید به صورت دستی ویرایشات خود را ارسال (Publish، Push یا Commit) یا به روزرسانی (Refresh، Pull، Download یا Sync) بنماید و (در صورت بروز هرگونه تضاد ویرایش) ویرایش های خود را ادغام کند.

به دلیل ماهیت تأخیر در ویرایش مشترک ناهمزمان، چندین کاربر می توانند در نهایت ویرایش همان خط، کلمه، عنصر، داده یا ردیف را انجام دهند که منجر به درگیری[۱۱] ویرایش شود که به ادغام دستی یا ادغام مجدد نیاز دارد.

در قسمت‌های آینده این نوشتار با نحوه کارکرد پلتفرم‌های کنترل نسخه و ویرایشگرهای اشتراکی بیشتر آشنا شده و تفاوت‌ها و تمایزهای این دو مفهوم را مشاهده میکنیم.

---

[۷] Real-Time Collaborative Editor
[۸] Cloud
[۹] Asynchronous
[۱۰] Offline
[۱۱] Conflict

فصل دوم: ویرایش مشارکتی ناهمزمان

۱-۲ تاریخچه

به طور کلی، تاریخچه ابزار کنترل نسخه را می توان به سه نسل تقسیم کرد :

نسل	شبکه‌سازی	عملیات‌ها	همزمانی	مثال‌های موجود
اول	بدون شبکه‌سازی	یک سند در هر لحظه	قفل	RCS, SCCS
دوم	متمرکز	چند پرونده	ادغام قبل از Commit	CVS, SourceSafe, Subversion, Team Foundation Server
سوم	توزیع‌شده	تغییرات	Commit قبل از ادغام	Bazaar, Git, Mercurial

جدول ۱ سه نسل کنترل نسخه

سابقه چهل ساله ابزارهای کنترل نسخه، حرکت مداوم به سمت همزمانی بیشتر را نشان می دهد.

- در ابزارهای نسل اول، توسعه همزمان فقط با قفل انجام می شد. یعنی فقط یک نفر می تواند همزمان روی یک پرونده کار کند.
- ابزارهای نسل دوم در مورد تغییرات همزمان ، با یک محدودیت قابل توجه ، کمی راحت تر هستند. قبل از اینکه کاربران مجاز به انجام Commit شوند، باید اصلاحات فعلی را در کار خود ادغام کنند.
- ابزارهای نسل سوم اجازه می دهد ادغام و Commit جدا شود.

در اواسط سال ۲۰۱۱، دنیای کنترل نسخه تغییرهای بزرگی به خود دید. اکثریت قریب به اتفاق برنامه نویسان حرفه ای از ابزارهای نسل دوم استفاده می کردند اما محبوبیت ابزار نسل سوم بسیار سریع در حال رشد بود. محبوب ترین VCS[۱۲] (ابزار کنترل نسخه) Apache Subversion بود، ابزاری منبع باز[۱۳] و در دسته نسل دوم ابزارها. بازار تجاری این ابزارها تحت سلطه IBM و Microsoft بود که هر دو در ابزارهای نسل دوم کاملاً جا افتاده بودند. اما در سطح جامعه، جایی که توسعه دهندگان در سراسر جهان درباره چیزهای جدید و جالب صحبت می کردند، صحبت ها پیرامون سیستم های کنترل نسخه توزیع شده (DVCS[۱۴]) بود. سه ابزار محبوب DVCS در آن زمان عبارت بودند از Bazaar، Git و Mercurial.

[۱۲] Version Control System
[۱۳] Open Source
[۱۴] Distributed Version Control System

۲-۲ چرا از کنترل نسخه استفاده میکنیم ؟

Version Control (با نام مستعار Revision Control یا همان کنترل منبع) به شما امکان می دهد با گذشت زمان پرونده های خود را ردیابی کنید. چرا این امر برای ما مهم است؟ زیرا در صورت بروز مشکل این امکان وجود دارد که به راحتی به نسخه قبلی کار برگردید. شما تا به حال احتمالاً سیستم کنترل نسخه خود را بدون اینکه بدانید چنین نام گیج کننده‌ای دارد، استفاده کرده‌اید. شما حتما پرونده‌هایی از این قبیل دارید:

- Resume۲۰۱٤
- Resume۲۰۱٥
- Logo-Old
- Logo-new

به همین دلیل است که ما از "Save As" استفاده می کنیم. شما فایل جدید را بدون از بین بردن پرونده قدیمی در دسترس خود می خواهید. این یک مشکل معمول است و راه حل ها معمولاً به این صورت هستند:

- یک نسخه پشتیبان تهیه می‌کنید.(Document.old.txt)
  - اگر باهوش باشیم، یک شماره نسخه یا تاریخ اضافه می کنیم
- حتی ممکن است از یک پوشه مشترک استفاده کنیم تا افراد دیگر بتوانند فایل ها را بدون ارسال از طریق ایمیل مشاهده و ویرایش کنند و امیدوار باشیم که افراد پس از ذخیره فایل، آنها را دوباره برچسب گذاری کنند.

۱-۲-۲ چرا ما به سیستم‌های کنترل نسخه نیاز داریم ؟

پوشه مشترک / سیستم نامگذاری ما برای پروژه های کلاسی یا مقالات یکبار مصرف مناسب است. اما پروژه های نرم افزاری چطور؟ به هیچ وجه چنین امکانی وجود ندارد و این سیستم پاسخگوی نیازهای این دسته از پروژه‌ها نیست. آیا فکر می کنید کد منبع پروژه‌های عظیم در یک پوشه مشترک، برای ویرایش افراد قرار دارد؟ و برنامه نویسان بدین صورت عمل میکنند که هر برنامه نویس فقط در یک زیر پوشه متفاوت کار می کند؟ به هیچ وجه.

پروژه های بزرگ و سریع با تعداد بسیاری از نویسندگان برای پیگیری تغییرات و جلوگیری از هرج و مرج عمومی به سیستم کنترل نسخه نیاز دارند. VCS موارد زیر را انجام می دهد:

- **پشتیبان گیری و بازیابی:** پرونده ها هنگام ویرایش ذخیره می شوند و می توانید در هر لحظه از زمان به نسخه مورد نظر خود در تاریخ و زمان مشخص دسترسی داشته باشید

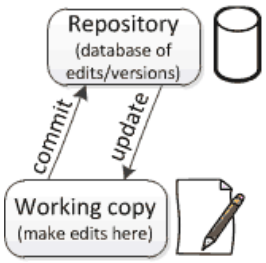
- **هماهنگ سازی:** به افراد اجازه می دهد تا پرونده ها را به اشتراک بگذارند و از آخرین نسخه به روز شده اسناد و برنامه‌ها بهره‌مند باشند.
- **Undo کوتاه مدت :** یک فایل را تغییر دادید و آنرا خراب کردید؟ تغییرات خود را دور بریزید و به آخرین نسخه سالم در پایگاه داده برگردید.
- **Undo طولانی مدت:** فرض کنید شما یک سال پیش تغییری ایجاد کردید و حال حاضر مشکلی در سیستم به وجود آورده. به نسخه قدیمی برگردید و ببینید چه تغییری در آن روز ایجاد شد.
- **دنبال کردن تغییرات:** با به روزرسانی پرونده ها ، می توانید پیام هایی را برای توضیح علت تغییر ثبت کنید (ذخیره شده در VCS، نه در پرونده). این کار باعث می شود که درک کنید چگونه یک پرونده در طول زمان تکامل می یابد و چرا.
- **پیگیری مالکیت:** VCS هر تغییری را با نام شخص ایجاد کننده برچسب می زند.
- **انشعاب و ادغام:** می توانید یک کپی از کد خود را در یک قسمت جداگانه منشعب کنید و آن را جداگانه اصلاح کنید (پیگیری به طور جداگانه تغییر می کند) و سپس می توانید کار خود را دوباره در پرونده مشترک ادغام کنید.

پوشه های مشترک سریع و ساده هستند ، اما نمی توانند با این ویژگی‌ها به رقابت بپردازند

۲-۴ عملکرد سیستم کنترل نسخه

بطور کلی کنترل نسخه از مخزنی[۱۵] (پایگاه داده تغییرات) و یک کپی که در آن کار خود را انجام می دهید تشکیل شده است.

کپی شما (که گاهی اوقات آنرا Checkout می نامند) کپی شخصی شما از تمام پرونده های پروژه است. شما بدون تأثیر روی هم تیمی های خود، این نسخه را به دلخواه ویرایش می کنید. وقتی از ویرایش های خود راضی بودید و کار شما به پایان رسید، تغییرات خود را در یک مخزن به ثبت میرسانید. مخزن یک پایگاه داده از تمام



دیاگرام ۱ ساختار ساده سیستم کنترل نسخه

[۱۵] Repository

ویرایش ها و یا نسخه های تاریخی (Snapshots) پروژه شما است. این مخزن ممکن است حاوی ویرایش هایی باشد که هنوز روی نسخه فعال شما اعمال نشده است. شما می توانید نسخه کاری خود را به روز کنید تا هر ویرایش یا نسخه جدیدی را که از آخرین باری که به روز کرده اید به مخزن اضافه شده است، در آن قرار گیرد.

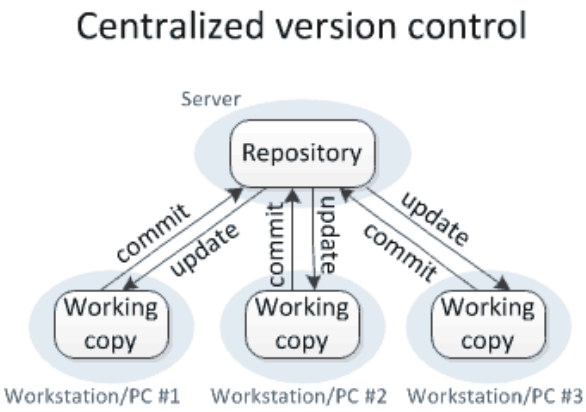
در ساده ترین حالت، پایگاه داده شامل یک تاریخچه خطی است: هر تغییر پس از تغییر قبلی ایجاد می شود. احتمال دیگر این است که کاربران مختلف به طور همزمان ویرایش کنند (این کار گاهی اوقات "انشعاب" نامیده می شود). در این حالت، تاریخچه نسخه شکسته می شود و سپس دوباره ادغام می شود. در تصویر زیر مثالهایی از این موارد آورده شده است.

دیاگرام ۲ساختار پروژه و اسناد در طول زمان

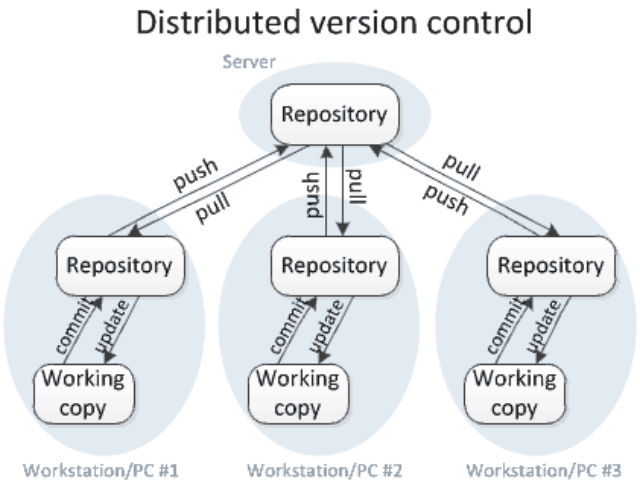
۱-۴-۲ کنترل نسخه توزیع شده و متمرکز

دو نسخه کلی از کنترل نسخه وجود دارد: متمرکز[۱۶] و توزیع شده[۱۷]. کنترل نسخه توزیع شده مدرن تر است، سریعتر اجرا می شود، کمتر در معرض خطا است، ویژگی های بیشتری دارد و درک آن تا حدودی پیچیده تر است. تفاوت اصلی بین کنترل نسخه متمرکز و توزیع شده در تعداد مخازن است. در کنترل نسخه متمرکز، فقط یک مخزن وجود دارد و در کنترل نسخه توزیع شده، چندین مخزن وجود دارد. در اینجا تصاویری از چیدمان های معمول این نسخه‌ها آورده شده است:

[۱۶] Centralized
[۱۷] Distributed



دیاگرام3 ساختار سیستم کنترل نسخه متمرکز



دیاگرام ۴ ساختار سیستم کنترل نسخه توزیع شده

در کنترل نسخه متمرکز، هر کاربر نسخه کار خود را می گیرد، اما فقط یک مخزن مرکزی وجود دارد. به محض Commit، همکاران شما امکان به روزرسانی و مشاهده تغییرات شما را دارند. برای اینکه دیگران تغییرات شما را ببینند، ۲ اتفاق باید بیفتد:

۱. شما Commit میکنید
۲. آنها بروزرسانی می کنند

در کنترل نسخه توزیع شده، هر کاربر مخزن و نسخه کار خود را دارد. پس از انجام Commit، دیگران دسترسی به تغییرات شما ندارند تا زمانی که تغییرات خود را به مخزن مرکزی Push کنید. هنگام به روزرسانی، تغییرات دیگران را دریافت نمی کنید مگر اینکه ابتدا آن تغییرات را در مخزن خود Pull کنید. برای اینکه دیگران تغییرات شما را ببینند، باید ۴ اتفاق بیفتد:

۱. شما Commit میکنید
۲. شما Push می کنید
۳. آنها Pull میکنند
٤. آنها بروزرسانی می کنند

توجه داشته باشید که دستورات commit و update فقط تغییراتی بین نسخه فعال و مخزن محلی ایجاد میکنند، بدون اینکه بر مخزن دیگری تأثیر بگذارد. در مقابل، دستورات Push و Pull بدون اینکه روی نسخه کار شما تأثیر بگذارد، بین مخزن محلی و مخزن مرکزی تغییراتی ایجاد می کنند.

۲-۴-۲ درگیری ها (Conflicts)

یک سیستم کنترل نسخه به چندین کاربر اجازه می دهد نسخه های خود را از یک پروژه به طور همزمان ویرایش کنند. معمولاً سیستم کنترل نسخه می تواند تغییرات همزمان را توسط دو کاربر مختلف ادغام کند: برای هر خط، نسخه نهایی نسخه اصلی است اگر هیچ یک از کاربران آن را ویرایش نکرده باشند یا اگر یکی از کاربران آن را ویرایش کرده باشد، نسخه نهایی نسخه آن کاربر است. تعارض زمانی رخ می دهد که دو کاربر متفاوت همزمان و متفاوت در همان خط از پرونده ویرایش ایجاد کنند. در این حالت، سیستم کنترل نسخه نمی تواند به طور خودکار تصمیم بگیرد که از هر دو ویرایش استفاده کند (یا ترکیبی از آنها، یا هیچ کدام!). برای حل منازعه، مداخله دستی لازم است.

تغییرات "همزمان" لزوماً دقیقاً در همان لحظه از زمان اتفاق نمی افتند. تغییر ۱ و تغییر ۲ به طور همزمان در نظر گرفته می شوند :

۱. کاربر A قبل از اینکه بروزرسانی انجام دهد که تغییر ۲ را در نسخه کاربری خود ایجاد کند، تغییر ۱ را ایجاد می کند و
۲. کاربر B قبل از اینکه بروزرسانی را انجام دهد که تغییر ۱ را در نسخه فعال خود ایجاد کند، تغییر ۲ را ایجاد می کند.

در یک سیستم کنترل نسخه توزیع شده، یک عمل صریح وجود دارد که merge نامیده می شود و ویرایش های همزمان دو کاربر مختلف را با هم ترکیب می کند. بعضی اوقات ادغام به طور خودکار کامل می شود، اما در صورت وجود تضاد، ادغام با اجرای یک ابزار ادغام از کاربر کمک می گیرد. در کنترل نسخه متمرکز، ادغام به طور ضمنی هر بار که به روز می کنید اتفاق می افتد.

فصل سوم: ویرایش مشارکتی همزمان

۱-۳ چالش

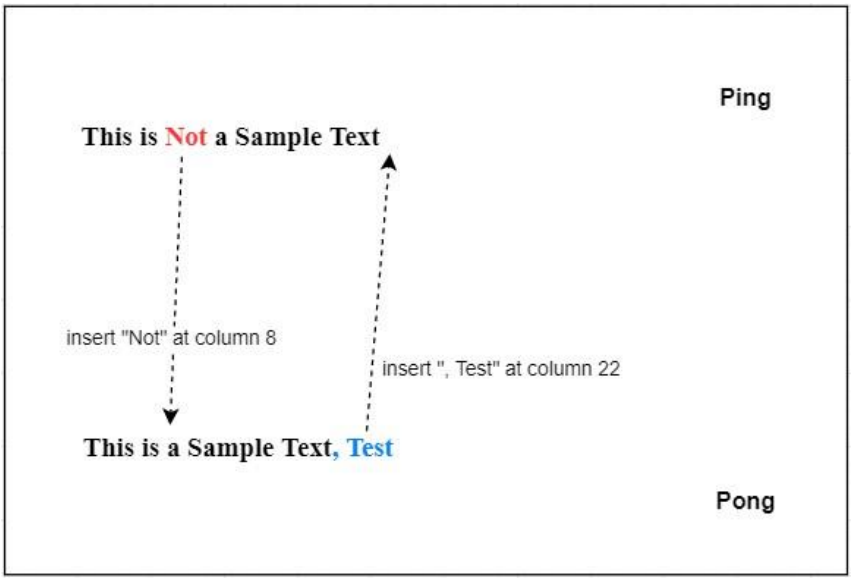
قبل از رفتن سراغ کد و پیاده‌سازی، ما باید راجع به تئوری این مبحث صحبت کنیم. پیچیدگی این سیستم توزیع شده را نمی‌توان دست کم گرفت و بنابراین یک نمای کلی در سطح بالا به درک آنچه اتفاق می افتد کمک خواهد کرد. در ابتدا، بطور مختصر آنچه را که پیاده‌سازی یک ویرایشگر متن مشارکتی را سخت می کند معرفی می کنیم.

ویرایش همزمان در محیط چند کاربره به طرز جالبی بسیار چالش برانگیز است. با این حال، چند مفهوم ساده می توانند این مشکل را ساده‌تر کنند. چالش اصلی با ویرایش مشترک این است که کنترل همزمانی (ویرایش های همزمان) سند دارای قابلیت جابجایی نیست. این امر باید قبل از اعمال یا با Undo کردن تاریخچه، یا با تغییر دادن عملیات قبل از اعمال آنها انجام شود تا به نظر برسد که عملیات‌ها جابجایی پذیر هستند.

۱-۱-۳ تاخیر در عملیات

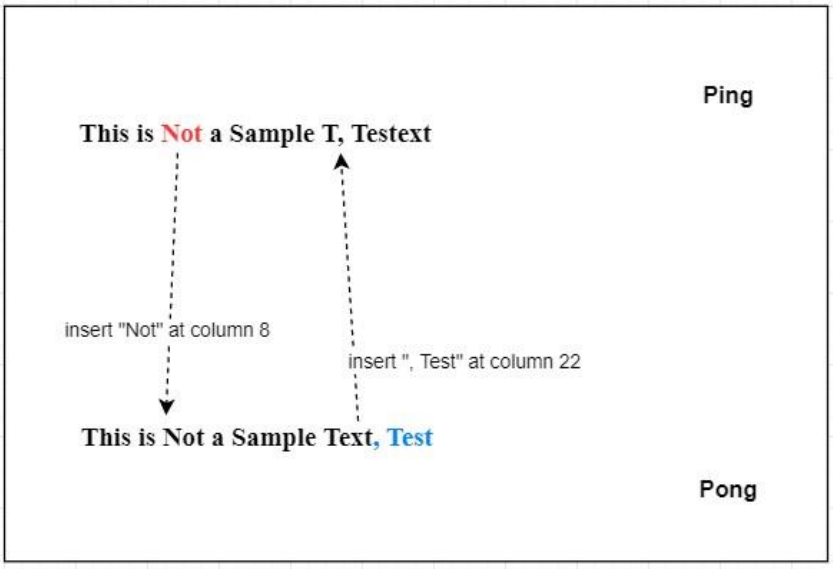
معرفی تأخیر بین Client و Server جایی است که مشکلات اصلی بوجود می آیند. تأخیر در یک ویرایشگر مشترک، احتمال تضاد نسخه را فراهم میکند.

بگذارید اینطور بگوییم که برای مثال شما دو دانشجو دارید، پینگ و پانگ، که با هم بر روی یک پروژه کار می‌کنند و در حال کار بر روی قسمت های مختلف سند هستند. پانگ یک تغییر ایجاد می کند: رشته "Test" را در انتهای خط وارد میکند. ساده‌ترین و پیش پا افتاده ترین راه که ویرایشگر پانگ می تواند تغییر را به ویرایشگر پینگ منتقل کند فقط ارسال یک پیام است. این سناریو در صورتی کار می کند که تغییرات بلافاصله قابل انتقال و اعمال در طرف مقابل باشد.



نصویر ۱ مثال تاثیر تاخیر در ویرایش همزمان

اما در واقعیت این اتفاق نخواهد افتاد. فرض کنید پیام پانگ در ترافیک شبکه گیر کرده و دیر به دست پینگ برسد و در عین حال پینگ رشته "Not" را به وسط جمله اضافه میکند. در این صورت در زمانی که پیام پانگ به دست پینگ میرسد انتهای جمله دیگر کاراکتر ۲۲ام نیست!



نصویر ۲ مثال تاثیر تاخیر در سیستم ورایش همزمان

در مثال قبلی، دو عملیات همزمان بر روی Client های مختلف اعمال شده است. وقتی آنها را به Client دیگر می فرستیم، در نهایت برای آنها عملیات‌های مختلفی اعمال می شوند و Client ها در وضعیت‌های متفاوتی قرار می گیرند.

```
insert(8, 'Not'), insert(22, ', Test')

!=

insert(22, ', Test'), insert(8, 'Not')
```

کد ۱ عدم امکان جابجایی عملیات ها در ویرایش همزمان

اگر هر عملیاتی بتواند با هر عملیات دیگری جابجا شود، Clientها بدون توجه به اینکه پیامها از کدام فرستنده میرسند، به همان حالت واحد موردنظر همگرا می شوند. اما در غیر این صورت، اگر عملیات در رفت و آمد باشد، همگام سازی به راحتی پیش نمی آید.

شرط دیگر همزمانی این است که شما بتوانید عملیاتی یکسان را دو بار انجام دهید و همان نتیجه‌ای را داشته باشید که یک بار اعمال عملیات به شما میدهد. این یعنی در صورتی که دو Client تصمیم بگیرند که عملیات مشابه را همزمان انجام دهند، دریافت عملیات مشتری دیگر (که اکنون تکراری محسوب میشود) باید ممنوع باشد (کاری انجام نگیرد).

۳-۲ راه حل

در این قسمت به بررسی یک راه حل برای موانع موجود در پیاده‌سازی ویرایشگر مشارکتی همزمان میپردازیم.

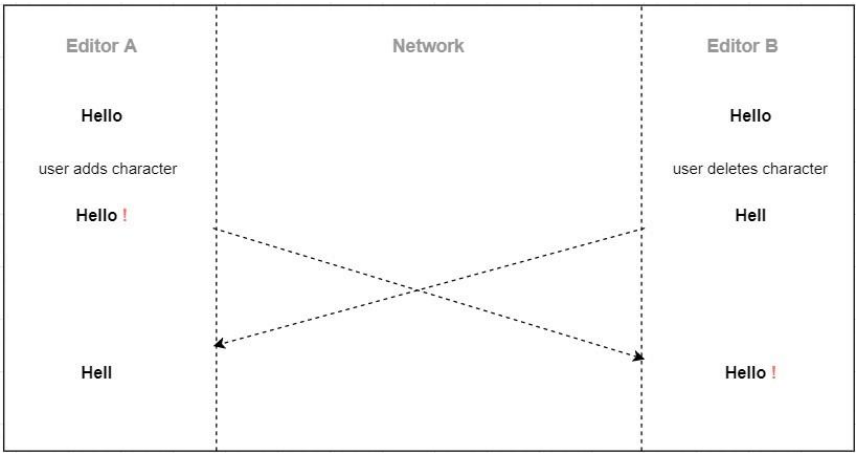
رفتار و شکل ظاهری ویرایشگر متن می تواند در هر برهه از زمان مانند یک Snapshot استخراج شده و در یک شی جاوا اسکریپت ساده یا JSON ذخیره شود. ما این Snapshot را حالت(State) می نامیم.

برای مشارکت بصورت همزمان، این حالت سند باید با ارسال پیام بین Nodeها از طریق یک شبکه، در بین چندین کاربر به اشتراک گذاشته شود. برای مدیریت صحیح این روند، یک پروتکل مورد نیاز است.

۳-۲-۱ چه مواردی نیاز به مدیریت دارند ؟

خوب، تصور کنید که دو کاربر همزمان چیزی را تایپ کنند. در چنین سناریویی :

۱. در نهایت هر دو مشتری با وضعیت دیگری مواجه خواهند شد
۲. یکی از این دو تغییر دوباره رونویسی می شود.



تصویر ۳ نمودار ویرایش همزمان یک کلمه توسط دو کاربر

دو مسئله ذکر شده در بالا مطابق با دو شرط مهم فنی است که پروتکل ما باید انجام دهد:

۱. همگرایی[۱۸]: همه ویراستاران باید پس از مدت زمان محدودی به یک حالت سند واحد همگرا شوند.
۲. همزمانی[۱۹]: ویرایش هایی که به طور موازی اتفاق می افتند، مستقل از ترتیب اجرا به یک نتیجه نهایی صحیح منجر می شوند.

برای اختصار، عمیق تر از این در این موضوع فرو نمی رویم و به سادگی می گوییم که می توان راه حل های موجود برای این پروتکل را در دو دسته زیر طبقه بندی کرد:

- Operational Transformation (OT): حالت سند را به عنوان دنباله ای از عملیات‌ها نشان میدهد. هر عملیات دارای یک Snapshot از وضعیت سند میباشد. حال، تصور کنید که این عملیات برای یک همکار ارسال شده است، که این همکار در این حین ویرایش انجام داده است. در اینصورت همکار مقصد Snapshot متفاوتی خواهد داشت، بنابراین ابتدا باید عملیات، قبل از اعمال، تغییر شکل(Transform) یابد. این اصل نحوه عملکرد OT است.
- Conflict-Free Replicated Data Type (CRDT): کمی پیچیده تر از OT است. از حافظه و پهنای باند بیشتری استفاده می کند، اما در ازای آن ثبات احتمالی و بدون نیاز به سرور مرکزی را تضمین می کند. بنابراین، می توانیم بگوییم از نظر نظری کامل تر است.

از آنجایی که در این نوشته و پروژه پیاده‌سازی شده از OT بهره گرفته شده در ادامه به بررسی این الگوریتم خواهیم پرداخت .

---

[۱۸] Convergence
[۱۹] Concurrency

۳-۳ Operational Transformation

OT توسط C. Ellis و S. Gibbs در سیستم GROVE (GRoup Outline Using Edit) در سال ۱۹۸۹ اختراع شد. تبدیل عملیاتی (OT) یک الگوریتم / تکنیک برای تبدیل عملیات است به گونه ای که می تواند در اسنادی که حالت‌های آنها متفاوت هستند اعمال شود و هر دو را به حالت همسان برگرداند. تحول عملیاتی[۲۰](OT) یک فناوری برای پشتیبانی از طیف وسیعی از ویژگی های همکاری در سیستم های نرم‌افزاری مشترک پیشرفته است. OT در اصل برای حفظ ثبات و کنترل همزمان در ویرایش مشترک اسناد متن ساده اختراع شد. در عین حال قابلیت ها و برنامه های آن گسترش یافته است که شامل واگرد(Undo) گروهی، قفل کردن، حل تعارض، اعلان و فشرده سازی عملیات و ... میشود . در سال۲۰۰۹، OT به عنوان یک تکنیک اصلی در پشت ویژگی های همکاری در Apache Wave و Google Docs پذیرفته شد.

۱-۳-۳ چرا از OT استفاده میکنیم ؟

OT، مناسب برای برنامه های وب است زیرا خوش‌بینانه است. سناریویی را در نظر بگیرید که چندین کاربر همزمان ویرایش یک سند میزبان شده در یک سرور مرکزی را انجام می دهند. هدف، ارائه تجربه ویرایش در زمان واقعی است تا کاربران بتوانند هنگام تغییر سند با یکدیگر همکاری کنند و تغییرات هر کاربر حفظ شود.

OT خوش بینانه فرض می کند که هر عملیاتی که در حال حاضر روی سند هر کاربر خاص اعمال می شود، با عملیاتی که ممکن است در همان لحظه توسط یکی دیگر از کاربرها اعمال شود منافاتی ندارد. این یعنی فرض اینکه تعارضات را نمی‌توان کشف کرد، تا اینکه با آنها مواجه شده و بعد به آنها رسیدگی شود. با این کار عملیات آزاد می شود تا روی سند اعمال شود زیرا توسط کاربر ایجاد می شود و رابط کاربری به روشی Responsive به روز می‌شود و نیازی به درخواست قفل سند از سرور نیست.

قفل کردن، و دیگر الگوریتم های همزمانی بدبینانه، در محدودیت های محیط برنامه وب مورد پسند نیستند، زیرا تأخیر زیادی صرف برقراری ارتباط رفت و برگشت با سرور و سایر سرویس گیرندگان میشود. اکثر کاربران در حالی که کاربر دیگری قفل را بدست گرفته است قادر به تغییر سند برای چند ثانیه نیستند. این یک مثال و نمونه برای تجربه بد کاربر است.

بعلاوه، طبیعی است که دیگر به فکر تغییر در سطح سند نباشیم و در عوض بر روی عملیات فردی که اعمال میشود تمرکز کنیم. این عملیات بسیار انعطاف پذیرتر از کل اسناد است. سوال اینجاست که اگر دو کاربر اسناد

[۲۰] Operational Transformation

خود را به گونه ای تغییر دهند که از همگام سازی خارج شوند، چگونه می توانید ضمن حفظ هر دو تغییر به روشی معنی دار، آنها را به حالت اصلی برگردانید؟

سرانجام، OT به ما اجازه می دهد تا در هر زمان فقط دو عملیات را در نظر بگیریم. به جای اینکه در مورد چگونگی تأثیر تغییرات کاربر A بر روی سند کاربر B پس از اینکه کاربر B تغییرات اخیر کاربر C را اعمال کند، ابراز نگرانی کنیم، ما فقط یک سند کاربر و سند اصلی را در هر زمان مقایسه می کنیم. این مورد نه تنها ساده تر است، بلکه دارای موارد تضاد کمتری است، همینطور به طور مستقیم به ساختار Server / CLient وب ترسیم می شود.

۲-۳-۳ الگوریتم OT چگونه کار میکند ؟

مروری کوتاه بر نحوه کار :

- هر تغییر (درج یا حذف) به عنوان یک عمل نشان داده می شود. عملیات را می توان برای سند فعلی اعمال کرد که منجر به حالت جدید سند می شود
- برای مدیریت عملیات همزمان، ما از عملکرد Tranform استفاده می کنیم که دو عملی را که در همان حالت سند اعمال شده اند (اما روی کلاینت های مختلف) تغییر می دهد و یک عملیات جدید را محاسبه می کند که می تواند بعد از عملیات دوم اعمال شود و تغییر در نظر گرفته شده اولین عملیات را حفظ می کند.

سیستم OT وضعیت سند را به عنوان History Log نشان می دهد، که در قلب دنباله ای از عملیات‌ها است. نکته اصلی در OT این است که هر عملی دارای یک Context[۲۱] است، و Context ها همیشه با هم مطابقت دارند. هنگام ادغام عملیات با حالت سند گره جدید، Context گره با Context کلی مطابقت ندارد، بنابراین عملیات باید تغییر(Transform) پیدا کند

علامت استاندارد برای این عمل IT[۲۲](Oa, Ob) است، که Oa و Ob دارای Context یکسانی هستند و یک O'a تبدیل و تغییر داده شده ارائه می دهند. این عملیات تبدیل شده علاوه بر این اینکه عملیات‌های Ob را در Context خود دارد، همان کار Oa اصلی را نیز انجام می دهد.

کمی بیشتر به جزئیات وارد میشویم؛ یک مدل نظیر به نظیر(Peer-to-Peer) را در نظر بگیرید که هر گره کپی مخصوص به خود را از حالت سند داشته باشد. هر گره به طور مداوم به تمام همسایگان خود به روزرسانی‌های

---

[۲۱] اساسا مجموعه عملیاتی که قبل از آن در log سیستم ثبت میشود

[۲۲] Inclusive Transformation

خود را ارسال می کند (توپولوژی می تواند ستاره ،یک درخت پوشا یا یک گراف کامل باشد). فرض ما بر این است که ارتباطات منظم است، اما مطمئنا با تاخیرهای غیرقابل پیش بینی (اساساً همان مدل ارتباطی TCP / IP و WebSockets) همراه است.

هنگامی که یک گره از گره دیگری به روزرسانی می کند، قبلاً همه به روزرسانی های قبلی را دریافت کرده است (به لطف ماهیت مرتب کانال ارتباطی)، و آنها را در حالت سند خود ادغام کرده است. حالت سند این گره شامل تمام عملکردهای حالت سند گره دیگر است، به علاوه سایر عملکردهایی که گرهای دیگر در زمان ارسال به روزرسانی (هنوز) مشاهده نکرده است.

Transformation Function ۳-۳-۳

برای مدیریت عملیات همزمان، ما از عملکرد Tranform استفاده می کنیم که دو عملیات را می گیرد که در همان حالت سند اعمال شده اند (اما روی کلاینت های مختلف) و یک عملیات جدید را محاسبه می کند که می‌تواند بعد از عملیات دوم اعمال شود و اولین عملیات را حفظ می کند.

در واقع، دو نوع عملکرد تغییر وجود دارد:

۱. Inclusion Transformation : که با IT(a,b) نمایش داده میشود که عملکرد a را در برابر عمل b تبدیل به عملیاتی جدید می کند به گونه ای که تأثیر b موثر باشد.
۲. Exclusion Transformation : که با ET(a,b) نمایش داده میشود و عملکرد a را در برابر عمل b تبدیل به عملیاتی جدید می کند به گونه ای که تأثیر b به طور موثر کنار گذاشته شود.

توابع Transform در سیستمهای مختلف OT متفاوت نامگذاری شده اند، و برخی از توابع Transform ترکیبی ممکن است هر دو ویژگی IT و ET را در یک عملکرد ترکیب کنند.

OT، در هسته اصلی، یک مکانیسم کنترل همزمانی خوش بینانه است. این اجازه را به دو ویرایشگر می دهد تا همزمان یک بخش از یک سند را بدون هیچ گونه تعارضی اصلاح کنند. یا بهتر بگوییم، مکانیزمی را برای حل معقولانه آن تعارضات فراهم می کند تا نه مداخله کاربر و نه قفل شدن لازم باشد

این در واقع یک مشکل دشوارتر از آن است که به نظر می رسد. تصور کنید که عملیات زیر را داریم :

```
۱. insertCharacters('go')
```

حال تصور کنید که ما دو ویرایشگر داریم که نشانگرهای آنها در انتهای سند قرار گرفته است. آنها به طور همزمان یک t و یک کاراکتر a وارد می کنند (به ترتیب). بنابراین، ما دو عملیات ارسال شده به سرور خواهیم داشت. مورد اول ۲ مورد را نگه می دارد و یک t را وارد می کند، مورد دوم ۲ مورد را حفظ می کند و a را وارد می کند. به طور طبیعی، سرور باید برخی از موارد یکپارچه بودن ویرایش ها را اجرا کند، بنابراین ابتدا یکی از این عملیات اعمال خواهد شد. با این حال، به محض اعمال هر یک از این عملیات ها، ذخیره برای دیگری نامعتبر است. بسته به ترتیب، متن سند حاصل یا "goat" یا "gota" خواهد بود.

آنچه در اینجا داریم یک مسئله ساده یک مرحله ای است. در مطالعه تئوری OT، ما به طور کلی این وضعیت را با استفاده از نمودارهایی مانند شکل زیر تجسم می کنیم:

دیاگرام ۵ نمودار اولیه تحول عملیاتی

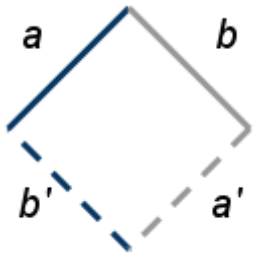
روشی که باید نمودارهای این چنینی را بخوانید به صورت نمایش گرافیکی عملیات روی دو سند به طور همزمان است. عملیات Client سند را به سمت چپ منتقل می کند. عملیات سرور سند را به سمت راست می برد. هر دو سرویس گیرنده و سرور سند را به سمت پایین حرکت می دهند. بنابراین، نمودارهای این چنینی به ما اجازه می دهد تا کاربرد عملیات را در یک "فضای حالت" واقعی تحت تأثیر قرار دهیم. خط آبی تیره مسیر مشتری را از طریق فضای حالت نشان می دهد، در حالی که خط خاکستری مسیر سرور را نشان می دهد. رئوس این مسیرها نقاطی در فضای حالت هستند، که نمایانگر حالت خاصی از سند هستند. وقتی هر دو سرویس گیرنده و خط سرور از یک نقطه عبور می کنند، به این معنی است که محتوای اسناد مربوطه آنها حداقل در آن مقطع زمانی خاص همگام بوده است.

برای رفع مشکل ذکر شده ما می توانیم این کار را با استفاده از یک تبدیل طبق الگوریتم Google’s OT انجام دهیم:

$$\text{xform}(a, b) = (a', b'), \text{ where } b' \circ a \equiv a' \circ b$$

کد ۲ تحول عملیات Google

به زبان ساده، این بدان معناست که تابع Transform دو عمل انجام می دهد، یک سرور و یک کلاینت و یک جفت عمل تولید می کند. این عملیات را می توان در حالت نهایی همتای خود اعمال کرد تا دقیقاً همان حالت را به صورت کامل تولید کند. از لحاظ گرافیکی ، می توانیم موارد این را با شکل زیر نشان دهیم:



دیاگرام ۶ نمودار نهایی تحول عملیاتی

دوباره به مثال عینی خود برگردیم و در نهایت می توانیم مشکل "goat" در مقابل "gota" را حل کنیم. ما با شرایطی شروع می کنیم که مشتری A با استفاده از متن سندی از "got" ، عملیات a را اعمال کرده است. اکنون عملیات b را از سرور دریافت می کند و به آن دستور می دهد تا بیش از ۲ مورد را حفظ کرده و کاراکتر "a" را وارد کند. با این حال، قبل از اینکه این عملیات را اعمال کند (که بدیهی است منجر به اشتباه بودن حالت سند می‌شود)، برای استخراج عملیات b′ از OT استفاده می کند. اجرای GoogleOT تعارض بین "t" و "a" را به نفع سرور برطرف می کند. بنابراین، b′ از اجزای زیر تشکیل خواهد شد:

۱. retain(۲)
۲. insertCharacters('a')
۳. retain(۱)

مشاهده خواهید کرد که دیگر عدم تطابق سند نداریم، زیرا آخرین Retain() اطمینان از رسیدن مکان نما به انتهای حالت سند را تضمین می کند.

در همین حال، سرور عملیات ما را دریافت کرده و یک سری مراحل مشابه را برای استخراج عملیات a انجام میدهد. یک بار دیگر، OT گوگل باید تعارض بین "t" و "a" را به همان روشی که درگیری برای مشتری A را حل کرده است، حل کند. ما در حال تلاش برای اجرای عملیات a (که کاراکتر "t" را در موقعیت ۲ قرار می دهد) به حالت سند سرور هستیم، که در حال حاضر "goa" است. وقتی کار ما تمام شد، باید محتوای دقیقاً مشابه سند

سمت Client کاربر A را به دنبال استفاده از b’ داشته باشیم. به طور خاص، حالت سند سرور باید "goat" باشد. بنابراین، فرآیند OT عملیات a’ متشکل از اجزای زیر را ایجاد می کند:

۱. retain(۳)
۲. insertCharacters('t')

Client کاربر A عملیات b را به حالت سند خود اعمال می کند، سرور عملیات a را به حالت سند خود اعمال می کند، و هر دو به سندی متشکل از متن "goat" می رسند.

در OT همه چیز در مورد عملکرد تبدیل و نحوه رفتار آن در این شرایط متفاوت است. همانطور که مشخص شد، این همه کارهایی است که OT به تنهایی برای ما انجام می دهد. OT در واقع فقط یک ابتکار همزمان است.

Character wise Transformation Function ۳-۳-۴

الگوریتم تابع تغییر شکل کاراکتر (برای نگهداری همسان بودن)، مدلی ساده است. به عنوان مثال، برای یک جفت عملیات منطبق بر کاراکتر Ins[p, c] (برای قرار دادن یک کاراکتر c در موقعیت p) و Del [p] (برای حذف یک کاراکتر در موقعیت p)، چهار عملکرد IT ، نشان داده شده به عنوان Tii، Tid، Tdi، Tdd را میتوان به صورت زیر تعریف کرد:

```
Tii(Ins[p1,c1], Ins[p2, c2]) {
      if p1 < p2  or (p1 = p2 and u1 > u2) // breaking insert-tie using user identifiers (u1, u2)
          return Ins[p1, c1];  // e.g. Tii(Ins[3, "a"], Ins[4, "b"]) = Ins[3, "a"]
      else return Ins[p1+1, c1]; } // Tii(Ins[3, "a"], Ins[1, "b"]) = Ins[4, "a"]

Tid(Ins[p1,c1], Del[p2]) {
      if p1 <= p2 return Ins[p1, c1]; // e.g. Tid(Ins[3, "a"], Del[4]) = Ins[3, "a"]
     else return Ins[p1-1, c1]; } // Tid(Ins[3, "a"], Del[1] ) = Ins[2, "a"]

Tdi(Del[p1], Ins[p2, c2]) {
      if p1 < p2 return Del[p1];  // e.g.  Tdi(Del[3], Ins[4, "b"]) = Del[3]
      else return Del[p1+1]; } // Tdi(Del[3], Ins[1, "b"]) = Del[4]

Tdd(Del[p1], Del[p2]) {
      if p1 < p2 return Del[p1]; // e.g.   Tdd(Del[3], Del[4]) = Del[3]
      else if p1 > p2 return Del[p1-1]; // Tdd(Del[3], Del[1]) = Del[2]
      else return I; } // breaking delete-tie using I (identity op)  Tdd(Del[3]. Del[3]) = I
```

کد ۳ توابع تغییر شکل عملیات بر روی کاراکتر

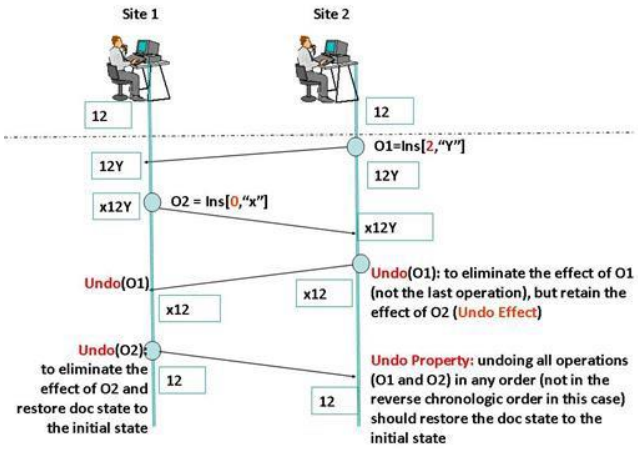
الگوریتم تابع تبدیل رشته ای به طور قابل توجهی چالش برانگیزتر از عملکردهای کاراکتر است، زیرا:

- یک رشته حذف شده یک محدوده حذف را پوشش می دهد، که ممکن است شامل کاراکترهای رشته و همچنین موقعیت های فاصله بین کاراکترها باشد.

- عملیات حذف رشته همزمان ممکن است خودسرانه با یکدیگر و حتی با عملیات درج همزمان همپوشانی داشته باشد.
- رشته ای که توسط یک عملیات درج قبلی وارد شده است، ممکن است با عملیات درج و حذف تغییر کند.

سیستم OT نیز برای رفع نیاز کاربران از Undo پشتیبانی میکند که نیازمندیهای عملیاتی خود را میطلبد:

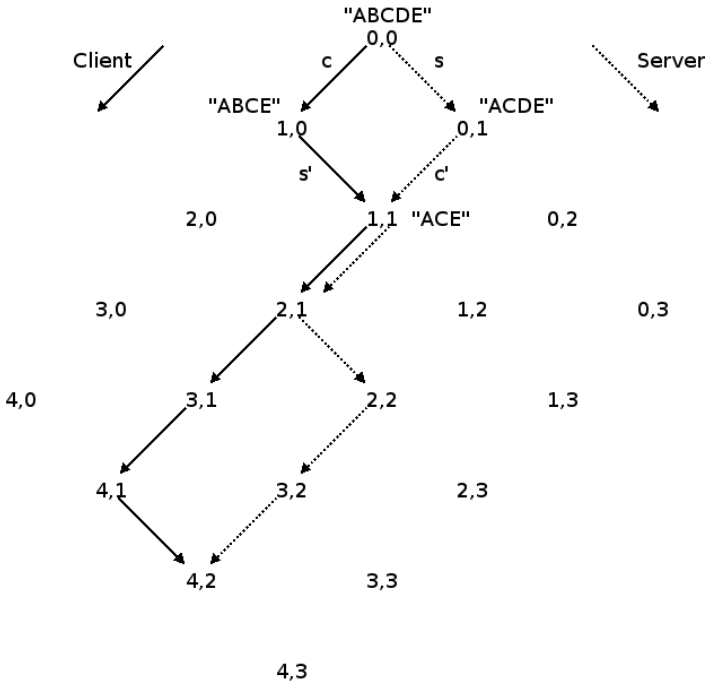
- یکی اثر خنثی سازی (Undo) است، که مستلزم آن است که با خنثی کردن یک عملیات O اثر حذف O حاصل شود اما اثرات سایر عملیات‌ها را در سند حفظ کند. به عبارت دیگر، اثر خنثی سازی O این است که حالت سند را به حالتی تبدیل کند که اگر O هرگز انجام نشده بود اما سایر عملیات انجام میشد، به آن حالت می رفت. این اثر خنثی سازی با اثر خنثی سازی خطی در محیط های ویرایش تک کاربر سازگار است و همچنین برای Undo غیر خطی (به عنوان مثال "لغو انجام هر عملی در هر زمان") در محیط های ویرایش چند کاربره و همزمان مناسب خواهد بود.
- مورد دیگر ویژگی لغو است، که مستلزم آن است که سند با لغو تمام عملیات انجام شده پس از آن حالت، بدون در نظر گرفتن ترتیب انجام این عملیات، به حالت قبلی برگردانده شود. این ویژگی برای اطمینان از قابلیت "بازیابی هر حالت قبلی" مورد نیاز است .



تصویر ۴ تعامل همزمان دو کاربر

Client — Server Acknowledgement ۳-۳-۵

Client می تواند عملیات ها را به ترتیب به سرور ارسال کند و بالعکس این اتفاق نیز می‌افتد. این بدان معناست که سرویس گیرنده و سرور می توانند فضای مختلف را از طریق مسیرهای مختلف تبدیل عملیاتی به همان حالت همگرا، بسته به زمان دریافت سایر عملیات ، طی کنند.



تصویر ۵ تعامل سرور و کلاینت

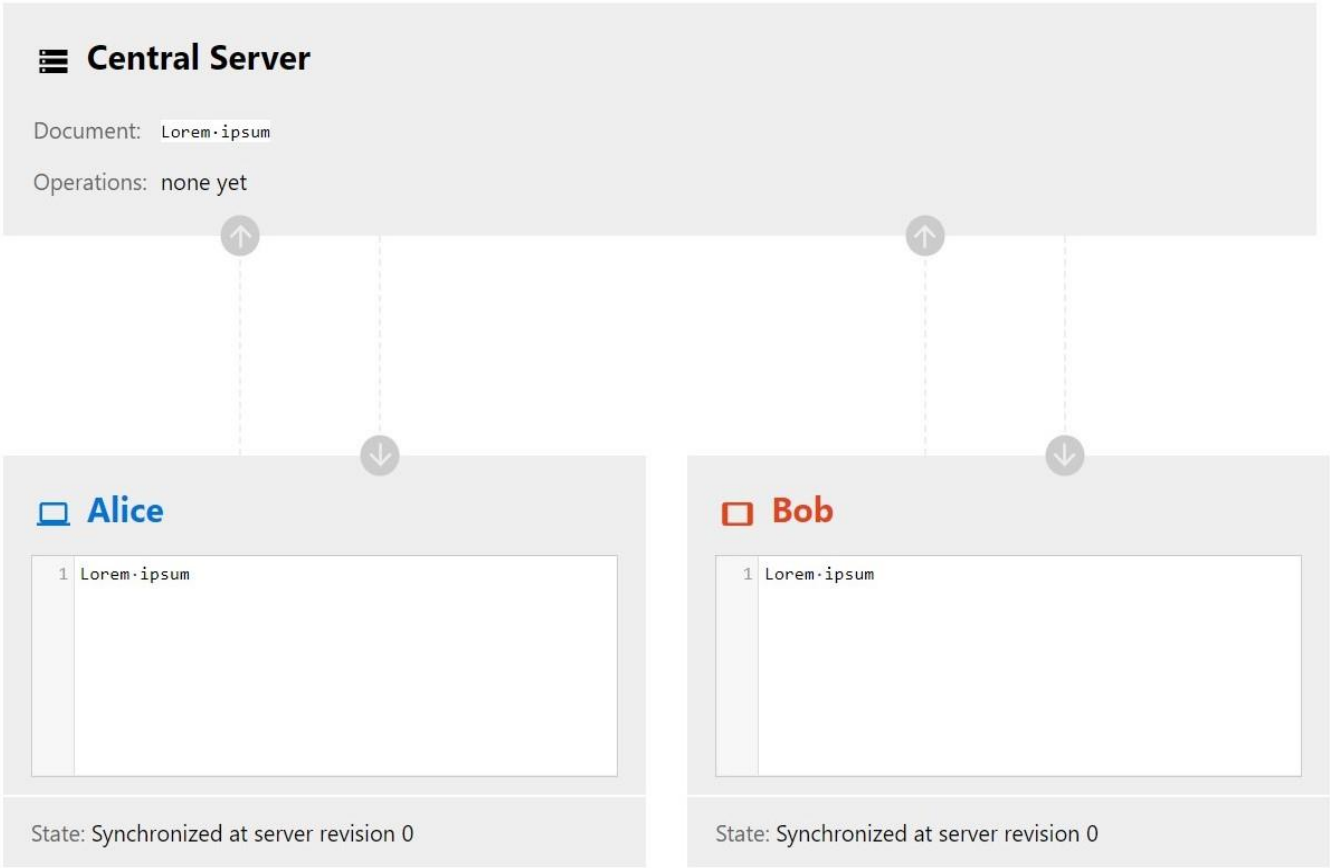
هنگامی که چندین Client به سرور متصل می شوند، هر جفت Client و سرور فضای حالت خاص خود را دارند. یک نقص در این مورد این است که سرور برای هر سرویس گیرنده متصل باید یک فضای خالی داشته باشد، که می تواند بسیار حافظه ساز باشد. بعلاوه، این الگوریتم سرور را مجبور می کند تا عملیات مشتری را بین فضاهای حالت تبدیل کند. داشتن یک سرور ساده و کارآمد در قابل اعتماد و مقیاس پذیر شدن مهم است. با این هدف، الگوریتم Operational Transformation متعلق به Google نظریه اصلی OT را اصلاح می کند و از Client می‌خواهد قبل از ارسال عملیات‌های بیشتر منتظر تأیید[۲۳] از سرور باشد. هنگامی که یک سرور عملکرد Client را تأیید می کند، به این معنی است که سرور عملکرد Client را Transform داده است، آن را روی نسخه سرور اعمال کرده و عملیات تبدیل شده را برای سایر سرویس گیرنده های متصل پخش میکند. در حالی

[۲۳] Acknowledgement

که مشتری Client تأیید است، عملیات تولید شده به صورت محلی را ذخیره می کند و بعداً آنها را به صورت عمده ارسال می کند.

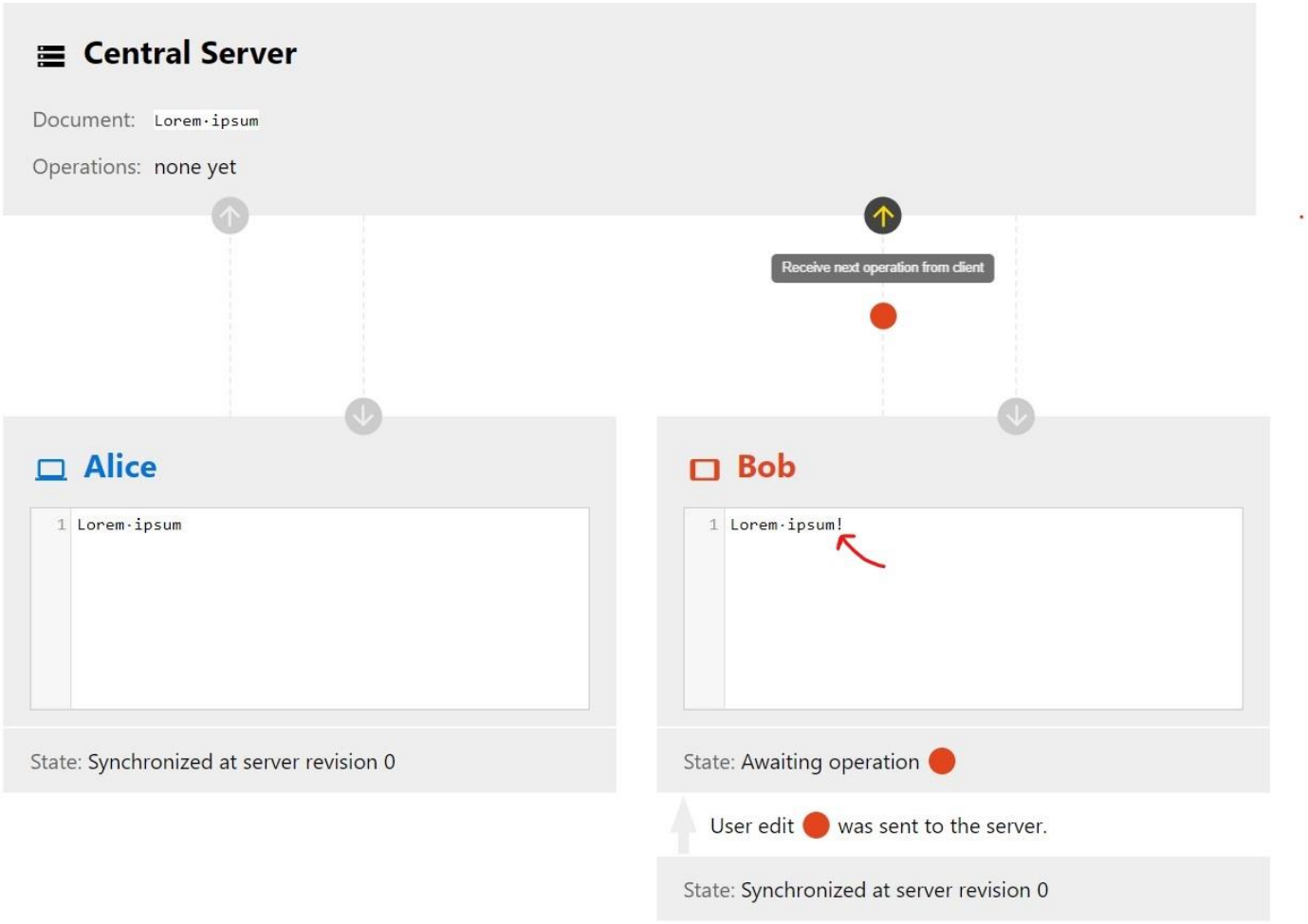
۳-۴ شبیه سازی کارکرد OT

- **حالت اولیه:** بدون تغییر



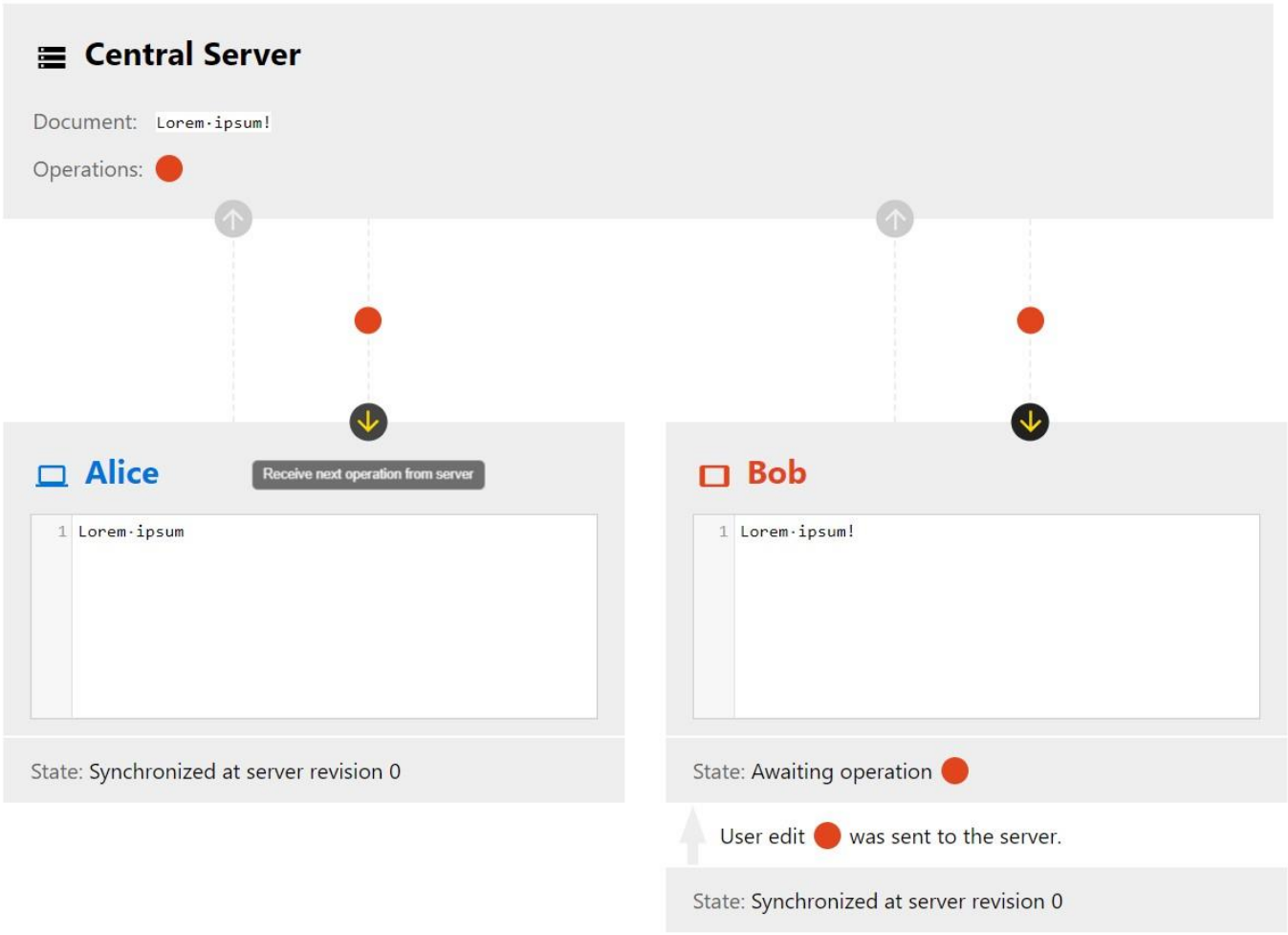
تصویر ۶ شبیه سازی OT مرحله ۱

- **ورود کاراکتر:** اطلاعات عملیات انجام گرفته به سرور ارسال میشود و Client منتظر تاییده تغییرات ارسالی و یا عملیات‌های دیگر از سمت سرور میماند.



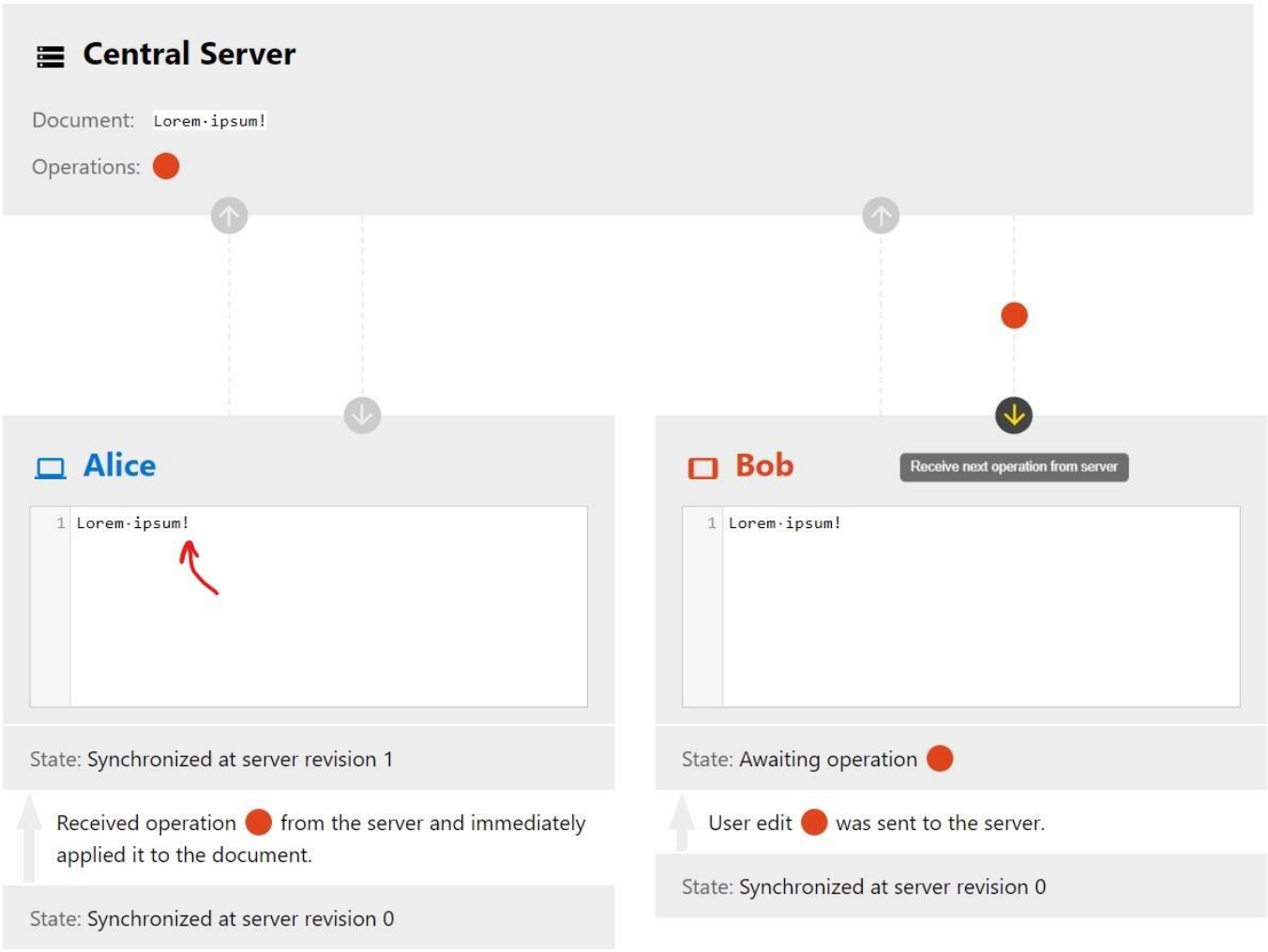
تصویر ۷ شبیه سازی OT مرحله ۲

- **انتقال عملیات از سرور به مقصد:** اطلاعات عملیات انجام شده از سرور به Client مقصد ارسال میشود



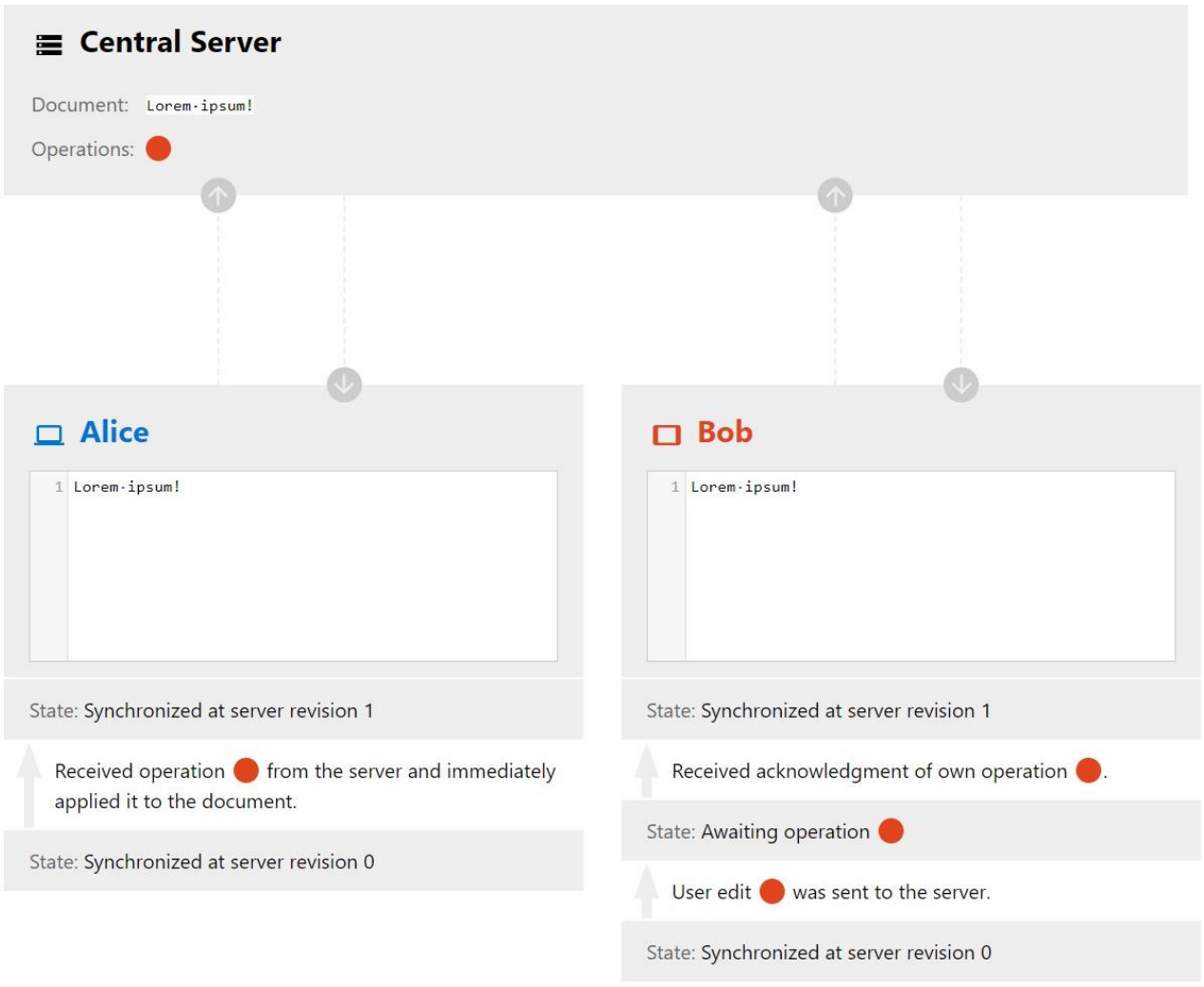
تصویر ۸ شبیه سازی OT مرحله ۳

- **دریافت و اعمال عملیات در مقصد و ارسال Acknowledgement به مبدا:** در Client مقصد پس از دریافت عملیات‌ها بلافاصله (در صورت امکان) تغییرات لازم انجام شده و تاییدیه انجام تغییرات از طریق سرور به مبدا ارسال میشود.



تصویر ۹ شبیه سازی OT مرحله ۴

- **تایید اعمال تغییرات در مبدا:** تاییدیه اعمال تغییرات در سمت مقصد از طریق سرور بدست مبدا میرسد



تصویر ۱۰ شبیه سازی OT مرحله ۵

فصل چهارم:پیاده سازی

پلتفرم Code۲Gether

تصویر ۱۱ معرفی پلتفرم code2gether

۴-۱ معرفی پلتفرم

پلتفرم پیاده سازی شده code۲gether یک پلتفرم ویرایشگر اشتراکی کد برای پروژه‌های تسعه وب به زبان‌های Html/Css/Javascript میباشد که به کاربران این امکان را میدهد که بصورت Real-Time با همکاری یکدیگر بر روی پروژه فعالیت داشته و همچنین نتیجه اجرا شده کدهای خود را بصورت Real-Time مشاهده نمایند. در این پلتفرم کاربران قادر هستند با یکدیگر بصورت مکالمه چت یا مکالمه تصویری نیز با یکدیگر ارتباط برقرار کنند.

۴-۲ تکنولوژی‌های مورد استفاده

در این پلتفرم از تکنولوژی‌های متفاوتی جهت پیاده‌سازی ویرایشگر، برقراری ارتباط و همزمانی استفاده شده است که در زیر به آنها اشاره میشود :

۴-۲-۱ NodeJs

Node.js به عنوان یک asynchronous event-driven JavaScript runtime، برای ساخت برنامه های شبکه مقیاس پذیر طراحی شده است. در مثال "Hello World" زیر، بسیاری از اتصالات می توانند همزمان کار کنند. پس از هر اتصال، تماس مجدد برقرار می شود، اما اگر کاری برای انجام وجود نداشته باشد، Node.js به حالت Sleep میرود.

این در تضاد با مدل همزمانی رایج امروزی است که در آن از نخ[۲۴] های سیستم عامل استفاده می شود. بعلاوه، کاربران Node.js از این که قفلی وجود ندارد، نگران قفل کردن مراحل کار نیستند. تقریباً هیچ عملکردی در Node.js مستقیماً ورودی / خروجی را انجام نمی دهد، بنابراین روند هرگز مسدود نمی شود. از آنجا که هیچ چیز مسدود نمی شود، توسعه سیستم های مقیاس پذیر در Node.js بسیار منطقی است.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

کد ۴ راه اندازی سرور nodejs

۴-۲-۲ ExpressJs

Express یک چارچوب نرم افزار وب تحت Node.js بسیار انعطاف پذیر است که مجموعه ای قوی از ویژگی های وب و برنامه های تلفن همراه را برای ما فراهم می کند. در این تکنولوژی با وجود تعداد بیشماری از روشهای ابزار HTTP و میان افزار موجود، ایجاد یک API قوی و سریع ، امری آسان است. Express یک لایه نازک از ویژگیهای اساسی برنامه وب را فراهم می کند، بدون اینکه ویژگی های Node.js را پنهان کند. بسیاری از فریم ورک های محبوب بر اساس Express ساخته شده اند.

۴-۲-۳ Handlebars

هندلبارز یک زبان ساده برای الگوسازی (Templating Language) است.

از این الگو و یک شی ورودی برای تولید HTML یا قالب های متن دیگر استفاده می کند.

---

[۲۴] Threads

عبارت فرمان Handlebars بصورت {{"محتویات"}} است . هنگامی که الگو اجرا می شود، این عبارات با مقادیر از یک شی ورودی جایگزین می شوند.

Handlebars قدرت لازم را برای ساختن الگوهای معنایی موثر فراهم می کند. Handlebars تا حد زیادی با الگوهای Mustache سازگار است. در بیشتر موارد، می توان Mustache را با Handlebars عوض کرد و به استفاده از الگوهای فعلی ادامه داد. Handlebars الگوها را در توابع JavaScript کامپایل می کند. این باعث می شود که اجرای الگو از سایر موتورهای الگو سریعتر باشد.

MongoDB ۴-۲-۴

MongoDB یک پایگاه داده اسناد است، به این معنی که داده ها را در اسناد مشابه JSON ذخیره می کند. ما اعتقاد داریم که این طبیعی ترین روش برای فکر کردن در مورد داده است، و بسیار رساتر و قدرتمندتر از مدل سنتی ردیف / ستون است.MongoDB یک پلتفرم واقعی داده با مجموعه ای گسترده از ابزارها است که کار با داده ها را برای همه بسیار آسان می کند، از توسعه دهندگان گرفته تا تحلیلگران.

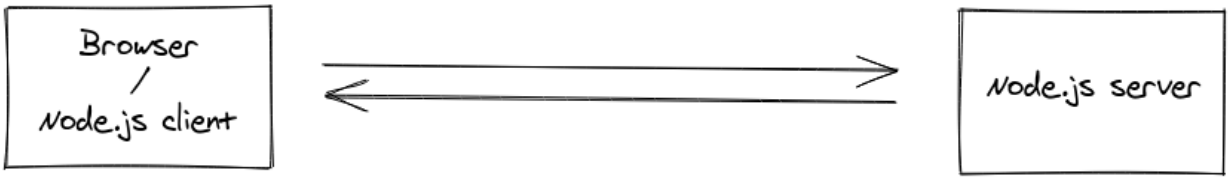
CodeMirror ۴-۲-۵

CodeMirror یک ویرایشگر متن همه کاره است که با زبان JavaScript برای مرورگرها پیاده سازی شده است. این برنامه برای ویرایش کد تخصصی است و دارای چندین حالت زبان و برنامه افزودنی است که قابلیت ویرایش پیشرفته تری را ارائه می کنند. یک API برنامه نویسی غنی و یک سیستم Theme به زبان CSS برای سفارشی سازی CodeMirror متناسب با برنامه شما و گسترش آن با قابلیت های جدید در دسترس است.

Socket.IO ۴-۲-۶

Socket.IO کتابخانه ای است که امکان برقراری ارتباط در زمان واقعی، دو طرفه و مبتنی بر رویداد را بین مرورگر و سرور فراهم می کند. این شامل:

- یک سرور Node.js
- یک کتابخانه Client به زبان Javascript برای مرورگر (که از Node.js نیز قابل اجرا است)



تصویر ۱۲تعامل NodeServer و NodeClient

۴-۲-۵-۱ Socket.IO چگونه کار میکند؟

Client در صورت امکان سعی در برقراری اتصال WebSocket دارد و در غیر اینصورت به HTTP-Long-Polling باز می گردد.

WebSocket یک پروتکل ارتباطی است که یک کانال دو طرفه و با تأخیر کم بین سرور و مرورگر را فراهم می کند. بنابراین ، در بهترین حالت، به شرطی که:

- مرورگر از WebSocket پشتیبانی کند (۹۷٪ از کل مرورگرها در سال ۲۰۲۰)
- هیچ عنصری (پروکسی ، فایروال ، ...) وجود نداشته باشد که مانع از اتصال WebSocket بین client و سرور شود

شما می توانید Socket.IO را به عنوان یک Wrapper در اطراف WebSocket API در نظر بگیرید

Socket.IO یک پیاده سازی WebSocket نیست. اگرچه Socket.IO در صورت امکان از WebSocket به عنوان یک ابزار حمل و نقل استفاده می کند، در عین حال فراداده اضافی به هر بسته اضافه می کند. به همین دلیل یک سرویس گیرنده WebSocket نمی تواند با موفقیت به یک سرور Socket.IO متصل شود. سرویس گیرنده IO نیز نمی تواند به یک سرور WebSocket ساده متصل شود.

۴-۲-۶ OTJs

پیرامون این تکنولوژی در فصل گذشته توضیحات لازم ارائه شده است.

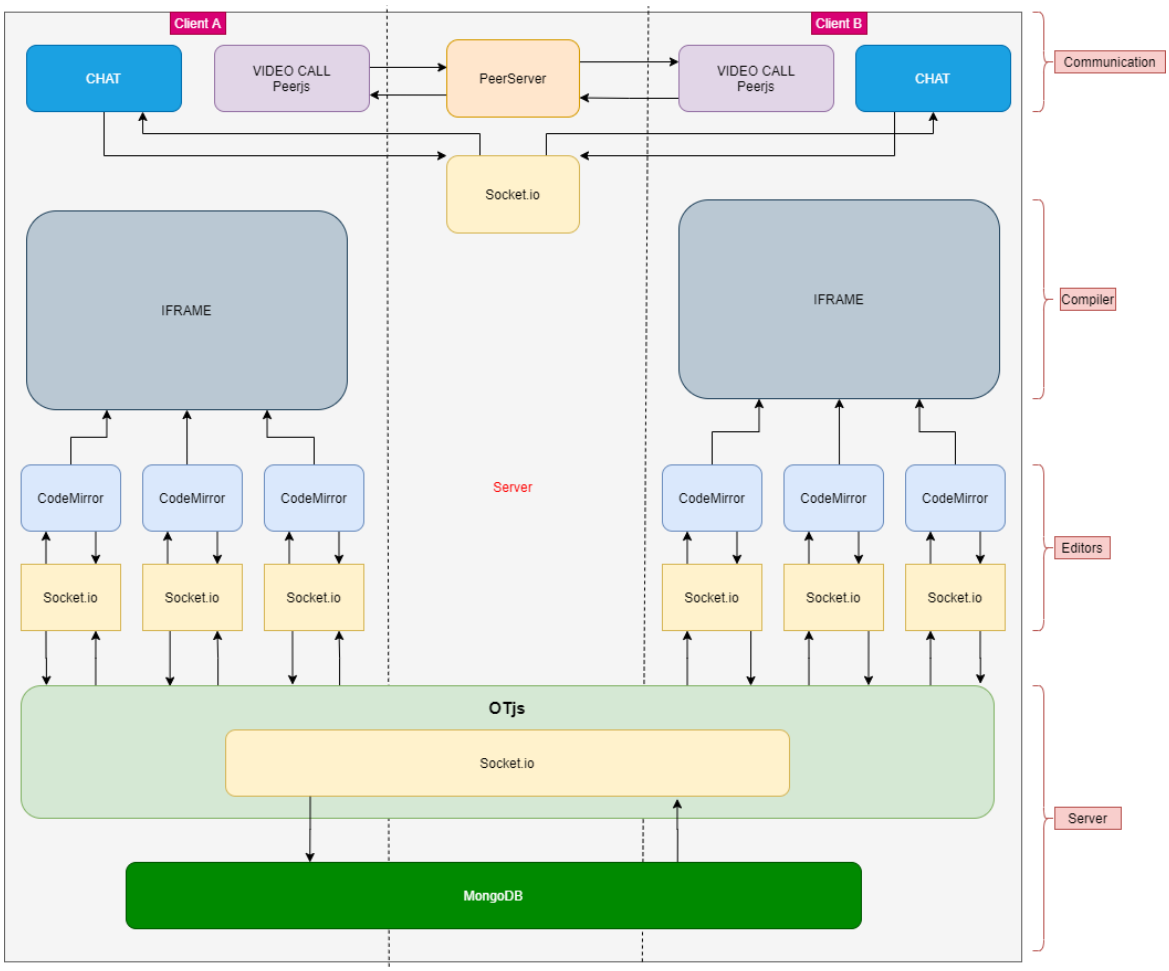
PeerJs ۴-۲-۷

PeerJS پیاده سازی WebRTC تحت مرورگر را ارائه می دهد تا یک رابط برنامه کاربردی peer-to-peer کامل، قابل تنظیم و با کاربردی آسان داشته باشد. تحت این تکنولوژی کاربران مجهز به شناسه(ID)، می‌توانند داده های P۲P[۲۵] یا اتصال جریان رسانه ای را با یک همکار از راه دور به اشتراک بگذارند. برای ارتباطات واسطه، PeerJS به PeerServer متصل می شود. توجه داشته باشید که هیچ داده ای نظیر به نظیر از سرور عبور نمیکند. سرور فقط به عنوان یک کارگزار ارتباط عمل می کند.

۴-۳ نمونه اولیه[۲۶]

در مرحله اول، یک نمونه اولیه ساده ایجاد میکنیم که تکنولوژی‌های انتخابی مورد استفاده را ترکیب می‌کند. این نمونه اجازه می دهد تا یک برانداز سریع از معماری نرم‌افزار بدون نیاز به مواجهه با پیچیدگی ساخت آن در کدنویسی، موجود باشد. با کمک این نمونه میتوان تعامل و ارتباط ماژول‌های متفاوت استفاده شده و نحوه اتصال آنها را به راحتی مشاهده و بررسی کرد. نمونه آورده شده در زیر یک نمونه بسیار ساده از شماتیک و طراحی معماری نرم افزار پیش روی شماست :

[۲۵] Peer To Peer
[۲۶] Prototype



دیاگرام ۷ پروتوتایپ نحوه کارکرد و اتصالات ماژول های نرم افزار

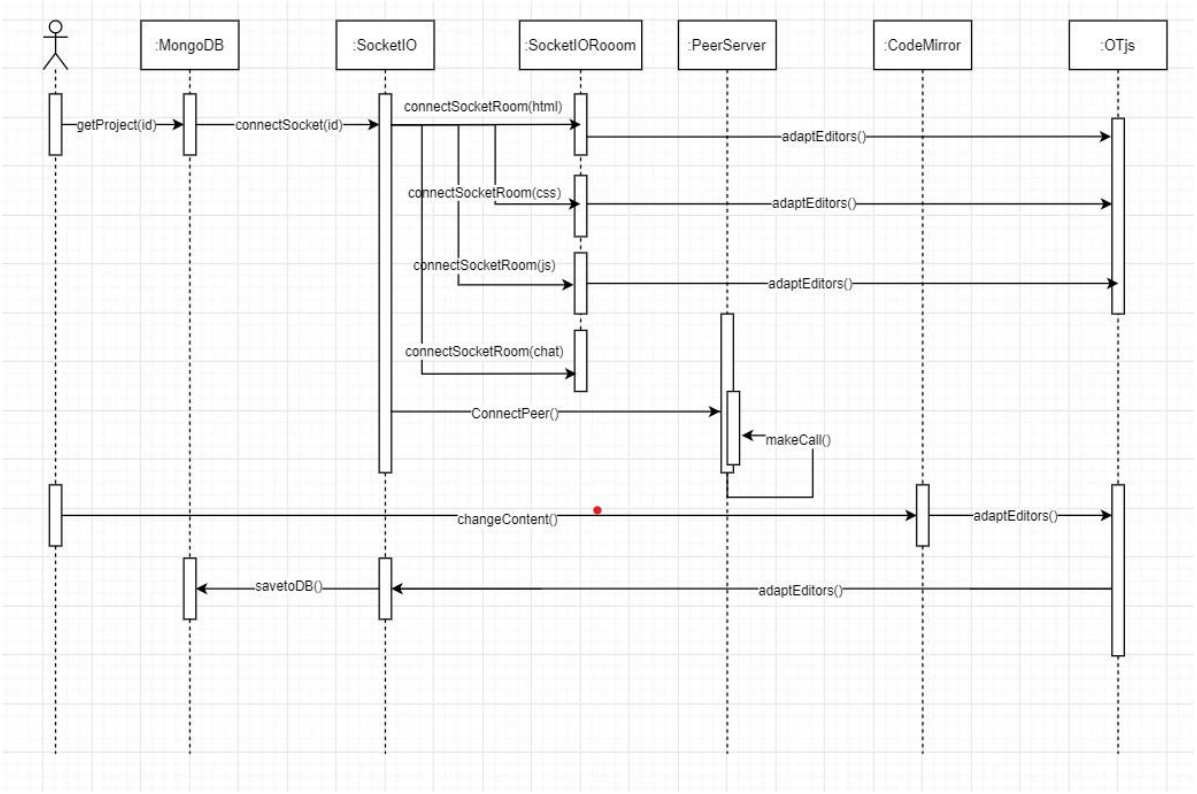
بطور کلی هنگام ورود به صفحه پروژه یک سوکت با شناسه پروژه ایجاد شده و از این طریق کاربران دیگر نیز میتوانند با اتخاب این پروژه به این سوکت متصل شده و بصورت اشتراکی به ویرایش بپردازند؛ به زبان ساده کل معماری بالا را میتوان داخل یک سوکت در نظر گرفت.

همانطور که در معماری بالا مشاهده میشود سیستم ویرایش متشکل از ۳ عدد ویرایشگر CodeMirror برای زبان‌های Html/Css/Javascript است که هریک از این ویرایگرها توسط یک سوکت، جهت اِعمال امکان ویرایش همزمان، به سیستم OT متصل میشوند که OT برای هر اتصال یک به اصطلاح Socket Room ایجاد میکند و از طریق آن با ویرایشگر ارتباط برقرار میکند. سپس OT پس از Sync کردن اطلاعات، کدهای وارد شده در ویرایشگر ها را، برای دسترسی‌های آینده، درون دیتابیس ذخیره مینماید.

از سمت دیگر در سیستم مکالمه چت رابط کاربری چت از طریق سوکت پیام‌ها را در Socket Room اختصاص داده شده به پروژه به اشتراک میگذارد تا توسط دیگر کاربران حاضر در این Room قابل مشاهده باشد.

در سمت دیگر، سیستم مکالمه ویدیویی توسط اتصال Peer به سرور PeerServer پیاده‌سازی شده متصل میشود و داده دریافتی از میکروفن و دوربین کاربر را دریافت و منتظر تماس میشود. پس از برقراری تماس به کمک شناسه‌های اختصاصی کاربران(ID)، داده‌های تماس از طریق این سرور بین دو کاربر در حال تماس به اشتراک گذاشته میشود.

برای روشنی بیشتر Sequence Diagram زیر با توجه به روند ویرایش و برقراری ارتباط به درک مطلب کمک میکند:

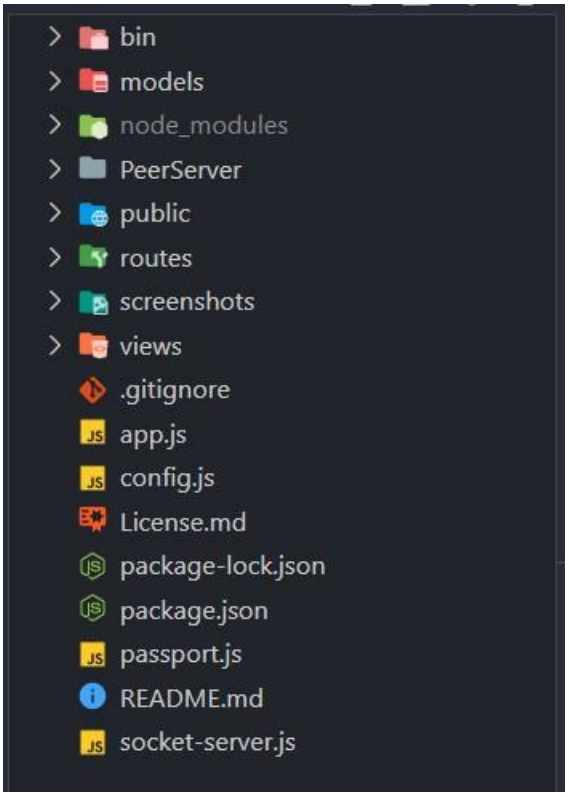


دیاگرام ۸ نمودار Sequence Diagram نرم افزار

۴-۴ پیاده‌سازی

۴-۴-۱ ساختار کلی پروژه

ساختار کلی پروژه به شکل زیر میباشد که به بررسی قسمت های مهم آن میپردازیم :



تصویر ۱۳ ساختار پروژه

- */Bin*

فایل‌های موجود در این پوشه مربوط به اجرا و راه اندازی سرور پروژه میباشند

- */models*

مدل‌های ذخیره سازی داده در این پوشه قرار دارند؛ این مدل‌ها شامل مدل‌های ذخیره‌سازی داده‌های کاربر و پروژه میباشند

- */node_modules*

ماژول‌ها و کتابخانه‌های استفاده شده در پروژه پس از اجرای دستور npm install در این پوشه نصب و نگهداری میشوند

- */PeerServer*

سرور peerjs مربوط به امکان مکالمه ویدیوی در این پوشه قرار دارد که با دستور npm start قابل اجراست و کاربران میتوانند به این سرور که در port شماره ۹۰۰۰ اجرا میشود، متصل شوند

- */Public*

فایل‌های جانبی مورد نیاز پروژه از جمله آیکون‌ها، تصاویر، کدهای css و کدهای جاوااسکریپت جانبی (از جمله فایل مربوط به کدهای پیاده سازی و هماهنگ سازی ویرایشگرها و مکالمه چت) درون این پوشه قرار میگیرند

- */Routes*

فایل‌های مربوط به مسیریابی پروژه که مسیریابی پلتفرم در مرورگر را کنترل میکنند در این پوشه قرار دارند

- */screenshots*

نمونه تصاویر پلتفرم جهت نمایش در ریپازیتوری گیت هاب در این پوشه قرار دارند.

- */view*

فایل‌های Handlebars استفاده شده برای Frontend پروژه در این پوشه ذخیره شده‌اند

- *App.js*

کدهای مربوط به ساخت و تنظیم و Conifg سرور اصلی پروژه، همچنین اتصال سرور به پایگاه داده در این فایل ذخیره شده‌اند

- *Config.js*

اطلاعات مربوط به Config ایمیل و لینک اتصال پایگاه داده در این فایل ذخیره شده‌اند

- *Passport.js*

کدهای مربوط به پیاده‌سازی Authentication و User registration/User Login در این فایل ذخیره شده‌اند

- *Socket-server.js*

کدهای مربوط به ایجاد و تنظیم سوکت‌های مختلف مورد نیاز پروژه در این فایل ذخیره شده‌اند.

۲-۴-۴ بررسی پیاده‌سازی و کد

در این بخش به بررسی مراحل پیاده‌سازی و کدهای نوشته و استفاده شده در پلتفرم میپردازیم. در این بخش از بررسی کدهای پایه‌ای مربوط به راه‌اندازی، احراز هویت کاربران و رابط کاربری صرف نظر شده و به بررسی کارکردهای عملیاتی سیستم ویرایشگر مشارکتی میپردازیم.

در اولین مرحله در صفحه پروژه نیاز به درج سه عدد ویرایشگر داریم که د ر این پروژه از ویرایشگر CodeMirror استفاده شده است که پس از Import کردن کتابخانه این ویرایشگر به کمک تَگ <script> میتوان یک عنصر <textarea> ایجاد و به شمل زیر ویرایشگر را درون آن بارگذاری کرد:

```
// html editor value
var htmlVal;

// Html Editor
var editor = CodeMirror.fromTextArea(document.getElementById("code-screen"), {
  lineNumbers: true,
  theme: "monokai",
  mode: "htmlmixed",
  extraKeys: { "Ctrl-Space": "autocomplete" },
});
// set editor size      MO Naseri, 14 hours ago • commenting
editor.setSize(400, 190);
```

کد ۵ تعریف ویرایشگر

در مرحله بعد برای دریافت مقادیر وارد شده درون ویرایشگر پس از هر تغییر از رویداد(Event) change استفاده میکنیم که در صورت تغییر کردن مقدار teaxarea ، آن مقدار را به ما بازمیگرداند :

```
// get value of editor on content change
editor.on("change", function (cm, change) {
  htmlVal = cm.getValue();
  compile();
});
```

کد ۶ تعریف رویداد برای تغییر ویرایشگر

در بدنه این عملیات از تابعی با نام compile() استفاده میشود که وظیفه این تابع جمع آوری مقادیر وارد شده در ویرایشگرها و بارگذاری و اجرای آنها در یک عنصر <ifram> است که کاربر خروجی پروژه را در این عنصر مشاهده میکند:

```
// compile htm;/css/js values into the iframe element
function compile() {
  var html = htmlVal;
  var css = cssVal;
  var js = jsVal;
  // get iframe element
  var compiler = document.getElementsByTagName("iframe")[0];
  const source = `
  <html>
    <head><style>${css}</style></head>
    <body>
      ${html}
      <script>${js}</script>
    </body>
  </html>
`;
  compiler.srcdoc = source;
}
```

کد ۷ تابع کامپایل برای اجرای کدها درون Iframe

در قسمت بعد لازم است برای هر یک از ویرایشگرها یک سوکت متناظر برای برقراری ارتباط با سیستم OT ایجاد شود :

```
// FOR SERVER DEPLOYMENT
var socket = io("http://51.195.28.68:3000");
var socketCss = io("http://51.195.28.68:3000", {
  forceNew: true,
});
var socketJs = io("http://51.195.28.68:3000", {
  forceNew: true,
});
```

کد ۸ ایجاد سوکت های لازم برای ویرایشگرها

سپس قبل از هرکاری می‌بایست سرور سوکت پروژه را تنظیم و بارگذاری کنیم:

```
module.exports = function (server) {

  var io = socketIO(server);
  io.on("connection", function (socket) {
```

کد ۹ ایجاد سرور سوکت

برای راه اندازی این سرور سوکت نیاز است تا سرور اصلی پروژه را به عنوان ورودی به این function بدهیم، از این رو در فایل /bin/www پس از ایجاد سرور، متغیر سرور را به این عملیات پاس میدهیم:

```
/**
 * Create HTTP server.
 */

var server = http.createServer(app);

// connect socket-server to app-server
require('../socket-server')(server);
```

کد ۱۰ اتصال سرور Node به سرور سوکت

بعنوان قدم بعد لازم است تا هریک از سوکت‌های مربوط به ویرایشگرها به SocketRoom متناظر با شناصه منحصر به فرد متصل شوند، از این رو برای اعمال منحصر به فرد بودن از شناسه سوکت کاربر به علاوه عبارات html/css/js استفاده مینماییم :

```
// get generated roomId from elements
var roomId = $("#roomId").val();

// connect htlm socket to room
socket.emit("joinRoom", {
  room: roomId,
  docId: roomId,
  username: username,
  editor: "html",
});

// connect css socket to room
socketCss.emit("joinRoom", {
  room: roomId + "css",
  docId: roomId,
  username: username,
  editor: "css",
});

// connect js socket to room      You, a few seconds ago
socketJs.emit("joinRoom", {
  room: roomId + "js",
  docId: roomId,
  username: username,
  editor: "js",
});
```

کد ۱۱ اتصال سوکت هر ویرایشگر به یک Room

پس از اتصال سوکت‌های مربوط به ویرایشگرها به Room منحصر به فرد خود ، لازم است تا این اتصال در سرور سوکت بررسی و رسیدگی شود. همانطور که در تصویر زیر مشاهده میشود سوکت پس از اتصال به سرور سوکت از لحاظ نوع اتصال و نوع ویرایشگر مورد بررسی قرار گرفته و فعالیت‌های آن بر این اساس تعریف میشوند؛ سپس همانطور که مشاهده میشود این سوکت به اتصال به سیستم ot اختصاص داده میشود:

```
socket.on("joinRoom", function (data) {
  // check if socket-room exists
  if (!roomList[data.room]) {
    // check the type of connection if it is from html editor
    if(data.editor === 'html') {
      console.log('html');
      // create an exclusive socketServer fot the editor that is connected
      var socketIOServer = new ot.EditorSocketIOServer(
        // the document which editor is using
        html,
        // editor operations
        [],
        // document id
        data.room,
        // what happens when user writes in document
        function (socket, cb) {
          var self = this;
          // find the project by its document id and update it's data
          Project.findByIdAndUpdate(
            data.docId,
            // update html part of project
            { html: self.document },
            // check if there's any error      You, a few seconds ago • Uncommitted changes
            function (err) {
              if (err) return cb(false);
              cb(true);
            }
          );
        }
      );
    }
```

کد ۱۲ نحوه و پیاده سازی اتصال ویرایشگر به سوکت

داده‌های پروژه در پایگاه داده mongoDB بصورت زیر ذخیره میشوند :

```
var projectSchema = new mongoose.Schema({
  // project name
  name: String,
  //projct owner id
  owner: String,
  // html data
  html: String,
  // css data
  css: String,
  // js data       You, a few seconds ago • Uncommitted
  js: String
});
```

کد ۱۳ مدل ذخیره سازی پروژه

در صورت اتصال موفق سوکت به ot ، یک رویداد با نام "doc" بر روی سوکت انتشار می‌یابد که حاوی اطلاعات لازم برای متصل کردن ویرایشگر از طریق سوکت به سیستم ot میباشد.

```
// init html editor when socket data recieved on "doc" event
socket.on("doc", function (obj) {
  init(obj.str, obj.revision, obj.clients, new SocketIOAdapter(socket));
});
```

کد ۱۴ دریافت اطلاعات مرتبط با ویرایشگر از اتصال سوکت

همانطور که گفته شد از این اطلاعات دریافتی جهت متصل کردن ویرایشگر به ot استفاده میشود که این کار توسط عملیات init اتفاق می‌افتد که ویرایشگر را در قالب یک client از طریق سوکت متناظر به ot متصل میکند:

```
// init html editor with data fetched from database and connect it to otjs
function init(str, revision, clients, serverAdapter) {
  // if editor is empty
  if (!code) {
    // set default value to editor
    editor.setValue(str);
  }

  // create a ot client for editor      You, a few seconds ago • Uncommitted
  cmClient = new EditorClient(
    // document revisions
    revision,
    // other users(sockets) wolrking on this project
    clients,
    // ot server provided with socket connection
    serverAdapter,
    // adapt code mirror with ot
    new CodeMirrorAdapter(editor)
  );
}
```

کد ۱۵ آماده سازی ویرایشگر به کمک اطلاعات دریافتی از پایگاه داده توسط سوکت

۳-۴-۴ پیاده سازی مکالمه تصویری و چت

تا اینجای کار سرویس ویرایشگر مشارکتی ما شروع به کار خواهد کرد

سپس برای ایجاد امکان مکالمه چت پس از ایجاد عنصرهای مورد نیاز در صفحه نیاز هست تا ارتباط کاربران از طریق سوکت فراهم شود. در این راستا ابتدا برای پیام‌های کاربران یک قالب مشخص طراحی و فراهم میکنیم :

```
// generate user message html
var userMessage = function (name, text) {      MO Naseri, 8 days ago • beautofied :)
  return (
    '<li class="media" style="width: 100%;"> <div class="media-body" style="width: 100%;"> <div class="media" style="width: 100
    '<div class="media-body" style="width: 100%;">' +
    '<span style="color: green">' +
    "<b>" +
    name +
    "</b> : " +
    "</span>" +
    '<span style="color: white;">' +
    text +
    "</span>" +
    "</div></div></div></li>"
  );
};
```

کد ۱۶ قالب نمایش پیام های کاربران

سپس جهت ارسال پیام کاربر به کاربران دیگر ابتدا لازم است تا ورودی کاربر در قسمت پیام گرفته شده و سپس از طریق سوکت و رویداد "chatMessage" بر روی سوکت انتقال میابد

```
// send user message through socket
var sendMessage = function () {
  // get user input in message area
  var userMessage = $("#userMessage").val();
  // emit the message in socket via "chatMessage" event
  socket.emit("chatMessage", { message: userMessage, username: username });
  $("#userMessage").val("");
};
```

کد ۱۷ ارسال پیام کاربر بر روی سوکت

کاربران دیگر حاضر در این پروژه این پیام را از طریق سوکت خود و بر اثر رویداد "chatMessage" دریافت میکنند و در قالب از قبل مشخص شده در چت خود بارگذاری میکنند :

```
// get other users message on this event
socket.on("chatMessage", function (data) {
  // load the given message in chatbox area
  $("#chatbox-listMessages").append(userMessage(data.username, data.message));
});
```

کد ۱۸ دریافت پیام کاربران دیگر از طریق سوکت

حال که سیستم چت نیز فعال و بارگذاری شد، نوبت به تنظیم سیستم مکامله تصویری میرسد؛ در این قسمت مانند قسمت‌های گذشته پس از بارگذاری عنصرهای مورد نیاز در رابط کاربری، در ابتدا می‌بایست یک سرور Peer جداگانه برای اتصال کاربران تنظیم و ایجاد نمود. برای این امر با کمک nodejs یک سرور جدید بارگذاری کرده و تنظیمات peer را در آن اعمال میکنیم :

```
var express = require('express');
var path = require('path');
var http = require('http');
var cors = require('cors');
var errorhandler = require('errorhandler');
var ExpressPeerServer = require('peer').ExpressPeerServer;

var options = {
  debug: true,
  key: 'code2gether'
};

var app = express();
var server = http.createServer(app);

var port = process.env.PORT || '9000';

app.set('port', port);

app.use(cors());
app.use(express.static(path.join(__dirname, 'public')));
app.use('/peerjs', ExpressPeerServer(server, options));
app.use(errorhandler());

process.on('uncaughtException', function (exc) {
  console.error(exc);
});

server.listen(port);
console.log('Server Running On Port: ', port);
```

کد ۱۹ ایجاد سرور Peer برای مکالمه تصویری

سپس نیاز است تا در اپلیکیشن نیز به این سرور متصل شویم :

```
// PeerJS object
var peer = new Peer(username + roomId, {
    host: 'localhost',
    port: 9000,       MO Naseri, 7 days ago • Added Video Call
    key: 'code2gether',
    path: '/peerjs'
});
```

کد ۲۰ اتصال به سرور Peer

در آخر با کمک کتابخانه Peer میتوانیم اقدام به برقراری تماس یا قبول تماس از سایرین بکنیم که توسط قطعه کد زیر قابل پیاده سازی میباشد:

```
// Receiving a call
peer.on("call", function (call) {
    // Answer the call automatically (instead of prompting user) for demo purposes
    call.answer(window.localStream);
    step3(call);
});

peer.on("error", function (err) {
    alert(err.message);
    // Return to step 2 if error occurs
    step2();
});

// Click handlers setup
$(function () {
    $("#make-call").click(function () {
        // Initiate a call!
        var call = peer.call($("#callto-id").val(), window.localStream);
        step3(call);
    });
    $("#end-call").click(function () {
        window.existingCall.close();
        step2();
    });
    step1();
});
```

کد ۲۱اعمال تماس با شناسه خاص و پایان تماس

توابع Step استفاده شده در این عملیات‌ها برای کنترل رابط کاربری و نمایش یا عدم نمایش دکمه‌های تماس به کاربر میباشند .پس از پیاده‌سازی تمامی این قسمت‌ها پروژه بطور کامل قابل بارگذاری و استفاده خواهد بود.

۴-۴-۴ اجرای پروژه

۱. ابتدا مطمئن شوید که nodejs (و مدیر بسته node) و MongoDB را روی سیستم خود نصب کرده اید ، در غیر اینصورت می توانید آنها را از پیوندهای زیر دریافت کنید:

- Nodejs

https://nodejs.org/en/download/

- MongoDB

https://www.mongodb.com/try/download/community

۲. مخزن(Repository) پروژه clone یا بارگیری کنید:

```
git clone https://github.com/mamathew98/code2gether.git
```

۳. یک ترمینال باز کنید و به دایرکتوری اصلی پروژه رفته و npm install را اجرا کنید و منتظر شوید تا وابستگی ها نصب شوند:

```
cd /code2gether
npm install
```

٤. اطمینان حاصل کنید که پورت ۳۰۰۰ دستگاه شما در دسترس است و کد زیر را اجرا کنید:

```
npm start
```

٥. ترمینال دیگری را باز کرده و به / Peer Server بروید و npm install را اجرا کنید و منتظر نصب وابستگی ها شوید

```
cd /PeerServer
npm install
```

۶. اطمینان حاصل کنید که پورت ۹۰۰۰ دستگاه شما در دسترس است و npm start را در مسیر / PeerServer اجرا کنید

```
npm start
```

۷. برای راه اندازی در سرور محلی خود ، به / public / javascripts / بروید و editor.js را باز کنید سپس قسمت مربوط به بارگذاری در سرور که با کامنت مشخص شده را، کامنت کرده و قسمت مربوط به بارگذاری محلی را از حالت کامنت خارج کنید

```
// FOR SERVER DEPLOYMENT
var socket = io("http://51.195.28.68:3000");
var socketCss = io("http://51.195.28.68:3000", {
  forceNew: true,
});
var socketJs = io("http://51.195.28.68:3000", {
  forceNew: true,
});

// FOR LOCAL DEPLOYMENT
// var socket = io("http://localhost:3000");
// var socketCss = io("http://localhost:3000", {
//   forceNew: true,
// });
// var socketJs = io("http://localhost:3000", {
//   forceNew: true,
// });
```

```
// FOR SERVER DEPLOYMENT
// var socket = io("http://51.195.28.68:3000");
// var socketCss = io("http://51.195.28.68:3000", {
//   forceNew: true,
// });
// var socketJs = io("http://51.195.28.68:3000", {
//   forceNew: true,
// });

// FOR LOCAL DEPLOYMENT
var socket = io("http://localhost:3000");
var socketCss = io("http://localhost:3000", {
  forceNew: true,
});
var socketJs = io("http://localhost:3000", {
  forceNew: true,
});
```

کد ۲۲ اجرای محلی/سرور

۸. یک مرورگر را باز کنید و به http: // localhost: ۳۰۰۰ بروید

اکنون می توانید یک پروژه را ثبت و ایجاد کنید و پیوند آن را با دیگران به اشتراک بگذارید تا بتوانید با هم در پروژه همکاری کنید

مجوز متن-باز (Open-Source-License)

پروانه ام‌آی‌تی: یک پروانه نرم‌افزاری آزاد، نوشته شده توسط موسسه فناوری ماساچوست (MIT) است. به عنوان یک پروانه آسان‌گیر، تنها محدودیت نه چندان مهم آن در استفاده مجدد است و سازگاری بسیار خوبی در قسمت‌های مختلف توسعه یک نرم‌افزار دارد. پروانه ام‌آی‌تی به نرم‌افزار اجازه می‌دهد که مجدداً در یک نرم‌افزار اختصاصی با مجوز غیر ام‌آی‌تی استفاده شود به شرط آنکه در تمامی نسخه‌های نرم‌افزار اختصاصی، سند شرایط استفاده پروانه ام‌آی‌تی وجود داشته باشد. این مجوز همچنین با مجوزهای کپی‌لفت از جمله پروانه عمومی همگانی گنو (جی‌پی‌ال) سازگار می‌باشد بدین معنی که ام‌آی‌تی اجازه ترکیب و توسعه با نرم‌افزارهای تولید شده با مجوز جی‌پی‌ال را می‌دهد و نه دیگر مجوزها.

از سال ۲۰۱۵، پیش از هر نوع پروانه نرم‌افزاری از خانواده جی‌پی‌ال و دیگر پروانه‌های آزاد و متن‌باز، پروانه ام‌آی‌تی محبوب‌ترین پروانه نرم‌افزاری بوده‌است.

قابل ذکرترین پروژه‌هایی که بر مبنای پروانه ام‌آی‌تی توسعه پیدا کرده‌اند می‌توان به Ruby on Rails، NodeJS، JQuery ، AngularJs اشاره کرد.

نتیجه گیری

امروزه با گسترش کسب و کارهای اینترنتی و افزایش نیاز کاربران به محصولات و اپلیکیشن‌های تحت وب، بار طراحی و توسعه محصولات تحت وب بر دوش توسعه دهندگان و شرکت‌ها و تیم‌های توسعه بطور چشم گیری افزایش داشته و با توجه به نیاز کاربران توسعه دهندگان مجبور هستند حجم کار قابل توجهی را در مدت کوتاهی تحویل و ارائه بدهند. از این رو بیش از پیش نیاز همکاری و مشارکت توسعه دهندگان با یکدیگر احساس میشود. همینطور در کنار این چنین مسائل، عده زیادی با توجه به بازار کار پیش آمده روی به یاد گیری و آموختن زبان‌های برنامه نویسی تحت وب نموده اند. در این راستا علی‌رقم وجود کانسپت‌هایی مانند git که امکان مشارکت و به اشتراک گذاشتن یک پروژه را در اختیار کاربران قرار میدهد، وجود پلتفرمی که بتواند در حداکثر حالت ممکن در زمان صرفه جویی کند و توسعه دهندگان و یا کارآموزهای زبان‌های تحت وب بصورت لحظه‌ای قابلیت مشارکت در پروژه بدون تداخل با یکدیگر را داشته باشند از مزیت بسیار بالای برخوردار خواهد بود. در عین حال در اکثریت پلتفرم‌های مشابه پیاده سازی شده در جهت این امر، تنها امکان ویرایش همزمان یک فایل در لحظه در اختیار کاربران قرار داده شده است. همچنین در برخی از این پلتفرم‌ها امکانات ارتباطی برای کاربران فراهم نشده است. فلذا پلتفرم code۲gether با توجه به قابلیت ویرایش همزمان چند فایل و ویرایشگر و همچنین قابلیت اجرای پروژه بصورت real-time و فراهم آوردن بستر ارتباطی چت و مکالمه تصویری، قدمی رو به جلو در جهت پیاده‌سازی ویرایشگرهای مشارکتی محسوب میشود .

پیشنهادات نسخه‌های آینده:

- اضافه کردن مدیریت فایل‌ها و گسترش پروژه به پروژه بزرگتر
- امکان اضافه کردن ویرایشگرها بصورت پویا و بصورت Tab و امکان مشاهده و اعمال تغییرات در هر Tab
- امکان اضافه کردن و مشاهده پیام یا Comment توسعه دهنده به رشته کد
- امکان برنامه نویسی و اجرای برنامه در زبان های دیگر

منابع

١. S.Kumawat & A.Khunteta ,Analysis of Operational Transformation Algorithms

٢. https://hackernoon.com/operational-transformation-the-real-time-collaborative-editing-algorithm-bf٨٧٥٦٦٨٣f٦٦

٣. http://www.codecommit.com/blog/java/understanding-and-applying-operational-transformation

٤. https://github.com/Operational-Transformation/ot.js/

٥. https://medium.com/@srijancse/operational-transformation-the-real-time-collaborative-editing-algorithm-bf٨٧٥٦٦٨٣f٦٦

٦. https://medium.com/@srijancse/how-real-time-collaborative-editing-work-operational-transformation-ac٤٩٠٢d٧٥٦٨٢

٧. https://socket.io/docs/

٨. https://codemirror.net/doc/manual.html

٩. https://peerjs.com/docs.html#api