پیام رسان

HATA Messenger

"Code Documentation"

فهرست

کلاینت(Client) ۴

توابع ٤

تابع ()main ٤

توابع ()login و ()createaccount ٤

توابع ()createchannel و ()joinchannel و ()logout ٤

توابع ()send_msg و ()leave ٤

تابع ()refresh ٥

تابع ()channel_member ٥

تابع ()RAP (Read and Print) ٥

تابع ()searchmember ٥

تابع ()searchmessages ٥

ظاهر برنامه خروجی ٦

تابع مرتبط با تم برنامه ٦

رنگ ها در صفحه کنسول ٦

منو پرینتر ها ٦

لوگو پرینتر ٦

تصاویری از محیط برنامه ٧

سرور (Server) ۱۰

توابع ۱۰

تابع ()main ۱۰

تابع ()RGCH ۱۰

تابع ()RGMM ۱۱

تابع ()randstring ۱۱

توابع جست و جو میان ساختار ها ۱۱

تابع ()readoperate ۱۱

تابع ()createaccount(char buff[]) ۱۲

تابع ()login(char buff[]) ۱۲

تابع ()nwchnnl(char buff[]) ۱۲

تابع (JoinCh(char buff[] ۱۲
تابع (SendMsg(char buff[] ۱۲
تابع (refresh(char buff[] ۱۳
تابع (chnlmmbr(char buff[] ۱۳
تابع (lvchnl(char buff[] ۱۳
تابع (logout(char buff[] ۱۳
تابع (searchMember(char buff[] ۱۳
تابع (MessageSearch(char buff[] ۱۳
تابع (searcher(char str[], char substr[] ۱۴
ظاهر برنامه خروجی ۱۴
جیسون پارسر (JSON Parser) ۱۵
ساز و کار و منطق عملیات ۱۵
توابع ۱۷
تابع (CreateNewObjectJSON(void ۱۷
تابع (CreateNewArrayJSON(void ۱۷
تابع (CreateNewStringJSON(const char const * string ۱۷
تابع (AddItemObjectJSON(JSON *object,const char *string,JSON *item ۱۷
تابع (AddItemArrayJSON(JSON *array, JSON *item ۱۷
تابع (OutputJSON(JSON *rootobject ۱۷
تابع (ParseJSON(const char * string ۱۸
تابع (GetObjectItemJSON(JSON *root,const char *string ۱۸
تابع (GetArrayItemJSON(JSON *array,int index ۱۸
تابع (GetArraySizeJSON(JSON *array ۱۸
تابع (DeleteJSON(JSON *input ۱۸

کلاینت(Client)

برنامه ایست که در اختیار کاربران قرار داده میشود تا با استفاده از آن بتوانند در بستر اینترنت پیام های خود را رد و بدل کنند. در ادامه توابع بنیادی و مهم این اپلیکیشن توضیح داده خواهند شد .

توابع

تابع main()

با شروع به کار این تابع ابتدا ملزومات ساخت و استفاده از سوکت ها در برنامه اجزا میشوند . پس از آن تم برنامه از فایل خوانده میشود ، لوگو برنامه نمایش داده میشود و سپس اکانت منو باز میشود . اگر کاربر با موفقیت وارد اکانت خود شود این تابع او را به منو اصلی برنامه میفرستد .

توابع login() و createaccount()

این توابع نام کاربری و رمز عبور را از کار برمیگیرد . رشته های مناسب برای ارسال به سرور را با استفاده از توابع کتابخانه رشته میسازد . پس از ساختن سوکت به سرور درخواست ساخته شده را ارسال میکند . جواب سرور را که به صورت رشته ای با پروتکل جیسون است را دریافت میکند آنرا پارس میکند و محتویات آنرا در رشته هایی میریزد ، اگر پیام موفقیت بود اگر حساب کاربری تازه ساخته شده باشد به اکانت منو وارد میشود تا کاربر وارد شود و اگر کاربر درخواست ورود داده بود توکن دریافت شده از سرور در متغیری جهانی(برای استفاده در دیگر توابع) ذخیره میشود و کاربر وارد منو اصلی میشود . اگر هم پیام خطا بود خطا را چاپ کرده و به اکانت منو باز میگردد.

توابع createchannel() و joinchannel() و logout()

این توابع نام کانال (در رابطه با دو تابع اول) را از کاربر میگیرند و مانند بالا رشته را ساخته و برای سرور ارسال میکنند و پاسخ سرور را پارس میکنند ، اگر موفقیت آمیز بود در دو مورد اول به چت‌منو میروند و در لوگ‌اوت وارد اکانت منو میشوند . در صورت خطا پیام خطا چاپ شده و کاربر به منو اصلی میرود .

توابع send_msg() و leave()

این توابع نیز رشته های مناسب را ساخته و برای سرور ارسال میکنند ، اگر موفقیت‌آمیز بود در سندمسج به چت‌منو بازمیگردد و در لیو به منو اصلی باز میگردد و در صورت خطا ، خطا نمایش داده میشود و به چت‌منو باز میگردد.

تابع refresh()

این تابع ابتدا دستور تازه سازی را برای سرور ارسال میکند . سرور پیامی حاوی پیام های دیده نشده از زمان ورود کاربر به کانال برای او ارسال میکند که به فرمت آرایه ای از آبجکت ها جیسون است . این تابع پس از پارس کردن پیام ، تعداد آجکت های موجود را پیدا میکند و در حلقه تک تک آبجکت ها را میخواند و در فایل chnl.txt ذخیره میکند که محتویات آن پس از خروج کاربر از کانال پاک و این فایل برای کانال های بعدی که کاربر وارد آن میشود مورد استفاده قرار میگیرد.

تابع channel_member()

این تابع درخواست خود را به سرور ارسال میکند و سرور پیامی حاوی آرایه ای از رشته های جیسون برای کاربر ارسال میکند . تابع به نوبت تمام آنها را از آرایهٔ پارس شده میخواند و چاپ میکند و منتظر فشردن کلیدی از طرف کاربر میماند و پس از فشرده شدن کلید به منو چت باز میگردد.

تابع RAP() (Read and Print)

این تابع فایل chnl.txt را میخواند و با استفاده از یک شمارنده شماره خط خوانده شده را نگه میدارد و اگر این شماره بین متغیر های f و l (که به صورت پیش فرض روی ده پیام آخر ست هستند و مقدار دهی های بعدی آنها در منو چت انجام میشود) بود آن خط را چاپ میکند .

تابع searchmember()

این تابع ابتدا نام اکانت مورد نظر کاربر را میگیرد و درخواست خود را به سرور ارسال میکند . پاسخ سرور پیامی مبتنی بر وجود یا عدم وجود اکانت در کانالی که کاربر در آن حاضر است دریافت میشود که چاپ میشود و سپس کاربر به منو چت بازمیگردد .

تابع searchmessages()

این تابع کلمه کلیدی را از کاربر دریافت میکند و درخواست خود را به سرور ارسال میکند . سرور پیامی حاوی آرایه ای از آبجکت ها (همانند refresh()) ارسال میکند . این پیام ها به ترتیب چاپ میشوند و برنامه منتظر فشردن کلیدی برای بازگشت میماند .

ظاهر برنامه خروجی

در اینجا توابعی که بیشتر مرتبط با ظاهر خروجی برنامه هستند برسی معرفی میشوند و تصاویری از محیط برنامه به نمایش گذاشته میشود.

تابع مرتبط با تم برنامه

برنامه دارای سه نوع تم طلایی ، هندوانه و اقیانوس است . در ابتدای برنامه آخرین تم تنظیم شده توسط کاربر از فایل خوانده میشود . تابع `loadtheme()` این کار را بر عهده دارد. تابع `PickTheme()` ، تم را از کاربر دریافت میکند و تنظیم میکند و آنرا در فایل نیز ذخیره میکند.

رنگ ها در صفحه کنسول

رنگ ها با استفاده از استاندارد *ANSI* در صفحه کنسول چاپ میشوند و در ابتدای برنامه با `#define` هایی کد رنگ ها تبدیل به حروف شده اند. تابع `CPATT(int texttype)` با توجه به تم برنامه و با توجه به نوع ورودی آن که ممکن است مقادیر `CHOOSED(1)` یا `NORMAL(2)` یا `MENU(3)` یا `OTHER(4)` باشد رنگ متن را از آن نقطه به بعد تغییر میدهد.

منو پرینتر ها

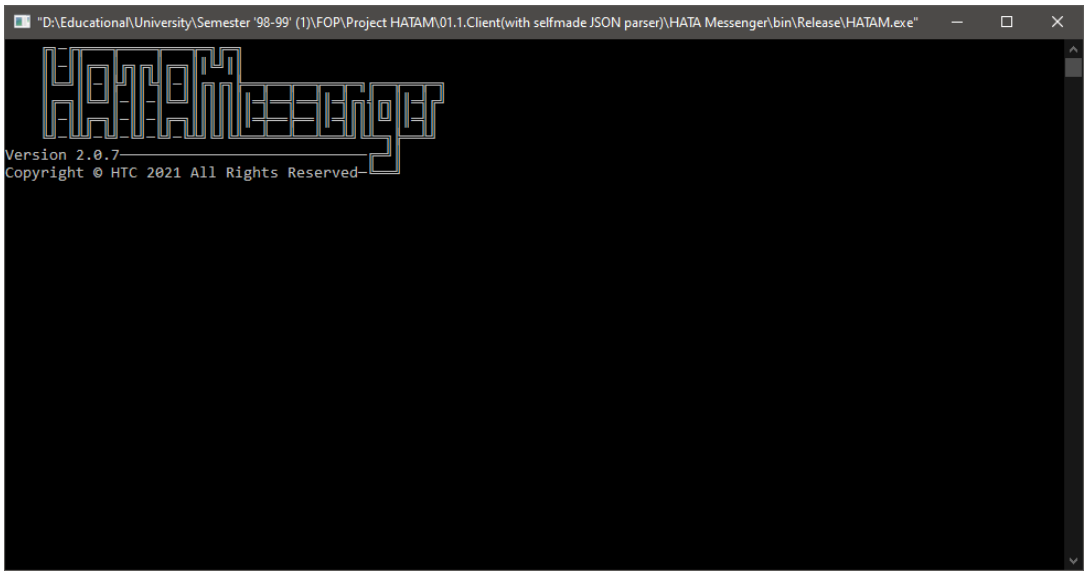
توابعی هستند که شمایل منو ها را با توجه به ورودی آنها که شماره گزینه ای که در حال حاضر فعال است چاپ میکند . گزینه ای که منتخب است به صورت روشن تر و با یک فلش در کنار آن چاپ خواهد شد .

جابجایی ها میان گزینه ها توسط کلید های جهت دار انجام میشود که طریقه استفاده از آنها با استفاده از دو `getch()` از کتابخانه `<conio.h>` در این لینک موجود است .

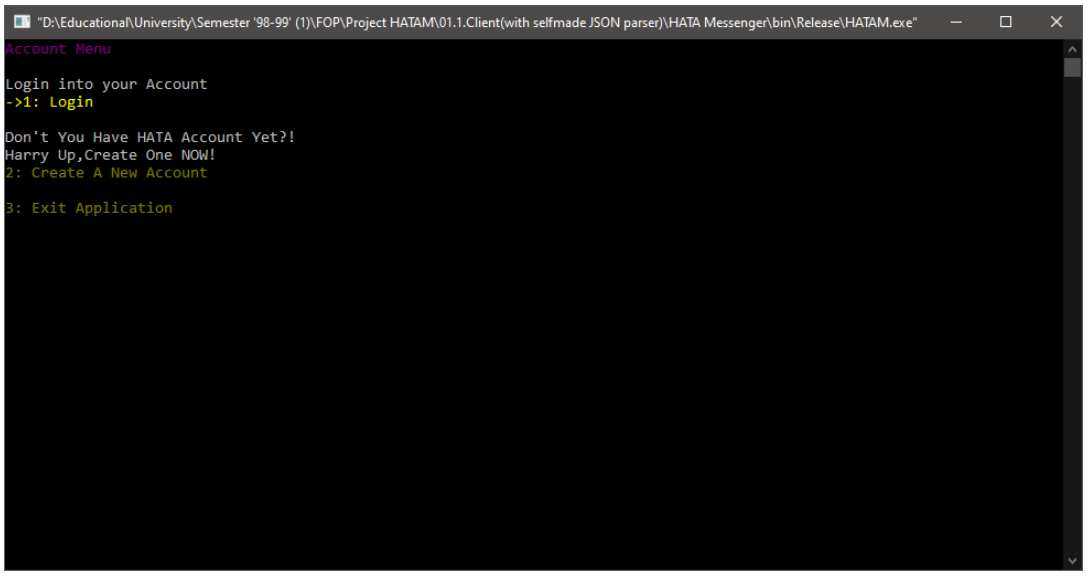
لوگو پرینتر

تابع `LogoPrint()` با استفاده از `UTF-8` لوگو برنامه که توسط این سایت طراحی شده است را چاپ میکند . این تابع با استفاده از `wprintf()` محتویات خود را چاپ میکند.

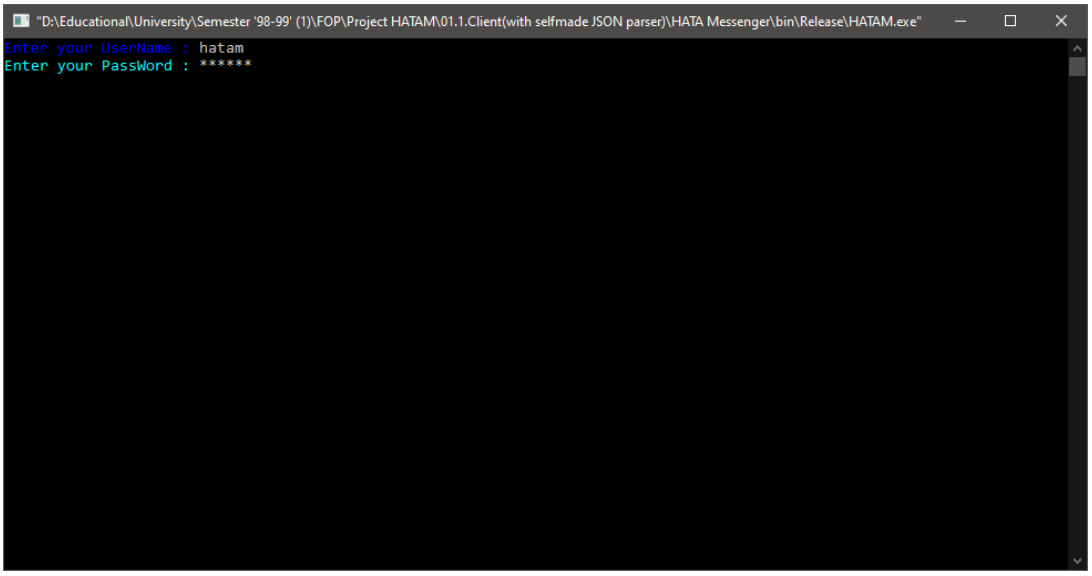
تصاویری از محیط برنامه



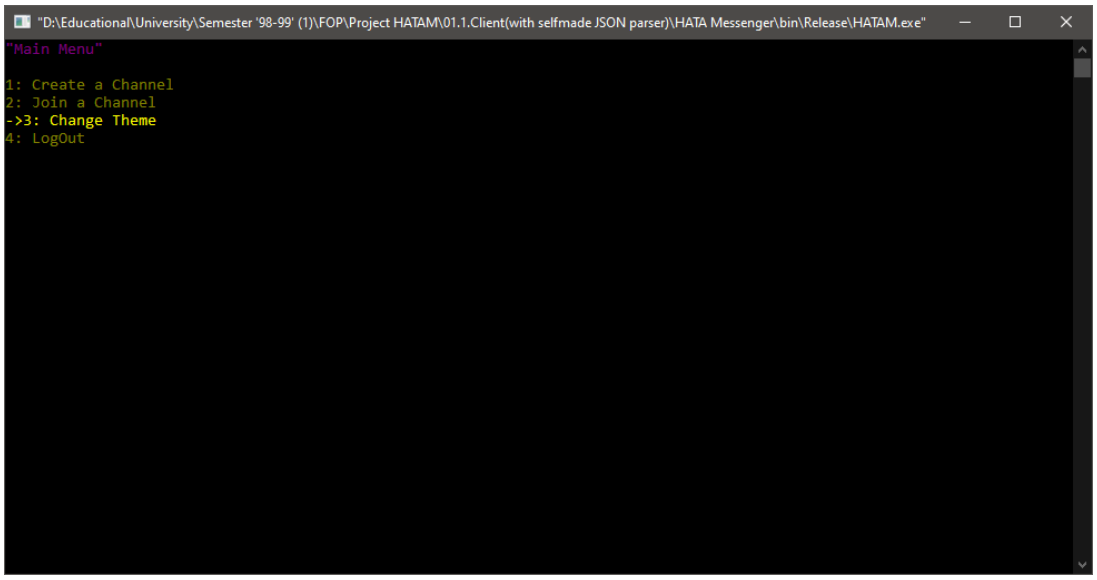
لوگو پرینتر



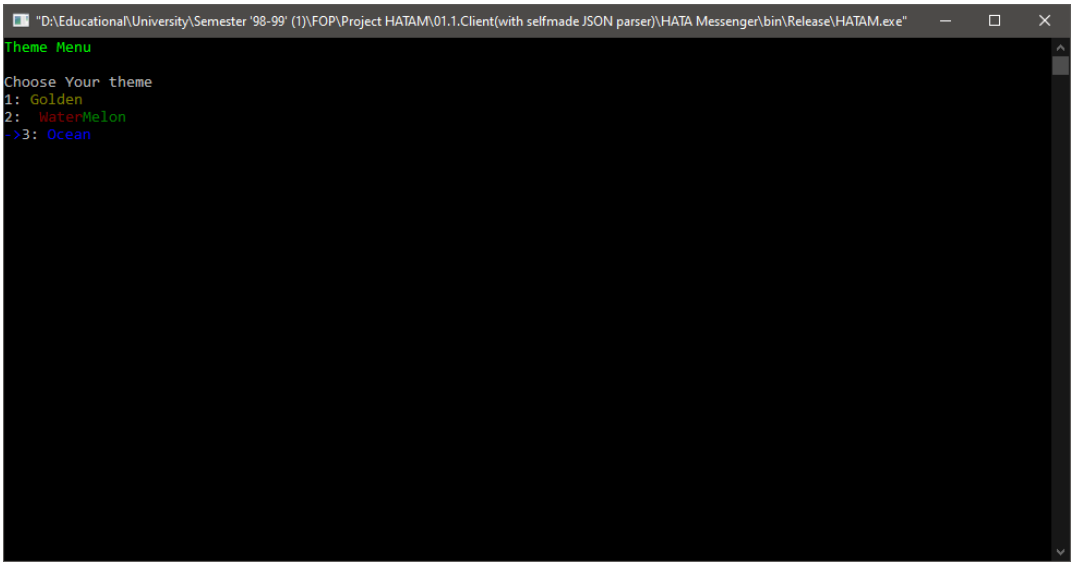
منو حساب کاربری



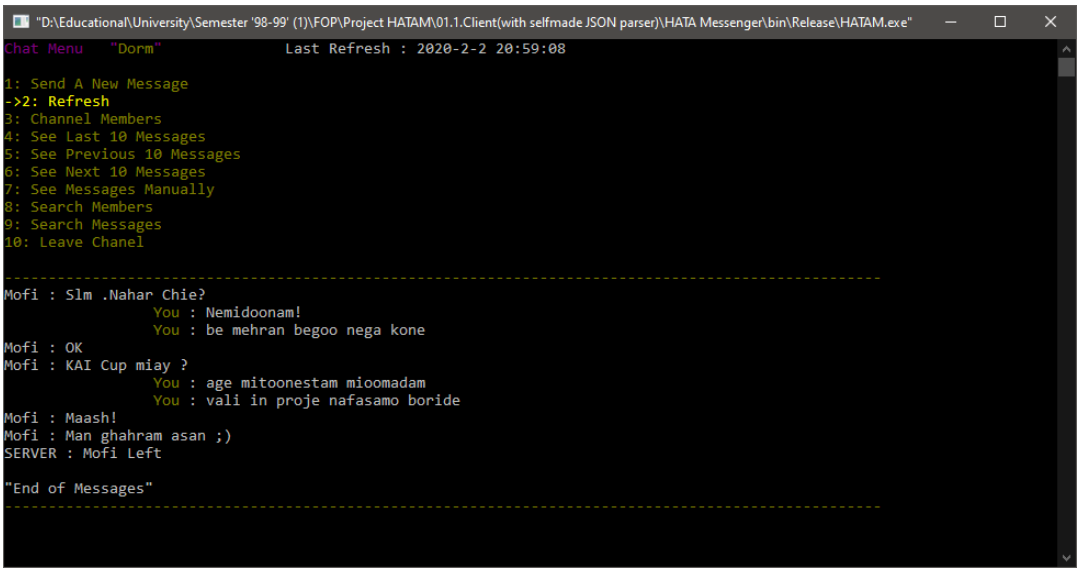
دریافت نام کاربری و رمزعبور



منوی اصلی (گزینه سوم در حال انتخاب است)



منوی انتخاب تم



منو گفت و گو

سرور (Server)

سرور پاسخ های مناسب را به درخواست های کاربران ارسال میکند . سرور از دو ساختار(structure) اساسی استفاده میکند که عبارتند از :

- Member مشتمل بر نام کاربر ، توکن ، نام کانالی که در آن حضور دارد و شماره آخرین پیامی که از سرور دریافت کرده است.
- Channel مشتمل بر نام کانال ، آرایه ای از آیدی کاربرانی که در آن حضور دارند و تعداد کاربرانی که در آن حضور دارند.

که سرور در ابتدای برنامه ۱۰۰۰ عدد از ساختار کاربر و ۵۰ عدد از ساختار کانال برای ذخیره اطلاعات در آنها میسازد.

سرور با انجام هر دستور ارسالی از سوی کاربر ، پیامی مبتنی بر عملیاتی که انجام داده و نتیجه آن پیامی در صفحه کنسول چاپ میکند.

در تمامی توابع ممکن است خطا هایی از جمله Authentication Failed رخ دهد که همانجا خطا ارسال ،چاپ و بازگشت به `readoperate()` انجام میشود.

توابع

تابع `main()`

در ابتدای آغاز به کار سرور ، سرور سوکتی برای خود میسازد ، آنرا bind میکند . تمامی اعضا و کانال هایی که از قبل موجود بوده اند را درون ساختار های خود ذخیره میکند تا نیازی به باز کردن هرباره فایل مربوط به هرکدام نباشد . در یک حلقه قرار میگیرد . در این حلقه زمانی که کلاینت دستور `connect` را اجرا میکند ، سرور آنرا `accept` میکند و برای ادامه کار تابع `readoperate()` صدا زده میشود.

تابع `RGCH()`

این تابع از ابتدای پوشه `channels` واقع در پوشه `Resources` شروع کرده و تمامی فایل های موجود در آنرا باز میکند ، اطلاعات مربوط به هر کانال را پس از پارس کردن رشته درون فایل به ساختار های مربوطه آن وارد میکند .

تابع RGMM()

این تابع از ابتدای پوشه memners واقع در پوشه Resources شروع کرده و تمامی فایل های موجود در آنرا باز میکند ، اطلاعات مربوط به هر عضو را پس از پارس کردن رشته درون فایل به ساختار های مربوطه آن وارد میکند .

تابع randstring()

این تابع زمانی که صدا زده میشود ، یک رشته تصادفی به طول ۳۰ تولید میکند ، آنرا در یک متغیر که حافظه ای به طول ۳۱ به آن تخصیص داده شده است میریزد و آدرس خانه اول آنرا برمیگرداند . لازم به ذکر است تابع ورودی نداشته و خروجی آن char * است.

توابع جست و جو میان ساختار ها

این توابع میان ساختار ها جست و جو میکنند و نتایج مطلوب را برمیگردانند :

- IsValidAuthToken(char *tkn) : توکن را دریافت میکند و اگر این توکن وجود داشت مقدار ۱ و در غیر اینصورت مقدار صفر را برمیگرداند.
- FindMemmberIDbyToken(char *tkn) : توکن را دریافت میکند . اگر چنین توکنی وجود داشت ایندکس آن عضو در آرایه اعضا را برمیگرداند در غیر این صورت مقدار ۱- برمیگرداند.
- FindMemberIDByUsername(char username[]) : نام کاربر را دریافت میکند و شماره ایندکس او را برمیگرداند .در غیر این صورت مقدار منفی ۱ برمیگرداند.
- FindChannelByName(char chnm[]) : نام کانال را میگیرد و ایندکس کانال را برمیگرداند . اگر موجود نبود ۱- برمیگردد.

تابع readoperate()

این تابع ابتدا با استفاده از تابع setsockopt محدودیت زمان ارسال برای کلاینت در نظر میگیرد . اگر این زمان تمام شود پیام اتمام زمان ارسال و سرور به کار خود ادامه میدهد . اگر کلاینت در زمان مقرر درخواست خود را ارسال کند این تابع نوع درخواست را تشخیص میدهد و تابع مربوط به آنرا صدا میزند و اگر درخواست شناخته شده نبود ، پیغام خطا ارسال میکند و به کار خود ادامه میدهد .

تابع createaccount(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، نام کاربری و پسورد را از آن استخراج میکند . با استفاده از توابع جست و جو در ساختار ها اگر کاربری با این نام وجود نداشت ، کاربر جدید را ثبت و فایل مرتبط با آنرا میسازد و ذخیره میکند در غیر این صورت پیغام خطا را برای کلاینت ارسال میکند.

تابع login(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، نام کاربری و پسورد را از آن استخراج میکند . با استفاده از توابع جست و جو در ساختار ها اگر کاربری با این نام وجود داشت ، پسورد را چک میکند اگر درست بود برای او یک توکن میسازد و پیام موفقیت را همراه با توکن ارسال میکند . در غیر این صورت خطا هایی که رخ داده است شناسایی پیام مربوط ارسال میشود

تابع nwchnnl(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن و نام موجود در آن را استخراج میکند . ابتدا چک میکند کانالی با این نام وجود نداشته باشد . اگر وجود نداشت با استفاده از جیسون ، آبجکتی شامل آرایهٔ پیام ها (که شامل اولین پیام سرور) و رشتهٔ نام کانال است. سپس در ساختار این کانال آیدی کاربر را اضافه میکند و در ساختار کاربر نیز نام کانال را میریزد.

تابع JoinCh(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن و نام موجود در آن را استخراج میکند . ابتدا چک میکند کانالی با این نام وجود نداشته باشد . اگر وجود داشت با استفاده از جیسون ، ورود او را به پیام ها اضافه میکند . سپس در ساختار این کانال آیدی کاربر را اضافه میکند و در ساختار کاربر نیز نام کانال را میریزد.

تابع SendMsg(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن و پیام را از آن استخراج میکند . با استفاده از توابع جست و جو در ساختار ها کانال و نام فرستنده را با استفاده از توکن بدست می‌آورد . پیام را به صورت یک آبجکت جیسون در می‌آورد . سپس فایل کانال مورد نظر را میخواند . جیسون آنرا خارج و پارس میکند . آبجکت ساخته شده را به آرایهٔ پیام ها اضافه میکند . جیسون را به صورت رشته درمی‌آورد و آنرا در فایل کانال ذخیره میکند.

تابع refresh(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن را توکن را استخراج میکند . کانال و کاربر را پیدا میکند . فایل کانال را باز ، و محتویاتش را پارس میکند. یک آبجکت جدید برای ارسال میسازد . پیام هایی که بعد از آخرین تازه سازی کاربر است را وارد آن آبجکت میکند و رشته خروجی جیسون ارسال را ارسال میکند. اگر پیامی توسط کسی که درخواست تازه‌سازی را داده است وجود داشته باشد با دو `\t` به جلو رانده میشود و رنگ آن نیز به رنگ زرد درمی‌آید.

تابع chnlmmbr(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد توکن را استخراج میکند . کانال را شناسایی میکند . با استفاده از آرایهٔ ایندکس های ثبت شده ، نام های کاربران حاضر را در آرایه جیسون قرار میدهد.

تابع lvchnl(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن را استخراج میکند . کانال و کاربر را شناسایی میکند . آیدی کاربر را از ساختار کانال خارج میکند . اسم کانال را از ساختار کلاینت پاک میکند . در صورت خطا ها ، خطا ها را ارسال میکند در غیر این صورت پیام موفقیت ارسال میشود.

تابع logout(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن را استخراج میکند . کاربر را شناسایی میکند و توکنش را پاک میکند .

تابع searchMember(char buff[])

این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن و نام عضو مورد نظر را استخراج میکند . کانال را پیدا میکند . با استفاده از ایندکس اعضا و تطبیق آن با نام استخراج شده نتیجه را برای کلاینت ارسال میکند .

تابع MessageSearch(char buff[])

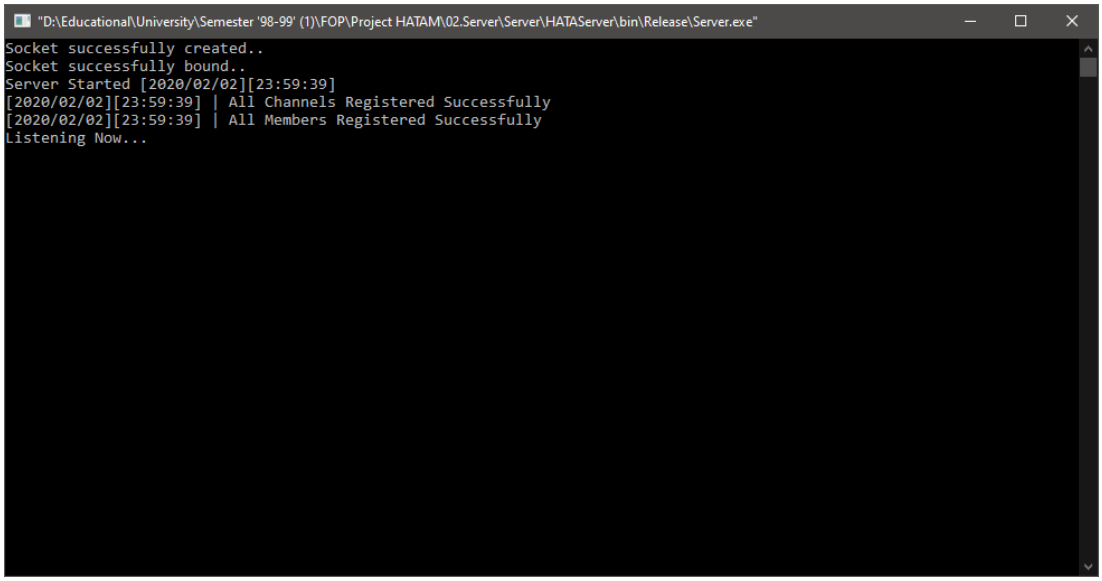
این تابع پیام فرستاده شده توسط کلاینت را به عنوان ورودی میگیرد ، توکن و کلید جست و جو را استخراج میکند . کانال را پیدا میکند ، فایل را میخواند . تمام پیام ها را برسی میکند(با استفاده از تابع `searcher`) و پیام هایی که دارای کلید باشند را در آرایه جیسون نگه میدارد و در انتها ارسال را انجام میدهد.

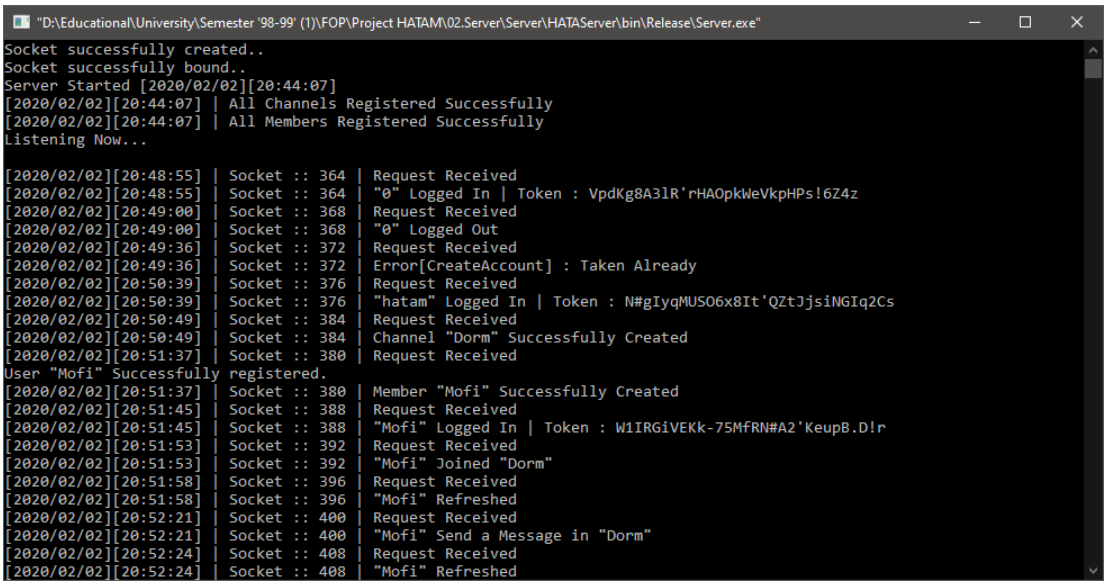
تابع `searcher(char str[], char substr[])`

این تابع دو ورودی رشته و زیررشته دارد . در رشته دنبال زیررشته میگردد و هرگاه آنر پیدا کرد مقدار ۱ و در غیر این صورت مقدار صفر را برمیگرداند .

ظاهر برنامه خروجی

برنامه هر سوکتی که از طریق آن ریکوئستی دریافت کرده را از نمایش میدهد . سپس عملیات ها را انجام میدهد و نتیجه ارسال ها را نیز روی صفحه کنسول نمایش میدهد . سرور تابعی اختصاصی برای نمایش خروجی‌ها ندارد.





جیسون پارسر (JSON Parser)

ساز و کار و منطق عملیات

در این قسمت ساختاری درختی برای هر جیسون در نظر گرفته شده است که به صورت زیر است (این ساختار تقلیدی از ساختار Dave Gamble سازنده cJSON است اما توابع تقلیدی نیستند) :

```
struct JSON
{
    struct JSON *next;
    struct JSON *prev;
    struct JSON *child;
    int type;
    /*
    String    1
    Array     2
    Object    3
    */
    /*The item's string, if type == JSON_String*/
    char *valuestring;
    /* The item's name string, if this item is the child of, or is in the list of subitems of an object. */
    char *string;
}
```

که در آن هر JSON یک فرزند از جنس JSON دارد که این فرزند بودن نشان دهنده نوعی زیرگروه بودن است.

قسمت next و prev در آرایه ها و آبجکت ها معنا پیدا میکند که هرکدام در دل خود ممکن است چندین JSON نگه دارد (children).

قسمت valuestring به string ها از جنس JSON مربوط میشوند که تنها میتوانند فرزند یکی از آرایه ها و آبجکت ها باشند .

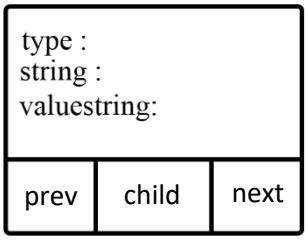
قسمت string میتواند مربوط به هرکدام از انواع JSON باشد.

مطلب مهمی که وجود دارد این است که در اینجا هر دو آبجکت و آرایه را به چشم آرایه نگاه میکنیم ، تفاوت آنها تنها در هنگام چاپ شدن است.

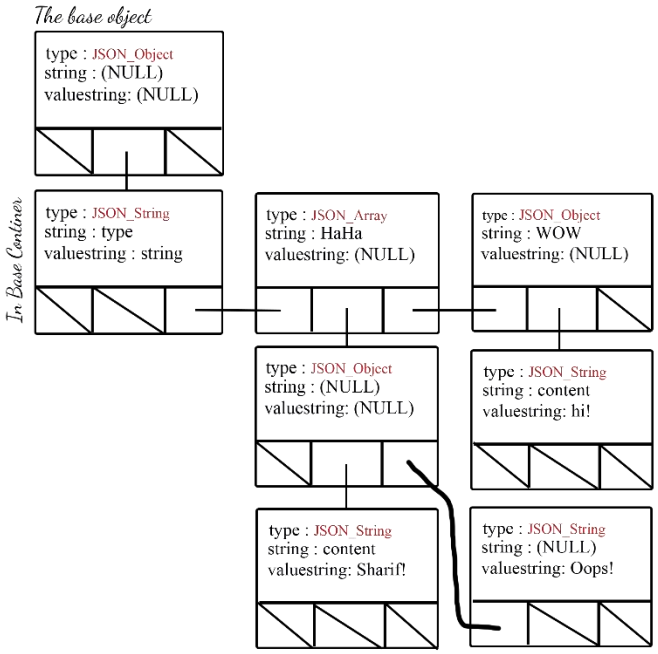
برای واضح شدن به مثال زیر توجه کنید :

```
{"type":"string","HaHa":[{"content":"Sharif!"},"Oops!"]
,"WOW":{"content":"hi!"}}
```

حال اگر هر ساختار JSON را به صورت زیر در نظر بگیریم(خط کج نشاندهنده NULL است) :



رشته بالا به شکل زیر در خواهد آمد :



توابع

تابع CreateNewObjectJSON(void)

این تابع یک JSON* میسازد و فضا برای آن در نظر میگیرد . تایپ آنرا مشخص میکند و آدرس آنرا برمیگرداند.

تابع CreateNewArrayJSON(void)

این تابع یک JSON* میسازد و فضا برای آن در نظر میگیرد . تایپ آنرا مشخص میکند و آدرس آنرا برمیگرداند.

تابع CreateNewStringJSON(const char const * string)

این تابع یک رشته میگیرد به و با استفاده از StrDuplicate فضا برای آن در نظر میگیرد. حال یک JSON* میسازد و رشته ساخته شده به valustring آن نسبت داده میشود . سپس تایپ آنرا مشخص میکند.

تابع AddItemObjectJSON(JSON *object,const char *string,JSON *item)

ابتدا string دریافتی را در string قسمت item میریزد و سپس با object مثل یک آرایه برخورد میکند و تابع AddItemArrayJSON صدا زده میشود و ورودی های object و item به آن داده میشود .

تابع AddItemArrayJSON(JSON *array, JSON *item)

اگر item خالی باشد بازمیگردد در غیر اینصورت روی فرزندان آرایه حرکت میکند تا به آخرین قسمت برسد. حال قسمت بعد آخرین فرزند را به قبل item متصل میکند و قسمت قبل item را به قسمت بعد آخرین فرزند آرایه متصل میکند.

تابع OutputJSON(JSON *rootobject)

این تابع از اولین فرزند rootobject شروع میکند و هرکدام از فرزندان را با توجه به نوع آنها به رشته نهایی اضافه میکند . هر کدام از انواع با استفاده از تابع خاصی که همه شبیه هم انجام میشوند. در هرکدام از این توابع اگر بخشی خود صاحب فرزند بود ابتدا فرزندان به رشته نهایی اضافه میشوند و سپس به ادامه فرزندان رده بالاتر پرداخته میشود .

تابع ParseJSON(const char * string)

ابتدا string اولیه را پیدا میکند . سپس قسمت بعدی را تشخیص میدهد ({ , [, ") با توجه به قسمت بعدی توابعی مشابه با خود را صدا میزند (برای مثال برای پارس کردن آرایه آنقدر جلو میرور تا به [برسد و برای آبجکت به { برسد). زمانی که یک بخش کامل بدست آمد با استفاده از توابع Add به JSON* اصلی افزوده میشود . زمانی که بیرونی ترین تابع به '\0' انتهای رشته اولیه رسید آدرس جیسون اصلی بازگردانده میشود . نکته اساسی در دسته توابعی که برای پارس کردن استفاده شده است این است که محتویات را تا زمان بسته شدن چیزی که به خاطر آن صدا زده شده اند (" , [, {) ادامه میدهد ، زمانی که بسته شد جیسون مناسب را میسازد و به ریشه جیسونی که آنرا صدا زده می‌افزاید.

تابع GetObjectItemJSON(JSON *root,const char *string)

این تابع بین تمام فرزندان root جست و جو میکند و آدرس فرزندی که string مشابه با رشته دریافتی داشته باشد را برمیگرداند و اگر به آخرین فرزند رسید و پیدا نکرد NULL برمیگرداند.

تابع GetArrayItemJSON(JSON *array,int index)

این تابع یک شمارنده نگه میدارد و بین فرزندان آرایه حرکت میکند و با گذر از هر فرزند یکی به شمارنده می‌افزاید تا زمانی که شمارنده با index برابر شود . زمانی که برابر شد آدرس آن فرزند را برمیگرداند و اگر به انتها رسید و به آن ایندکس نرسید NULL برمیگرداند .

تابع GetArraySizeJSON(JSON *array)

این تابع یک شمارنده نگه میدارد و بین فرزندان آرایه حرکت میکند و با گذر از هر فرزند یکی به شمارنده می‌افزاید تا زمانی که به آخرین فرزند برسد . آنگاه شمارنده را برمیگرداند.

تابع DeleteJSON(JSON *input)

از بیرونی ترین لایه شروع به پیش روی میکند . اگر فرزند داشته باشد ابتدا همین تابع به صورت بازگشتی برای فرزندش صدا زده میشود و سپس اگر valuestring و string داشته باشد این دو free میشوند سپس خود آن free میشود و بعد از آن تابع به سراغ گره بعدی میرود تا به انتهای گره ها برسد.