

پروژه دوم درس مبانی رایانش توزیع شده

بهار ۱۴۰۳

پویا صادقی	مجید فریدفر	اولدوز نیساری	زهرا حجتی
۸۱۰۱۹۹۴۴۷	۸۱۰۱۹۹۵۶۹	۸۱۰۱۹۹۵۰۵	۸۱۰۱۹۹۴۰۳

	Concurrent ticket reservation system
1	راه اندازی پروژه جدید
1	پیاده سازی سرویس رزرو بلیت
2	پیاده سازی concurrency control
2	پیاده سازی Client Interface
2	پیاده سازی Fairness و Resource management
2	پیاده سازی Logging و Error handling
3	caching
17	تقسیم وظایف پروژه

دیگراتوری و وظایف اصلی کد

این کد یک سیستم مدیریت رزرو بلیت به زبان Go پیاده سازی می کند.

گام های پروژه :

1- راه اندازی پروژه جدید :

از آنجایی که یک سیستم کلاینت سرور را به کمک زبان برنامه نویسی go را می خواستیم بسازیم سعی کردیم ساختار منظم و سازمان یافته ای برای آن داشته باشیم کد های مربوط به بخش های سرور و کلاینت را جدا کردیم ، به علاوه فولدری به نام manager برای سازمان دهی ارتباط بین این دو ، فولدری به نام model برای ذخیره ساختمان داده های مشترک ، فولدری به نام rest برای ایجاد اتصال ها و فولدری به نام utils برای تعریف توابع کمکی.

در گام اول در فولدر model درفایل event.go , ticket.go استراکت های Event و Ticket را به همراه کانستراکتور هایشان تعریف می کنیم.

درباره ساختار داده ها :

- **event** یک رویداد که شامل شناسه، نام، تاریخ، تعداد کل بلیت ها و تعداد بلیت های موجود است.

- **ticket** یک بلیت که شامل شناسه و شناسه رویداد مربوطه است.

2- پیاده سازی سرویس رزرو بلیت :

نکته ای که در پیاده سازی استراکت ها وجود دارد این است که باید حرف اول بزرگ داشته باشند تا به عنوان یک آبجکت بین فایل های اکسترنال شناسایی شوند.

برای سرور استراکتی به نام ticket Service تعریف می کنیم و در فولدر manager قرار می دهیم . این استراکت کمک می کند سرور بتواند درخواست های کلاینت را در سمت خودش مدیریت کند .

- **TicketService** یک سرویس برای مدیریت رویدادها و بلیت ها که از sync.Map برای ذخیره رویدادها استفاده می کند و دو Mutex برای کنترل دسترسی همزمان استفاده شده است.

3 متد بر روی این استراکت تعریف می شوند :

CreateEvent

- این تابع یک رویداد جدید ایجاد می کند.

- اگر هیچ اروری رخ ندهد، مقدار nil برای ارور برگردانده می شود.

- در صورتی که یک ارور رخ دهد، آن ارور به عنوان مقدار برگردانده می شود.

ListEvents

- این تابع لیست تمام رویدادهای موجود را برمی گرداند.

- این تابع هیچ اروری تولید نمی‌کند و نیازی به هندل کردن ارورها نیست.

BookTickets

- این تابع بلیت‌ها را برای یک رویداد خاص رزرو می‌کند.

- اگر بلیت‌ها با موفقیت رزرو شوند، مقدار nil برای ارور برگردانده می‌شود.

- در صورتی که یک ارور رخ دهد، آن ارور به عنوان مقدار برگردانده می‌شود.

سرور می‌تواند نمونه‌ای از این استراکت بگیرد و در زمان نیاز به هندلر مربوطه در زمان مدیریت اتصال و ارتباط بین کلاینت و سرور بدهد.

ReadData

- این تابع ایونت‌ها را از دیتابیس می‌خواند و به فرمت مطلوب برنامه تبدیل می‌کند (تبدیل جیسون به لیستی از ایونت‌ها)

- اگر فایل دیتابیس از پیش ساخته نشده باشد، آن را درست می‌کند.

- در صورتی که یک ارور رخ دهد، متوقف می‌شود.

این تابع در ابتدای ست‌آپ شدن سرور صدا زده می‌شود.

WriteData

- این تابع ایونت‌ها را در دیتابیس ذخیره می‌کند و لیست ایونت‌ها را به جیسون تبدیل می‌کند.

- اگر فایل دیتابیس از پیش ساخته نشده باشد، آن را درست می‌کند.

- در صورتی که یک ارور رخ دهد، متوقف می‌شود.

این تابع زمانی که در ترمینال مربوط به سرور، `Ctrl + C` زده شود، صدا زده می‌شود.

3- پیاده‌سازی Concurrency control :

یکی از نقاط بحرانی در تابع رزرو است، دسترسی همزمان چند کلاینت به این ناحیه می‌تواند موجب data inconsistency شود. برای حل این مشکل روی تابع BookTickets از mutex استفاده می‌کنیم.

4- پیاده‌سازی Client interface :

برای چنین سیستمی وجود یک interface برای کلاینت ضروری است. از طریق این interface کاربران درخواست‌های خود را به سرور ارسال کنند. برای مدیریت درخواست‌ها متناظر توابع نمایش event ها و رزرو بلیت.

اینجا هم ممکن است با مشکل درخواست‌های همزمان مواجه باشیم که برای حل این مشکل از go routines استفاده می‌کنیم.

5- پیاده‌سازی Fairness , Resource Management :

برای کنترل resource ها از lock\unlock و mutex استفاده شده است که دسترسی ها را چک می کند. توضیحات بیشتر این بخش در ادامه در توضیحات کدها قرار داده شده اند.

6- پیاده سازی logging , error handling :

این بخش از کد شامل توابعی است که برای هندل کردن و چاپ خطاها و اطلاعات مربوط به رزرو بلیت و ارتباط با کلاینت ها استفاده می شود. بیایید هر تابع را بصورت خط به خط بررسی کنیم:

logReservation

- این تابع برای چاپ اطلاعات مربوط به رزرو بلیت استفاده می شود.
- اگر یک خطا رخ داده باشد (مقدار err غیر nil باشد) ، پیام خطا به همراه اطلاعات مربوط به رزرو بلیت چاپ می شود.
- در غیر این صورت، یعنی اگر خطا رخ نداده باشد، اطلاعات مربوط به رزرو بلیت چاپ می شود.

handleReservationError

- این تابع برای چاپ پیام خطاهای مربوط به رزرو بلیت استفاده می شود.
- اگر یک خطا رخ داده باشد (مقدار err غیر nil باشد) ، پیام خطا به همراه شماره کلاینت (clientID) چاپ می شود.
- در غیر این صورت، هیچ عملی انجام نمی شود.

handleClientInteraction

- این تابع برای چاپ وضعیت مربوط به ارتباط با کلاینت ها استفاده می شود.
- اگر یک خطا رخ داده باشد (مقدار err غیر nil باشد) ، تابع handleReservationError برای چاپ پیام خطا فراخوانی می شود.
- در غیر این صورت، اطلاعات مربوط به رزرو بلیت و ارتباط با کلاینت چاپ می شود.

7-caching :

دسترسی به cache نیز با به کار گیری sync map انجام شده که دسترسی همزمان (concurrent) را مدیریت می کند. این بخش نیز در ادامه بیشتر توضیح داده شده است.

بخش های پیاده سازی پروژه :

(1 server interface :

اینترفیس اصلی سرور از طریق Ticket Service انجام می شود و از طریق handler یک کانال بین این استراکت و استراکت سرور و کلاینت برقرار می شود.

در پوشه ی app در قسمت server، در اصل یک server برای http راه اندازی می شود.

1. **import**: کتابخانه های مورد نیاز برای اجرای برنامه وارد می شوند. این شامل `manager` و `server` از پکیج `TicketReservation/src`، `log` برای ثبت خطاها و `sync` برای ایجاد نقشه های همزمان است.

2. **main**: تابع اصلی برنامه که هنگام اجرای برنامه فراخوانی می شود.

3. در تابع **main**:

- **port** و **filePath** مقادیر پیش فرض برای پورت سرور و مسیر فایل پایگاه داده تعریف می شوند.
- **ticketService**: یک نمونه از `TicketService` ایجاد می شود که دو نقشه همزمان برای رویدادها و بلیط ها دارد.
- **ReadData** و **WriteData** داده ها از فایل JSON خوانده شده و در پایان برنامه به فایل نوشته می شوند.
- **server**: یک نمونه از `Server` ایجاد می شود که `TicketService` را به عنوان یکی از ویژگی های خود دارد.
- **SetupHttpApiServer**: سرور HTTP راه اندازی می شود و در صورت بروز خطا، پیام خطا ثبت می شود.

(2 client interface)

اینترفیس اصلی کلاینت از در پوشه `rest` تعریف شده است که دارای 4 متد می باشد، این بخش بیشتر مربوط به کارکرد اصلی ui خود `client` می باشد:

- item.go, menu.go

در تو فایل `menu.go` و `item.go` به ترتیب منو و استراکت اطلاعات کلاینت ها می باشد. پکیج این دو ui نامگذاری شده است.

struct Item: این ساختار یک آیتم منو را تعریف می کند. هر آیتم شامل یک شناسه (ID)، یک نام (Name) و یک توضیح (Description) است.

تابع InitBaseMenu: این تابع یک منو اولیه را برای مدل مشخص می کند. این منو شامل چند آیتم است که هر کدام یک عملیات خاص را نشان می دهند، مانند "نمایش رویدادها"، "رزرو بلیت"، "ایجاد رویداد جدید"، "نمایش راهنما" و "خروج از برنامه". این آیتم ها سپس به فیلد `Items` مدل اختصاص داده می شوند.

به طور کلی، این کد یک رابط کاربری ساده برای یک برنامه رزرو بلیت را تعریف می کند. کاربر می تواند از طریق این منو عملیات مختلف را انجام دهد.

- model.go

در ادامه کد model.go پیاده سازی شده است که بخشی از client interface از پکیج ui است. یک برنامه تعاملی در ترمینال را با استفاده از کتابخانه Bubble Tea پیاده سازی می کند. در اینجا توضیحات کلی در مورد هر بخش ارائه می شود:

1. **struct Model**: این ساختار داده ای شامل یک لیست از آیتم ها (Items) و یک شماره انتخاب شده (Selected) است.

2. **تابع Init**: این تابع مدل را مقداردهی اولیه می کند. در اینجا، هیچ کار خاصی انجام نمی شود و فقط nil برگردانده می شود.

3. **تابع Update**: این تابع پیام های ورودی را بررسی می کند و بر اساس آن ها مدل را به روزرسانی می کند. برای مثال، اگر کاربر کلید "بالا" یا "پایین" را فشار دهد، شماره آیتم انتخاب شده تغییر می کند. اگر کاربر کلید "ورود" را فشار دهد یا برنامه را ببندد، برنامه خاتمه می یابد.

4. **تابع View**: این تابع یک رشته را برای نمایش منو ایجاد می کند. اگر یک آیتم انتخاب شده باشد، آن را با یک نشانگر (►) سبز رنگ نشان می دهد. همچنین، توضیحات هر آیتم را با رنگ مشکی روشن نشان می دهد. به طور کلی، این کد یک رابط کاربری ساده برای یک برنامه تعاملی در ترمینال را پیاده سازی می کند. کاربر می تواند با استفاده از کلیدهای "بالا" و "پایین" بین آیتم های منو حرکت کند و با فشار دادن کلید "ورود" یک آیتم را انتخاب کند.

- eventModel.go

این کد یک برنامه کاربردی رابط کاربری (UI) برای یک سیستم رزرو بلیت است. این کد با استفاده از کتابخانه bubbletea برای مدیریت UI و lipgloss برای زیباسازی خروجی کار می کند. برنامه از مدل EventModel استفاده می کند که یک لیست از رویدادها و یک شاخص انتخاب شده را نگه می دارد.

- تابع **NewEventModel**: یک نمونه جدید از EventModel با لیست رویدادهای داده شده ایجاد می کند.

- تابع **Init**: برای مقداردهی اولیه مدل استفاده می شود. در اینجا هیچ کار خاصی انجام نمی شود.

- تابع **Update**: پیام های ورودی را می گیرد و بر اساس آن ها وضعیت مدل را به روز می کند. برای مثال، اگر کاربر کلید up یا down را فشار دهد، شاخص انتخاب شده تغییر می کند.

- تابع **View**: یک رشته را برمی گرداند که نمایش UI را توصیف می کند. این رشته بر اساس وضعیت فعلی مدل ساخته می شود.

در کل، کاربر می تواند با استفاده از کلیدهای up و down بین رویدادها حرکت کند و با فشار دادن enter یک رویداد را انتخاب کند.

- app.go

این بخش نیز از پکیج ui است و یک کلاینت API را نگه می دارد.

- تابع **NewApp**: یک نمونه جدید از App با کلاینت API داده شده ایجاد می کند.

- تابع **showEvents**: رویدادهای موجود را از سرور دریافت و نمایش می‌دهد.
- تابع **bookTickets**: به کاربر اجازه می‌دهد تا بلیت برای یک رویداد را رزرو کند. کاربر ابتدا یک رویداد را انتخاب می‌کند و سپس تعداد بلیت‌های مورد نیاز را وارد می‌کند.
- تابع **createNewEvent**: به کاربر اجازه می‌دهد تا یک رویداد جدید ایجاد کند. کاربر نام، تاریخ و تعداد کل بلیت‌ها را مشخص می‌کند.
- تابع **help**: یک منوی راهنما را نمایش می‌دهد که توضیح می‌دهد چگونه از برنامه استفاده کنید.
- تابع **step**: یک دور از حلقه اصلی برنامه را اجرا می‌کند. این تابع ابتدا یک منو را نمایش می‌دهد و سپس بر اساس انتخاب کاربر عملیات مربوطه را اجرا می‌کند.
- تابع **RunUI**: این تابع حلقه اصلی برنامه را اجرا می‌کند. در هر دور حلقه، صفحه پاک می‌شود و تابع **step** فراخوانی می‌شود. اگر تابع **step**، معادل **false** را برگرداند، برنامه خاتمه می‌یابد. در غیر این صورت، کاربر باید Enter را فشار دهد تا به منوی بعدی برود.
- تابع **clearTerminal**: این تابع صفحه ترمینال را پاک می‌کند. این کار با استفاده از دستورات مختلف برای سیستم‌های عامل مختلف انجام می‌شود.
- تابع **RunTest**: این تابع در حال حاضر پیاده‌سازی نشده است و یک خطای "Implement App.RunTest!" را ایجاد می‌کند. این تابع به نظر می‌رسد برای اجرای یک تست فشار (pressure test) در نظر گرفته شده است، که تعداد مشتریان و درجه بارگذاری به عنوان پارامترها دریافت می‌کند.

3) فایل های جانبی فولدر rest :

- endpoints.go:

- این کد تعریف می‌کند که چگونه مسیرهای مختلف در سرور HTTP ما مورد استفاده قرار می‌گیرند. هر یک از مسیرها با یک تابع مربوطه متصل است که در درخواست های HTTP مربوطه کار می‌کند.
1. **apiSetReservation**: این مسیر به تابع **setReservationHandler** متصل است که برای رزرو بلیت استفاده می‌شود.
 2. **apiGetEvents**: این مسیر به تابع **getEventsHandler** متصل است که برای دریافت رویدادهای موجود استفاده می‌شود.
 3. **apiCreateEvent**: این مسیر به تابع **createEventHandler** متصل است که برای ایجاد یک رویداد جدید استفاده می‌شود.
 4. **serverAddr**: این مقدار آدرسی را تعریف می‌کند که سرور HTTP ما بر روی آن گوش می‌دهد. در این مورد، آدرس "127.0.0.1:8080" است که به معنای localhost یا میزبان محلی در پورت 8080 است.

- handler.go :

این بخش مربوط به تابع **handler** است که اینترفیس بین **client** و **server** را برقرار می‌کند، این کار توسط سه تابع HTTP را پیاده سازی می‌شود:

createEventHandler و setReservationHandler، این توابع به ترتیب برای رزرو بلیت، دریافت رویدادهای موجود، و ایجاد یک رویداد جدید استفاده می شوند.

1. **setReservationHandler**: این تابع یک درخواست HTTP را دریافت می کند که شامل اطلاعات رزرو بلیت است. این اطلاعات از بدنه درخواست خوانده می شود و سپس برای رزرو بلیت به سرویس بلیت ارسال می شود. در صورت موفقیت آمیز بودن، شناسه های بلیت رزرو شده به عنوان پاسخ برگردانده می شود.

2. **getEventsHandler**: این تابع فهرستی از رویدادهای موجود را از سرویس بلیت دریافت و به عنوان پاسخ برمی گرداند.

3. **createEventHandler**: این تابع یک درخواست HTTP را دریافت می کند که شامل اطلاعات یک رویداد جدید است. این اطلاعات از بدنه درخواست خوانده می شود و سپس برای ایجاد یک رویداد جدید به سرویس بلیت ارسال می شود. در صورت موفقیت آمیز بودن، شناسه رویداد جدید به عنوان پاسخ برگردانده می شود.

هر کدام از این توابع از متد Encode برای تبدیل داده های خروجی به JSON و ارسال آن به عنوان پاسخ استفاده می کنند. همچنین، در صورت بروز خطا در هر یک از این مراحل، پیام خطا به عنوان پاسخ برگردانده می شود.

- server.go:

این قسمت اختصاص داده شده به سرور است که یک HTTP را با استفاده از کتابخانه gorilla/mux راه اندازی می شود. سرور دارای سه مسیر HTTP است: apiSetReservation، apiCreateEvent و apiGetEvents که به توابع مربوطه متصل شده اند.

1. **Server struct**: این ساختار داده ای یک فیلد TicketService دارد که از نوع manager.TicketService است. این فیلد به سرویس بلیت دسترسی می دهد که برای ایجاد رویدادها، نمایش رویدادهای موجود، و رزرو بلیت ها استفاده می شود.

2. **SetupHttpApiServer**: این تابع یک سرور HTTP را با استفاده از gorilla/mux راه اندازی می کند. این سرور دارای سه مسیر است که به توابع مربوطه متصل شده اند. سرور بر روی آدرس مشخص شده (serverAddr) گوش می دهد و زمان خواندن و نوشتن را به 15 ثانیه تنظیم می کند. در نهایت، سرور با استفاده از ListenAndServe شروع به گوش دادن می کند.

هر کدام از این مسیرها به یک تابع مربوطه متصل است که در درخواست های HTTP مربوطه کار می کند. این توابع setReservationHandler، createEventHandler و getEventsHandler هستند که در کد قبلی شما تعریف شده اند.

- Iclient.go

این بخش یک رابط (interface) به نام IClient تعریف می کند. این interface چهار تابع را تعریف می کند که هر کلاسی که این interface را پیاده سازی کند باید این توابع را تعریف کند:

به طور کلی، این رابط IClient یک قرارداد برای ارتباط با سیستم رزرو بلیت است و هر کلاسی که این interface را پیاده سازی کند باید این توابع را تعریف کند.

- client.go

این بخش مربوط به کلاینت برای سیستم رزرو بلیت است. در اینجا توضیحات کلی در مورد هر بخش ارائه می‌شود:

1. **Client**: این ساختار داده‌ای یک URL سرور را نگه می‌دارد که برای ارسال درخواست‌ها به سرور استفاده می‌شود.

2. **NewClient**: این تابع یک نمونه جدید از ساختار Client را با URL سرور مشخص شده ایجاد می‌کند.

3. **GetEvents**: این تابع یک درخواست GET به سرور ارسال می‌کند تا لیستی از رویدادها را دریافت کند. پاسخ سرور را به یک لیست از ساختارهای Event تبدیل می‌کند.

4. **BookTicket**: این تابع یک درخواست POST به سرور ارسال می‌کند تا بلیت‌های مورد نظر را رزرو کند. اطلاعات مربوط به رزرو (شناسه رویداد و تعداد بلیت‌ها) را به عنوان بخش بدنه درخواست ارسال می‌کند.

5. **CreateEvent**: این تابع یک درخواست POST به سرور ارسال می‌کند تا یک رویداد جدید ایجاد کند. اطلاعات مربوط به رویداد (نام، تاریخ و تعداد کل بلیت‌ها) را به عنوان بخش بدنه درخواست ارسال می‌کند.

6. **BurstTest**: این تابع در حال حاضر پیاده‌سازی نشده است و با اجرای آن خطای "implement me" رخ خواهد داد.

هر تابع در صورت بروز خطا، پیام خطا را چاپ می‌کند و خطا را برمی‌گرداند. همچنین، اگر پاسخ سرور با کد وضعیت موفقیت‌آمیز (200) نباشد، پیام خطای مربوطه را چاپ می‌کند. در نهایت، هر تابع داده‌های دریافتی را از JSON به ساختارهای مربوطه تبدیل می‌کند.

(4) فولدر utils :

- **json handler** : در این فایل داده‌های رویدادها و بلیت‌ها را می‌خوانیم

- **generateUUID** : یک تابع برای تولید یک شناسه یکتا.

نتایج و خروجی برنامه :

در این قسمت مشاهده می‌کنیم که ایجاد کردن ایونت به درستی انجام می‌شود:

```
TicketReservation — ./client
~/goland/TicketReservation — ./client

**[Create new event]**
Name: ConferenceX
Date (YYYY-MM-DD hh:mm): 2027-12-10 10:30
Total tickets: 30

Event created successfully!
"Event ID: 50b4528d-11bc-467e-ad46-2f5dfcebe054 "

Press enter to return
```

```
TicketReservation — ./server
~/goland/TicketReservation — ./server

[→ TicketReservation git:(main) x ./server]
2024/05/03 22:42:58 Server is running on 127.0.0.1:8000
map[Date:2027-12-10 10:30 Name:ConferenceX totalTickets:30]
[Server] (create event)2024/05/03 22:43:54 Event created successfully: &{50b4528d-11bc-467
e-ad46-2f5dfcebe054 ConferenceX 2027-12-10 10:30:00 +0000 UTC 30 30}
```

مشاهده می کنیم که event نیز به درستی نمایش داده می شود:

```
TicketReservation — ./client
~/goland/TicketReservation — ./client client ✖ +
(1) 50b4528d-11bc-467e-ad46-2f5dfcebe054
    Name: ConferenceX
    Date: 2027-12-10 10:30
    Available Tickets: 30 of 30

Press enter to return
```

دوباره event جدید می سازیم:

```
TicketReservation — ./client
~/goland/TicketReservation — ./client +
**[Create new event]**
Name: WorkshopY
Date (YYYY-MM-DD hh:mm): 2024-06-01 14:45
Total tickets: 150

Event created successfully!
"Event ID: e738e163-7a36-422a-9d61-313f9b47b848 "

Press enter to return
|
```

```
TicketReservation — ./server
~/goland/TicketReservation — ./server

TicketReservation git:(main) x ./server
2024/05/03 22:42:58 Server is running on 127.0.0.1:8000
map[Date:2027-12-10 10:30 Name:ConferenceX totalTickets:30]
[Server] (create event)2024/05/03 22:43:54 Event created successfully: 8{50b4528d-11bc-467e-ad46-2f5dfcebe054 ConferenceX 2027-12-10 10:30:00 +0000 UTC 30 30}
[Server] (get events)2024/05/03 22:44:50 Events retrieved successfully.
map[Date:2024-06-01 14:45 Name:WorkshopY totalTickets:150]
[Server] (create event)2024/05/03 22:45:30 Event created successfully: 8{e738e163-7a36-422a-9d61-313f9b47b848 WorkshopY 2024-06-01 14:45:00 +0000 UTC 150 150}
```

در این قسمت می‌خواهیم که بلیت رزرو کنیم و همانطور که مشاهده می‌شود، هردو ایونت نمایش داده شده است:

```
TicketReservation — ./client
~/goland/TicketReservation — ./client

[x] ConferenceX
    Date: 2027-12-10 10:30
    Available Tickets: 30
    Total Tickets: 30
    (ID: 50b4528d-11bc-467e-ad46-2f5dfcebe054)

[ ] WorkshopY (2024-06-01 14:45)
    Available Tickets: 150
```

در این قسمت event ها برای نشان داده شدن لیست و فرستاده می شوند:

```
TicketReservation — ./server
~/goland/TicketReservation — ./server

[→ TicketReservation git:(main) x ./server
2024/05/03 22:42:58 Server is running on 127.0.0.1:8000
map[Date:2027-12-10 10:30 Name:ConferenceX totalTickets:30]
[Server] (create event)2024/05/03 22:43:54 Event created successfully: 6{50b4528d-11bc-467
e-ad46-2f5dfcebe054 ConferenceX 2027-12-10 10:30:00 +0000 UTC 30 30}
[Server] (get events)2024/05/03 22:44:50 Events retrieved successfully.
map[Date:2024-06-01 14:45 Name:WorkshopY totalTickets:150]
[Server] (create event)2024/05/03 22:45:30 Event created successfully: 6{e738e163-7a36-422
a-9d61-313f9b47b848 WorkshopY 2024-06-01 14:45:00 +0000 UTC 150 150}
[Server] (get events)2024/05/03 22:46:16 Events retrieved successfully.
```

به event دوم می رویم و 7 بلیت را رزرو می کنیم:

```
TicketReservation — ./client
~/goland/TicketReservation — ./client

[ ] ConferenceX (2027-12-10 10:30)
    Available Tickets: 30

[x] WorkshopY
    Date: 2024-06-01 14:45
    Available Tickets: 150
    Total Tickets: 150
    (ID: e738e163-7a36-422a-9d61-313f9b47b848)
```

```
TicketReservation — ./client
~/goland/TicketReservation — ./client
Booking Ticket for: e738e163-7a36-422a-9d61-313f9b47b848
  Name: WorkshopY
  Date: 2024-06-01 14:45
  Available Tickets: 150 of 150

Quantity: 7

Tickets booked successfully!
"Ticket ID:
( 1 ) 3b9a580f-e57e-45ff-a2f7-fb45c403780c
( 2 ) 1ad1a0d9-70df-4742-8d30-3183ffebf5ab
( 3 ) e7349ed0-97b9-4580-8cf4-21ca9ab94f66
( 4 ) e75eb8eb-f1ae-45ad-943c-c45909fd8863
( 5 ) f92f5a87-df11-4263-b827-1aa5ae7aa013
( 6 ) 025babb6-6169-4f31-a94e-34d786983ab0
( 7 ) f380229d-d4b4-4c2d-91d2-590a4d024cfd

Press enter to return
```

مشاهده می کنیم که لاگ سرور نیز به درستی انجام شده است:

```
TicketReservation — ./server
~/goland/TicketReservation — ./server
[+] TicketReservation git:(main) x ./server
2024/05/03 22:42:58 Server is running on 127.0.0.1:8000
map[Date:2027-12-10 10:30 Name:ConferenceX totalTickets:30]
[Server] (create event)2024/05/03 22:43:54 Event created successfully: 8{50b4528d-11bc-467e-ad46-2f5dfcebe054 ConferenceX 2027-12-10 10:30:00 +0000 UTC 30 30}
[Server] (get events)2024/05/03 22:44:50 Events retrieved successfully.
map[Date:2024-06-01 14:45 Name:WorkshopY totalTickets:150]
[Server] (create event)2024/05/03 22:45:30 Event created successfully: 8{e738e163-7a36-422a-9d61-313f9b47b848 WorkshopY 2024-06-01 14:45:00 +0000 UTC 150 150}
[Server] (get events)2024/05/03 22:46:16 Events retrieved successfully.
[Server] (ticket reservation)2024/05/03 22:47:16 Reservation successful.
```

در این قسمت بلیت ها را بیشتر از ظرفیت event رزرو میکنیم (51) و مشاهده می کنیم که رزرو fail می شود:

```
TicketReservation — ./client
~/goland/TicketReservation — ./client
Booking Ticket for: 50b4528d-11bc-467e-ad46-2f5dfcebe054
  Name: ConferenceX
  Date: 2027-12-10 10:30
  Available Tickets: 30 of 30

Quantity: 50
[Client] (bookTicket)2024/05/03 22:48:05 Error response status code: 400
Failed to book ticket(s)

Press enter to return
|
```

سرور علت fail شدن را لاگ می اندازد:

```
TicketReservation — ./server
~/goland/TicketReservation — ./server

TicketReservation git:(main) x ./server
2024/05/03 22:42:58 Server is running on 127.0.0.1:8000
map[Date:2027-12-10 10:30 Name:ConferenceX totalTickets:30]
[Server] (create event)2024/05/03 22:43:54 Event created successfully: 8{50b4528d-11bc-467e-ad46-2f5dfcebe054 ConferenceX 2027-12-10 10:30:00 +0000 UTC 30 30}
[Server] (get events)2024/05/03 22:44:50 Events retrieved successfully.
map[Date:2024-06-01 14:45 Name:WorkshopY totalTickets:150]
[Server] (create event)2024/05/03 22:45:30 Event created successfully: 8{e738e163-7a36-422a-9d61-313f9b47b848 WorkshopY 2024-06-01 14:45:00 +0000 UTC 150 150}
[Server] (get events)2024/05/03 22:46:16 Events retrieved successfully.
[Server] (ticket reservation)2024/05/03 22:47:16 Reservation successful.
[Server] (get events)2024/05/03 22:47:58 Events retrieved successfully.
[Server] (ticket reservation)2024/05/03 22:48:05 Failed to book tickets: not enough Tickets available
```

در نهایت مشاهده می کنیم که از event اول 7 بلیت رزرو شده و از event دوم بلیتی کم نشده است:


```
TicketReservation -- ./client
~/goland/TicketReservation -- ./client client +
(1) 50b4528d-11bc-467e-ad46-2f5dfcebe054
    Name: ConferenceX
    Date: 2027-12-10 10:30
    Available Tickets: 30 of 30
(2) e738e163-7a36-422a-9d61-313f9b47b848
    Name: WorkshopY
    Date: 2024-06-01 14:45
    Available Tickets: 143 of 150

Press enter to return
```

منوی help نیز در ادامه قرار داده شده است:

```
TicketReservation -- ./client
~/goland/TicketReservation -- ./client client +
**[Help menu]**

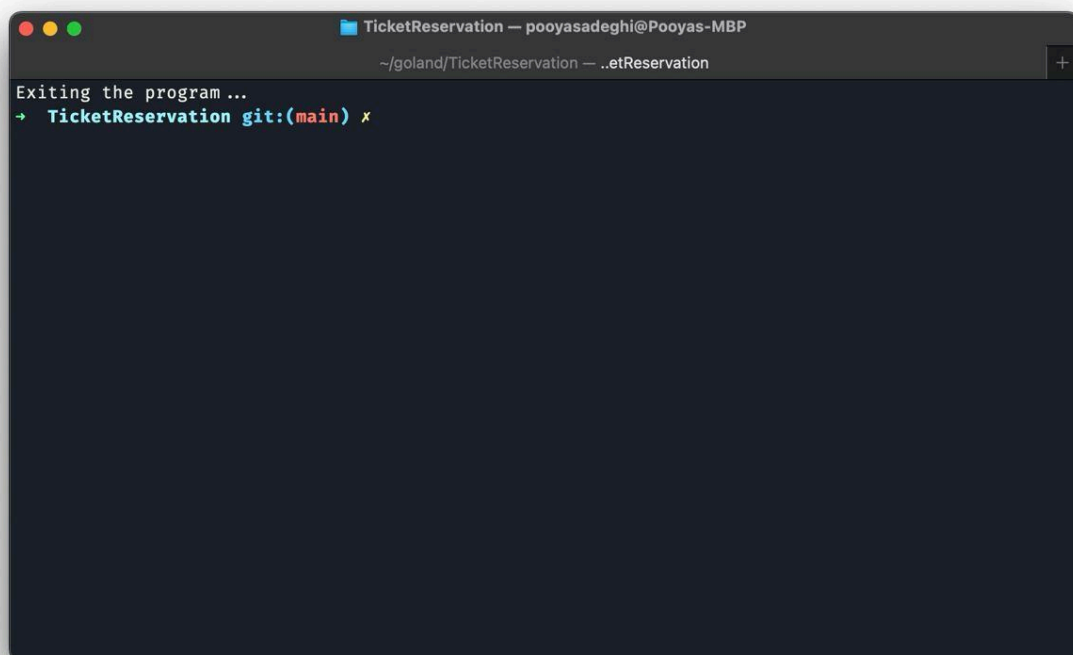
Your are in the interactive UI, here are what you can do:
1. Show events: You can see are events defined in the system, Events won't be filtered by date for
   ticket availability and sorted in random order.
2. Book ticket: You can book tickets for an event if enough number of ticket is remained.
3. Create new event: You can create a new event, you can define the name, date and total tickets f
   or it.
4. Help: You can see this menu again.
5. Exit: You can exit the program. You can also use "ctrl+c", "esc", "q" to exit.

There also exists a test mode, designed to test server inder pressure. You can't access that mode
here.
To enable test mode, run the program with the -test -client <number of parallel clients> -pressure
<number of request each client send in each test stage>
Example: ./client -test -client 5 -pressure 10
This will run 5 parallel clients, each sending 10 requests in each test stage.

To run client, use -p flag to set port. default is 8000 .

Press enter to return
|
```

همچنین هنگامی که اگزیت انجام بشود عملکرد برنامه بصورت زیر می باشد:



```
TicketReservation — pooyasadeghi@Pooyas-MBP
~/goland/TicketReservation — .etReservation
Exiting the program...
→ TicketReservation git:(main) x
```

تقسیم وظایف پروژه:

زهره و اولدوز منطق اولیه رو با توجه به نمونه کدهای توی صورت پروژه پیاده کردن و همچنین زهره فایل گزارش کار و اولدوز فایل README رو نوشتن. بخش ticketManager توسط اولدوز و handler توسط زهره پیاده سازی شد. پویا رابط REST کلاینت و سرور رو پیاده کرد و کلا کدهای سمت کلاینت با پویا بود و کمی هم از گزارش. مجید کدهای سمت سرور و دیتابیس مربوط به سرور و بخش cache رو پیاده سازی کرد و آزمون سرور تحت فشار رو انجام داد.