پیام‌رسان

استفاده از فریمورک Express مجاز است.

می‌خواهیم یک شبکه از کاربران مختلف داشته باشیم که بتوانند با یکدیگر چت کنند.

موجودیت‌ها

- **کاربر (User)**
  - همانند هر شبکه‌ی اجتماعی دیگری، این شبکه نیز از مجموعه‌ای از کاربران تشکیل شده است.
  - هر کاربر می‌تواند وارد یک گروه شود یا گروه خود را ایجاد کند.
  - کاربری که گروه را ایجاد کند، به‌عنوان ادمین آن گروه در نظر گرفته می‌شود.
- **گروه (Group)**
  - هر گروه شامل یک یا چند کاربر است. در این برنامه، هر کاربر تنها می‌تواند در یک گروه عضو باشد!
  - دو نقش در گروه میتواند وجود داشته باشد:
    - ادمین: سازنده‌ی گروه
    - کاربر عادی: کسی که سازنده‌ی گروه نیست
  - اضافه شدن یک شخص به یک گروه به این‌صورت خواهد بود که ابتدا شخصی که در گروهی عضو نیست درخواست اضافه شدن به گروه را می‌فرستد و ادمین می‌تواند درخواست را قبول کند.
  - دو گروه می‌توانند با یکدیگر ادغام شوند؛ به این‌صورت که ادمین یک گروه به گروه دیگر درخواست کانکشن بفرستد و در گروه دیگر، ادمین این درخواست را قبول کند.
  - افراد گروه‌های مختلف در صورتی که گروه‌هایشان با یکدیگر ادغام شده باشد، می‌توانند با یکدیگر چت خصوصی (دو نفره) داشته باشند.
- **چت باکس (ChatBox)**
  - هر دو کاربری که واجد شرایط باشند (که در ادامه به این شرایط اشاره خواهد شد) می‌توانند با یکدیگر چت کنند.

endpoint های *API*

نکته‌ی مهم: لطفاً *API* را با توجه به مستندات موجود در *Swagger* طراحی کنید (*route* ها، کد و ساختار پاسخ خروجی، خطاها و سایر جزییات را در نظر داشته باشید). حتی یک فیلد اضافه یا کم می‌تواند باعث شود شما نمره‌ی تست مربوطه را از دست بدهید. هم‌چنین انتظار داشته باشید که فرمت دیتاهای ورودی به شکلی باشد که در *Swagger* آمده است. برای سادگی کار، اگر در یک درخواست مشکلی وجود داشت و به دلایل مختلف (مانند نداشتن دسترسی کافی، یا این که کاربر در گروه دیگری عضو است، یا پسورد در لاگین اشتباه است، یا آی‌دی موجود نیست یا ...) فرآیند با مشکل مواجه شد، کد پاسخ ۴۰۰ را برگردانید. فرمت این خطا نیز در *Swagger* مشخص شده است.

احراز هویت

به‌جز دو *route* ثبت‌نام و لاگین، سایر *route* ها نیازمند احراز هویت هستند (*route* هایی که روی آن‌ها علامت قفل است). درخواست‌ها با توکن *JWT* شناخته خواهند شد. این توکن در *header* با کلید `Authorization` مقداردهی خواهد شد. حتماً در *payload* توکن، آی‌دی شخص را با کلید `userId` ذخیره کنید.

ثبت‌نام

کاربر از طریق این *endpoint* می‌تواند ثبت‌نام کند.

```
POST /api/v1/auth/signup
```

لاگین

کاربر از طریق این *endpoint* می‌تواند وارد حساب کاربری خود شود.

```
POST /api/v1/auth/login
```

ایجاد گروه

کاربر از طریق این *endpoint* می‌تواند گروه خود را ایجاد کند.

```
POST /api/v1/groups
```

اگر کاربر در گروهی عضو باشد، باید خطای `400` دریافت کند.

مشاهده‌ی لیست گروه‌ها

کاربر از طریق این *endpoint* می‌تواند لیست تمامی گروه‌ها را از جدید به قدیم مشاهده کند.

```
GET /api/v1/groups
```

مشاهده‌ی اطلاعات گروهی که شخص در آن عضو است

کاربر از طریق این *endpoint* می‌تواند اطلاعات گروهی که در آن عضو است را مشاهده کند.

```
GET /api/v1/groups/my
```

نکات

- اعضای گروه از جدیدترین به قدیمی‌ترین باید نمایش داده شوند (زمان وارد شدن به گروه).
- اگر کاربر در گروهی عضو نبود، باید خطای `400` دریافت کند.
- کاربری که `owner` نیست باید نقشش `normal` در نظر گرفته شود.

ارسال درخواست عضویت

کاربر از طریق این *endpoint* می‌تواند به گروهی درخواست عضو شدن ارسال کند.

```
POST /api/v1/join_requests
```

نکته: اگر کاربر در گروهی عضو بود، باید خطای `400` دریافت کند.

مشاهده‌ی درخواست‌های عضویت خود

کاربر از طریق این *endpoint* می‌تواند تمامی درخواست‌های عضویت خود را که به سایر گروه‌ها ارسال کرده است را مشاهده کند.

```
GET /api/v1/join_requests
```

نکته: درخواست‌ها باید از جدیدترین به قدیمی‌ترین نمایش داده شوند.

مشاهده‌ی درخواست‌های عضویت ارسال‌شده به گروه

کاربر از طریق این *endpoint* می‌تواند درخواست‌های ارسال‌شده به گروهش را مشاهده کند.

```
GET /api/v1/join_requests/group
```

نکات

- درخواست‌ها باید از جدیدترین به قدیمی‌ترین نمایش داده شوند.
- اگر کاربر صاحب گروهی نبود، باید خطای `400` دریافت کند.

تأیید کردن درخواست ارسال‌شده به گروه برای عضویت

کاربر از طریق این *endpoint* می‌تواند درخواست عضویت در گروهش توسط شخص دیگری را قبول کند.

```
POST /api/v1/join_requests/accept
```

نکته: اگر کاربر صاحب گروهی نبود، باید خطای `400` دریافت کند.

ارسال درخواست ادغام با گروهی دیگر

کاربر از طریق این *endpoint* می‌تواند به گروه دیگری درخواست ادغام بدهد.

```
POST /api/v1/connection_requests
```

نکته: اگر کاربر صاحب گروهی نبود، باید خطای `400` دریافت کند.

دریافت لیست درخواست‌های ادغام با گروه خود

کاربر از طریق این *endpoint* می‌تواند درخواست‌های ادغام با گروه خود را مشاهده کند.

```
GET /api/v1/connection_requests
```

نکات

- درخواست‌ها باید از جدیدترین به قدیمی‌ترین نمایش داده شوند.
- اگر کاربر صاحب گروهی نبود، باید خطای `400` دریافت کند.

تأیید کردن درخواست ادغام گروه دیگر با گروه خود

کاربر از طریق این *endpoint* می‌تواند ادغام گروه دیگری با گروه خود را بپذیرد.

```
POST /api/v1/connection_requests/accept
```

نکته: اگر کاربر صاحب گروهی نبود، باید خطای `400` دریافت کند.

چت کردن با شخصی دیگر

کاربر از طریق این *endpoint* می‌تواند به شخص دیگری پیام ارسال کند.

```
POST /api/v1/chats/{user_id}
```

نکته: در صورتی که دو کاربر در گروه‌هایی عضو باشند که آن دو گروه با یکدیگر متصل نباشند، آن‌گاه باید خطای `400` داده شود.

دریافت پیام‌های موجود در چت با شخصی دیگر

کاربر از طریق این *endpoint* می‌تواند پیام خود در چت با کاربر دیگری را مشاهده کند.

```
GET /api/v1/chats/{user_id}
```

نکات

- پیام‌ها باید از جدیدترین به قدیمی‌ترین نمایش داده شوند.
- اگر با شخص موردنظر تا به حال چت صورت نگرفته باشد یا آن شخص موجود نباشد، باید خطای `400` داده شود.

دریافت لیست چت‌های خود با دیگران

کاربر از طریق این *endpoint* می‌تواند لیست چت‌های خود با سایر کاربران را مشاهده کند.

```
GET /api/v1/chats
```

نکات

- چت‌ها باید از جدیدترین به قدیمی‌ترین نمایش داده شوند. آخرین پیام دریافتی یا ارسالی ملاک است.
- اگر شخص تا به حال با هیچ فردی چت نداشته است، باید یک آرایه‌ی خالی برگردانده شود.

نیازی به *persistent* بودن داده‌های برنامه نیست، اما استفاده از دیتابیس برای نگه‌داری داده‌ها امتیازی است.

آن‌چه باید آپلود کنید

پس از پیاده‌سازی *API* ، محتویات پوشه‌ی اصلی پروژه را به‌جز پوشه‌ی `node_modules` زیپ کرده و آپلود کنید.