

## MODELS:

### ⇒ User:















```
alias: {
  type: String,
  required: true,
  trim: true,
  maxlength: 20
},
email: {
  type: String,
  required: true,
  trim: true,
  unique: true
},
salt: String,
encry_password: {
  type: String,
  required: true
},
userList: {
  type: [users' id in 1 km radius],
  ref: "User",
  unique: false
},
coords: {
  type: [latitude, longitude],
  default: null,
  unique: false
}
```

### ⇒ City:










```
name: {
  type: String,
  required: true
},
users: {
  type: [users' id in a city],
  ref: "User",
  unique: false
}
```

## OBJECTS:

### ⇒ User:

 _id	ObjectId("613091d1c25b331390098749")	ObjectId
▼  userList	[ 2 elements ]	Array
 [0]	ObjectId("6130940300e4e10c0c0436d7")	ObjectId
 [1]	ObjectId("6130d8ac1e75793f1cf47973")	ObjectId
▼  coords	[ 2 elements ]	Array
 [0]	0.111	Double
 [1]	0.113	Double
 alias	user1	String
 email	user1@gmail.com	String
 salt	b5a09770-0bcb-11ec-bca0-679d15c3747b	String
 encry_password	5f84fb4009b7c49c8ad8f19f2d8b4414183df4aa58f2deff0a3b7d12286e5...	String
 createdAt	2021-09-02 08:56:49.515Z	Date
 updatedAt	2021-09-03 10:50:20.744Z	Date
 _v	13	Int32

### ⇒ City:

 _id	ObjectId("6130c7b3cba31b0c388ceab8")	ObjectId
▼  users	[ 3 elements ]	Array
 [0]	ObjectId("613091d1c25b331390098749")	ObjectId
 [1]	ObjectId("6130940300e4e10c0c0436d7")	ObjectId
 [2]	ObjectId("6130d8ac1e75793f1cf47973")	ObjectId
 name	delhi	String
 createdAt	2021-09-02 12:46:43.091Z	Date
 updatedAt	2021-09-03 06:20:00.714Z	Date
 _v	4	Int32

**BACKEND PORT- 8000, URL- localhost:8000/api**

**API:**

⇒ **SIGNUP:**

**API** - /signup

**Method** – Post

**Body** – {

```
    "alias": "value with at least 3 characters",  
    "email": "valid email id"  
    "password": "value with at least 3 characters"  
}
```

**Headers** – Content-Type: application/json

**Description** – for user signup. Create user in Database using user model

**Response list** –

- a. {error: "User with this email already exist"}
- b. {  
 msg: "user successfully created"  
 user: created user object  
}

⇒ **SIGNIN:**

1.

**API** - /signin

**Method** - Post

**Body** – {

```
    "email": "valid email id"  
    "password": "value with at least 3 characters"  
}
```

**Headers** – Content-Type: application/json

**Description** – for user sign in.

**Response list** –

- a. {error: "no user is registered with this email"  
 {error: "email and password does not match"}}
- b. { token (authentication token),  
 user: { \_id, alias, email }  
}

2.

**API** - /city/:userId

**Method** - Post

**Body** – {

“name”: “city name when sign in”

}

**Headers** –

a. Content-Type: application/json

b. Authorization: Bearer token (which we got on sign in)

**Description** – Add the signed in user to the city.users array.

If no such city is there in DB it will create one and then add user.

**Response list** –

a. {error: “there already exist a user with this id in the city list”}

b. {

sucess: “The city list is updated”,

city: city object

}

3.

**API** - /user/updatelist/:userId

**Method** - Put

**Body** – {

“latitude”: number,

“longitude”: number,

“city”: “user’s city name”

}

**Headers** –

a. Content-Type: application/json

b. Authorization: Bearer token (which we got on sign in)

**Description** – Update user “coords” array.

Update every user’s userList array present in the city.

**Response list** –

a. {error: “there is no such city in DB”}

b. {success: “user lists successfully updated”}

⇒ **WHILE LOGGED IN:**

1. When user's position changes

**API** - /user/updatelist/:userId

**Method** - Put

**Body** – {

```
    "latitude": number,  
    "longitude": number,  
    "city": "user's city name"
```

}

**Headers** –

- a. Content-Type: application/json
- b. Authorization: Bearer token (which we got on sign in)

**Description** – Update user "coords" array.

Update every user's userList array present in the city.

**Response list** –

- a. {error: "there is no such city in DB"}
- b. {success: "user lists successfully updated"}

2. Get list of users in 1 km radius

**API** - /user/list/:userId

**Method** - Get

**Headers** –

- a. Authorization: Bearer token (which we got on sign in)

**Description** – get list of all user objects present in userList.

**Response list** – [user objects in userList]

⇒ **SIGN OUT:**

1.

**API** - /signout/:userId

**Method** - Post

**Body** – {

“city”: “user’s city name”

}

**Headers** –

- a. Content-Type: application/json
- b. Authorization: Bearer token (which we got on sign in)

**Description** – Remove user id from all other user’s userList present in the user’s userList.

Remove user’s id from city users list.

Empty user’s userList array.

Clear authentication token.

Sign out user.

**Response list** –

- a. {message: “user signout successfully”}