













-  High-Level Architecture:
  -  In User Service:
  -  In Publication Service:
-  User Service
  -  Connect `user_events_queue`:
  -  Emit events in user service:
-  Publication Service
  -  `users-events.controller.ts`:
  -  `user-cache.service.ts`:
  -  `user.interface.ts`:
  -  `users.module.ts`:
  -  `users.controller.ts`:

## High-Level Architecture:

---

### In User Service:

- Use `.emit()` when user is created/updated.
- Configure a second RabbitMQ queue (`user_events_queue`) .

### In Publication Service:

- Listen to events from `user_events_queue` .
- Receive data and store in in-memory cache via `user-cache.service.ts`.

## User Service

---

### Connect `user_events_queue`:

In `main.ts`, add another microservice connection:

```
app.connectMicroservice<MicroserviceOptions>({
  transport: Transport.RMQ,
  options: {
```

```

    urls: ['amqp://user:password@localhost:5672'],
    queue: 'user_events_queue',
    queueOptions: { durable: true },
  },
});

```



## Emit events in user service:

In your UserService (**Register**, **updateUser**, etc.):

```

constructor(
@Inject('PUBLICATION_EVENTS_SERVICE') private readonly userEventsClient:
ClientProxy
) {}
async createUser(dto: CreateUserDto) {
  const user = await this.userRepository.save(dto); // Emit event to publication
  service
  this.userEventsClient.emit('user_created', {
    id: user.id,
    name: user.name,
    email: user.email,
    // any other fields
  }); return user;
}

```

Register **PUBLICATION\_SERVICE** using **ClientsModule** with the same **user\_events\_queue**. the module with the same folder we have service like **user.module**

```

import { Module } from '@nestjs/common';
import { ClientsModule, Transport } from '@nestjs/microservices';

@Module({
  imports: [
    ClientsModule.register([
      {
        name: 'PUBLICATION_EVENTS_SERVICE', // 👉 must match the Inject() token
        transport: Transport.RMQ,
        options: {
          urls: ['amqp://user:password@localhost:5672'],
          queue: 'user_events_queue', // 🚩 This is the queue to emit events to
          queueOptions: {
            durable: true,
          },
        },
      },
    ]),
  ],
})

```

```
providers: [  
  // Your service that emits events (e.g., UserService)  
  ],  
  exports: [ClientsModule], // Export it if needed elsewhere  
})  
export class UserModule {}
```



## Publication Service

 - Do this Schema :

name\_ms/src/

-> users/

->user-cache.service.ts

-> user.interface.ts

-> users-events.controller.ts

-> users.controller.ts

-> users.module.ts



### users-events.controller.ts:

This listens for user events and updates the cache:

```
// publication-service/src/users/users-events.controller.ts  
import { Controller } from '@nestjs/common';  
import { EventPattern, Payload } from '@nestjs/microservices';
```

```
import { UserCacheService } from './user-cache.service';
import { User } from './user.interface';

@Controller('users-events')
export class UsersEventsController {
  constructor(private readonly userCacheService: UserCacheService) {}

  @EventPattern('user_created')
  async handleUserCreated(@Payload() user: User) {
    console.log('[PUBLICATION_SERVICE] 👤 Received user_created event', user);
    this.userCacheService.addUserToCache(user);
  }

  @EventPattern('user_updated')
  async handleUserUpdated(@Payload() user: User) {
    console.log('[PUBLICATION_SERVICE] 🔄 Received user_updated event', user);
    this.userCacheService.updateUserInCache(user); // same method can update
  }
}
```



## user-cache.service.ts:

- You will find here two methods you can use to get any data from table user  
`getUserById(id: number)` & `getAllUsers()`
- How to use it , First add this to any controller or service where you want to use  
`privatereadonlyuserCacheService: UserCacheService`

and add the methode like this

```
`const users = ` `this.userCacheService.getUserById(id);`

`const users = this.userCacheService.getAllUsers();`
```

Simple in-memory caching:

```
// publication-service/src/users/user-cache.service.ts
import { Injectable, OnModuleInit, OnModuleDestroy } from '@nestjs/common';
import { Inject } from '@nestjs/common';
import { ClientProxy } from '@nestjs/microservices';
import { firstValueFrom } from 'rxjs';
import { User } from './user.interface';

@Injectable()
// export class UserCacheService implements OnModuleInit {
  export class UserCacheService implements OnModuleInit, OnModuleDestroy {
    private userCache: Map<number, User> = new Map();
    private refreshTimer: NodeJS.Timeout;
```

```

constructor(
  @Inject('USER_SERVICE') private readonly userServiceClient: ClientProxy,
) {
  // Optionally schedule cache clearing to prevent memory bloat
  this.scheduleNextMidnightRefresh(); // Optional
}

async onModuleInit() {
  // Fetch users when the service starts
  await this.loadInitialUsers();

  // Schedule daily cache refresh at midnight
  this.scheduleNextMidnightRefresh();
}

// Cleanup when module is destroyed
onModuleDestroy() {
  if (this.refreshTimer) {
    clearTimeout(this.refreshTimer);
  }
}

// Schedule the next midnight refresh
private scheduleNextMidnightRefresh() {
  const now = new Date();
  const tomorrow = new Date(now);
  tomorrow.setDate(tomorrow.getDate() + 1);
  tomorrow.setHours(0, 0, 0, 0); // Set to midnight

  const timeUntilMidnight = tomorrow.getTime() - now.getTime();

  this.refreshTimer = setTimeout(async () => {
    console.log('Performing scheduled midnight cache refresh');
    this.clearCache();
    // await this.loadInitialUsers();

    // Schedule the next day's refresh
    this.scheduleNextMidnightRefresh();
  }, timeUntilMidnight);

  console.log(`Next cache refresh scheduled in ${Math.floor(timeUntilMidnight /
1000 / 60)} minutes`);
}

// Clear the entire cache
private clearCache() {
  const userCount = this.userCache.size;
  this.userCache.clear();
  console.log(`[CACHE] 🗑️ Cleared ${userCount} users at midnight`);
}

// Add methods to be called from event handlers
addUserToCache(user: User) {
  this.userCache.set(user.id, {
    ...user,
    nomCompleat: `${user.nom} ${user.prenom}`
  });
  console.log(`[CACHE] ➕ User ${user.id} added`);
}

```

```

updateUserInCache(user: User) {
  this.userCache.set(user.id, {
    ...user,
    nomComple: `${user.nom} ${user.prenom}`
  });
  console.log(`[CACHE] 🔄 User ${user.id} updated`);
}

removeUserFromCache(userId: number) {
  this.userCache.delete(userId);
  console.log(`[CACHE] ❌ User ${userId} removed`);
}

private async loadInitialUsers() {
  try {
    console.log(`[CACHE] 📦 Loading initial users...`);

    // Send Request to get all users via RabbitMQ from user service
    const response = this.userServiceClient.send({ cmd: 'get_all_users' }, {});
    console.log('Request sent to user service, waiting for response...');

    const users = await firstValueFrom(response);
    console.log('Received response from user service:', users);

    if (!users || users.length === 0) {
      console.warn(`[CACHE] ⚠️ No users found`);
      return;
    }
    // Add users to the cache
    users.forEach(user => {
      this.addUserToCache(user);
    });

    console.log(`[CACHE] ✅ Loaded ${users.length} users`);
    console.log('Current cache size:', this.userCache.size);
  } catch (error) {
    console.error(`[CACHE] ❌ Failed to load initial users:`, error);
    console.error('Error details:', error.message);
    if (error.stack) console.error(error.stack);
  }
}

getUserById(id: number): User | undefined {
  const numericId = +id;
  return this.userCache.get(numericId);
}

getAllUsers(): User[] {
  return Array.from(this.userCache.values());
}

```

```
// publication-service/src/users/user.interface.ts
export interface User {
  id: number;
  email: string;
  nom: string;
  prenom: string;
  role: string;
  nomComple?: string; // We'll compute this from nom and prenom
}
```

## ✓ **users.module.ts:**

```
// publication-service/src/users/users.module.ts
import { Module } from '@nestjs/common';
import { ClientsModule, Transport } from '@nestjs/microservices';
import { UserCacheService } from '../user-cache.service';
import { UsersEventsController } from '../users-events.controller';
import { UsersController } from '../users.controller';

@Module({
  imports: [
    ClientsModule.register([
      {
        name: 'USER_SERVICE',
        transport: Transport.RMQ,
        options: {
          urls: ['amqp://user:password@localhost:5672'],
          queue: 'user_events_queue',
          queueOptions: {
            durable: true
          }
        },
      },
    ]),
  ],
  controllers: [UsersEventsController, UsersController],
  providers: [UserCacheService],
  exports: [UserCacheService],
})
export class UsersModule {}
```

## ✓ **users.controller.ts:**

Use this if you want to send values to Frontend :

```
// publication-service/src/users/users.controller.ts
import { Controller, Get, Param, NotFoundException } from '@nestjs/common';
import { UserCacheService } from '../user-cache.service';

@Controller('users')
export class UsersController {
  constructor(private readonly userCacheService: UserCacheService) {}

  // Get user by ID
  @Get('/:id')
  getUserById(@Param('id') id: number) {
    const user = this.userCacheService.getUserById(id);
    if (!user) {
      throw new NotFoundException(`User with ID ${id} not found`);
    }
    return user;
  }
  // Get all users
  @Get()
  getAllUsers() {
    const users = this.userCacheService.getAllUsers();
    return users;
  }
}
```