User Notification Preferences API Challenge

Here's a comprehensive plan to implement the **User Notification Preferences API Challenge**:

Plan and Breakdown

1. Project Setup

1. **Initialize the Project:**
    - Create a new Nest.js project: `nest new user-notification-api`.

Install required dependencies:
bash
Copy code
```
npm install @nestjs/mongoose mongoose class-validator
class-transformer @nestjs/platform-express @nestjs/config
```

    -

Add `dotenv` for environment variable management:
bash
Copy code
```
npm install dotenv
```

    -

2. **Setup MongoDB Integration:**
    - Add MongoDB connection logic in `app.module.ts` using `MongooseModule.forRoot()`.
3. **Environment Variables:**

Create a `.env` file for MongoDB URI and other config:
env
Copy code
```
MONGO_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/<dbn
ame>?retryWrites=true&w=majority
NODE_ENV=development
```

    -

- Use `@nestjs/config` to load `.env` variables.

---

2. API Design and Implementation

A. Models (Schemas)

UserPreference Schema: Define the schema using `@nestjs/mongoose` and validation with `class-validator`:

typescript
Copy code

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { IsEmail, IsEnum, IsObject, IsString } from 'class-validator';

@Schema({ timestamps: true })
export class UserPreference {
  @Prop({ required: true })
  @IsString()
  userId: string;

  @Prop({ required: true, unique: true })
  @IsEmail()
  email: string;

  @Prop({ type: Object, required: true })
  @IsObject()
  preferences: {
    marketing: boolean;
    newsletter: boolean;
    updates: boolean;
    frequency: 'daily' | 'weekly' | 'monthly' | 'never';
    channels: { email: boolean; sms: boolean; push: boolean };
  };

  @Prop({ required: true })
  @IsString()
  timezone: string;

  @Prop()
  lastUpdated: Date;
```

```
}
export const UserPreferenceSchema =
SchemaFactory.createForClass(UserPreference);
```

1.

NotificationLog Schema:
typescript
Copy code

```
@Schema({ timestamps: true })
export class NotificationLog {
  @Prop({ required: true })
  userId: string;

  @Prop({ required: true })
  type: 'marketing' | 'newsletter' | 'updates';

  @Prop({ required: true })
  channel: 'email' | 'sms' | 'push';

  @Prop({ default: 'pending' })
  status: 'pending' | 'sent' | 'failed';

  @Prop()
  sentAt?: Date;

  @Prop()
  failureReason?: string;

  @Prop({ type: Object })
  metadata: Record<string, any>;
}
export const NotificationLogSchema =
SchemaFactory.createForClass(NotificationLog);
```

2.

B. CRUD API for User Preferences

1. **Endpoints:**
    - `POST /api/preferences`: Create user preferences.
    - `GET /api/preferences/:userId`: Retrieve preferences for a user.
    - `PATCH /api/preferences/:userId`: Update preferences.
    - `DELETE /api/preferences/:userId`: Delete preferences.
2. **Service and Controller Implementation:**
    - Create a `PreferencesService` to handle CRUD operations.
    - Use `MongooseModel` for database operations.
    - Implement `PreferencesController` to define route handlers.
3. **Validation:** Use `class-validator` DTOs for request validation.

C. Notification Management

1. **Endpoints:**
    - `POST /api/notifications/send`: Simulate sending notifications.
    - `GET /api/notifications/:userId/logs`: Fetch notification logs for a user.
    - `GET /api/notifications/stats`: Provide basic notification stats.
2. **Logic:**
    - Simulate sending notifications with logs.
    - Include fields like `status`, `failureReason`, and `sentAt`.

---

3. Middleware

- **Rate Limiting:** Use `nestjs-rate-limiter` to throttle requests.
- **Logging:** Add logging middleware to track all incoming requests.

---

4. Testing

1. **Unit Tests:**
    - Test `PreferencesService` and `NotificationService` methods.
    - Validate DTOs for edge cases.
    - Mock MongoDB operations.
2. **Integration Tests:**
    - Test API endpoints with `SuperTest`.
    - Validate database integration using an in-memory MongoDB instance (`mongodb-memory-server`).

---

5. Deployment

1. **Vercel Configuration:**

Add `vercel.json` for API deployment:
json
Copy code

```
{
  "builds": [
    { "src": "dist/main.js", "use": "@vercel/node" }
  ],
  "routes": [{ "src": "/api/(.*)", "dest": "dist/main.js" }]
}
```

   -
   - Deploy with: `vercel --prod`.
2. **AWS Lambda (Optional):**
   - Use `@nestjs/serverless` for Lambda deployment.

---

6. Bonus Features

- **OpenAPI/Swagger:**

Add Swagger module for API docs:
bash
Copy code

```
npm install @nestjs/swagger
```

   -
   - Setup Swagger in `main.ts`.
- **Authentication:** Add API Key validation middleware.
- **Queueing Simulation:** Integrate `Bull` for notification queue simulation.

Deliverables

1. **GitHub Repository:**
   - Include source code, tests, `README.md`, `.env.example`, and setup instructions.

    - Add a Postman collection (optional).
2. **Deployed API:**
    - Provide the live API URL (Vercel or AWS Lambda).
3. **Example Requests/Responses:** Include in `README.md` and Postman collection.