HƯỚNG DẪN CHI TIẾT: FIREBASE & RESTFUL API VỚI .NET FRAMEWORK

Mục đích: Tài liệu này cung cấp kiến thức nền tảng về Firebase, Firebase Cloud Firestore và cách xây dựng RESTful API với .NET Framework để hỗ trợ sinh viên thực hiện các đề thi.

MỤC LỤC

1. Tổng quan về Hệ sinh thái Firebase
2. Firebase Cloud Firestore - Chi tiết
3. Xây dựng RESTful API với .NET Framework
4. So sánh Firestore vs SQL Database
5. Best Practices và Tips

---

1. TỔNG QUAN VỀ HỆ SINH THÁI FIREBASE

1.1. Firebase là gì?

Firebase là một nền tảng (Platform) được phát triển bởi Google, cung cấp các dịch vụ backend-as-a-service (BaaS) giúp phát triển ứng dụng nhanh chóng mà không cần xây dựng backend từ đầu.

Đặc điểm chính:

- ☑ **No-Server Management**: Không cần quản lý server
- ☑ **Real-time Database**: Cơ sở dữ liệu real-time
- ☑ **Scalable**: Tự động mở rộng theo nhu cầu
- ☑ **Cross-platform**: Hỗ trợ iOS, Android, Web
- ☑ **Free Tier**: Có gói miễn phí cho phát triển

1.2. Các dịch vụ chính trong Firebase

Firebase Authentication

- Xác thực người dùng (Email/Password, Google, Facebook, Phone, v.v.)
- Quản lý session và token
- **Khi nào dùng**: Khi cần đăng nhập/đăng ký

Cloud Firestore

- NoSQL database real-time
- Lưu trữ dữ liệu dạng document (JSON-like)
- **Khi nào dùng**: Khi cần database real-time, linh hoạt schema

Realtime Database

- NoSQL database real-time (cũ hơn Firestore)
- Dữ liệu dạng JSON tree
- **Khi nào dùng**: Ứng dụng real-time đơn giản

Cloud Storage

- Lưu trữ file (hình ảnh, video, documents)
- **Khi nào dùng**: Upload/download files

Cloud Functions

- Serverless functions (chạy code backend)
- **Khi nào dùng**: Xử lý logic phức tạp, background jobs

Cloud Messaging (FCM)

- Push notifications
- **Khi nào dùng**: Gửi thông báo đến thiết bị

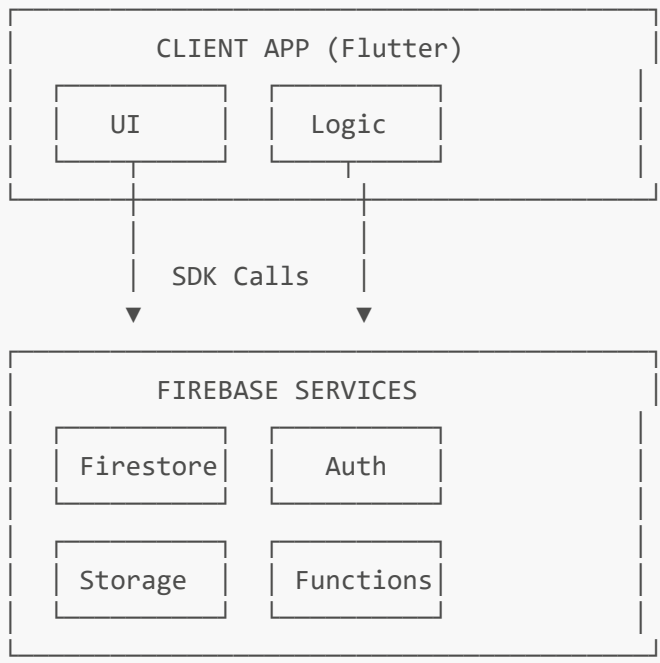
Analytics

- Phân tích hành vi người dùng
- **Khi nào dùng**: Tracking, analytics

Hosting

- Hosting cho web app
- **Khi nào dùng**: Deploy website tĩnh

1.3. Kiến trúc Firebase



1.4. Tại sao chọn Firebase?

Ưu điểm:

1. **Phát triển nhanh**: Setup trong vài phút
2. **Real-time**: Dữ liệu sync tự động
3. **Offline Support**: Hoạt động offline, sync khi online
4. **Security Rules**: Bảo mật dữ liệu ở phía server
5. **Miễn phí**: Gói Spark (free) đủ cho học tập

Nhược điểm:

1. **Chi phí**: Có thể tốn kém khi scale lớn
2. **Vendor Lock-in**: Phụ thuộc vào Google
3. **Query hạn chế**: Không mạnh như SQL
4. **Không có JOIN**: Phải tự xử lý relationships

---

2. FIREBASE CLOUD FIRESTORE - CHI TIẾT

2.1. Firestore là gì?

Cloud Firestore là NoSQL document database của Firebase, lưu trữ dữ liệu dạng **documents** (tương tự JSON objects) trong **collections**.

Cấu trúc dữ liệu:

```
Database
  └── Collection (customers)
        └── Document (customerId: "abc123")
              ├── Field: email = "user@example.com"
              ├── Field: fullName = "Nguyễn Văn A"
              └── Field: loyaltyPoints = 100
        └── Document (customerId: "def456")
              └── ...
  └── Collection (menu_items)
        └── Document (itemId: "item001")
              └── ...
```

2.2. Các khái niệm cơ bản

Collection

- Tương đương **Table** trong SQL
- Chứa nhiều **Documents**
- Ví dụ: `customers`, `menu_items`, `reservations`

Document

- Tương đương **Row** trong SQL
- Chứa các **Fields** (key-value pairs)
- Có **Document ID** duy nhất
- Ví dụ: Một customer cụ thể

Field

- Tương đương **Column** trong SQL
- Key-value pair
- Có thể là: String, Number, Boolean, Timestamp, Array, Map, Null

Subcollection

- Collection nằm trong một Document
- Dùng để tổ chức dữ liệu có quan hệ
- Ví dụ: `customers/{customerId}/orders/{orderId}`

2.3. Các kiểu dữ liệu trong Firestore

```
// String
"fullName": "Nguyễn Văn A"

// Number (Integer hoặc Double)
"loyaltyPoints": 100
"price": 80000.0

// Boolean
"isActive": true
"isVegetarian": false

// Timestamp
"createdAt": Timestamp.now()

// Array
"preferences": ["vegetarian", "spicy", "seafood"]
"ingredients": ["Thịt bò", "Hành tây", "Gia vị"]

// Map (Object)
"orderItems": [
  {
    "itemId": "item001",
    "itemName": "Phở Bò",
    "quantity": 2,
    "price": 80000.0
  }
]

// Null
"tableNumber": null
```

2.4. Thiết lập Firestore trong Flutter

Bước 1: Tạo Firebase Project

1. Truy cập Firebase Console
2. Click "Add project"
3. Nhập tên project (ví dụ: `restaurant-app-2151061234`)
4. Chọn Google Analytics (tùy chọn)
5. Click "Create project"

Bước 2: Thêm Flutter App vào Firebase

1. Trong Firebase Console, click "Add app" → chọn Flutter
2. Đăng ký app với package name: `com.example.2151061234`
3. Download file `google-services.json` (Android) và `GoogleService-Info.plist` (iOS)
4. Đặt file vào:
    - Android: `android/app/google-services.json`
    - iOS: `ios/Runner/GoogleService-Info.plist`

Bước 3: Cài đặt Dependencies

Thêm vào `pubspec.yaml`:

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^2.24.2
  cloud_firestore: ^4.13.6
  shared_preferences: ^2.2.2
```

Chạy:

```
flutter pub get
```

Bước 4: Khởi tạo Firebase trong Flutter

Trong `main.dart`:

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyApp());
}
```

Bước 5: Tạo Firestore Database

1. Trong Firebase Console → Firestore Database
2. Click "Create database"
3. Chọn chế độ:
   - **Production mode**: Bảo mật cao (cần Security Rules)
   - **Test mode**: Cho phép đọc/ghi trong 30 ngày (dùng cho test)
4. Chọn location (ví dụ: `asia-southeast1` - Singapore)

2.5. CRUD Operations với Firestore

2.5.1. CREATE - Thêm Document

```
import 'package:cloud_firestore/cloud_firestore.dart';

// Thêm document với ID tự động
Future<void> addCustomer() async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').add({
    'email': 'customer@example.com',
    'fullName': 'Nguyễn Văn A',
    'phoneNumber': '0123456789',
    'address': '123 Đường ABC',
    'preferences': ['vegetarian', 'spicy'],
    'loyaltyPoints': 0,
    'createdAt': FieldValue.serverTimestamp(),
    'isActive': true,
  });
}

// Thêm document với ID tùy chỉnh
Future<void> addCustomerWithId(String customerId) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).set({
    'email': 'customer@example.com',
    'fullName': 'Nguyễn Văn A',
    'phoneNumber': '0123456789',
    'address': '123 Đường ABC',
    'preferences': ['vegetarian', 'spicy'],
    'loyaltyPoints': 0,
    'createdAt': FieldValue.serverTimestamp(),
    'isActive': true,
  });
}
```

2.5.2. READ - Đọc Document

```dart
// Đọc một document theo ID
Future<Map<String, dynamic>?> getCustomer(String customerId) async {
  final db = FirebaseFirestore.instance;

  final doc = await db.collection('customers').doc(customerId).get();

  if (doc.exists) {
    return doc.data();
  }
  return null;
}

// Đọc tất cả documents trong collection
Future<List<Map<String, dynamic>>> getAllCustomers() async {
  final db = FirebaseFirestore.instance;

  final snapshot = await db.collection('customers').get();

  return snapshot.docs.map((doc) => {
    'id': doc.id,
    ...doc.data() as Map<String, dynamic>
  }).toList();
}

// Đọc với điều kiện (Query)
Future<List<Map<String, dynamic>>> getActiveCustomers() async {
  final db = FirebaseFirestore.instance;

  final snapshot = await db.collection('customers')
      .where('isActive', isEqualTo: true)
      .get();

  return snapshot.docs.map((doc) => {
    'id': doc.id,
    ...doc.data() as Map<String, dynamic>
  }).toList();
}

// Đọc với sắp xếp và giới hạn
Future<List<Map<String, dynamic>>> getTopCustomers() async {
  final db = FirebaseFirestore.instance;

  final snapshot = await db.collection('customers')
      .orderBy('loyaltyPoints', descending: true)
      .limit(10)
      .get();

  return snapshot.docs.map((doc) => {
    'id': doc.id,
    ...doc.data() as Map<String, dynamic>
  }).toList();
}
```

2.5.3. UPDATE - Cập nhật Document

```
// Cập nhật một số fields
Future<void> updateCustomer(String customerId, Map<String, dynamic> data) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).update({
    ...data,
    'updatedAt': FieldValue.serverTimestamp(),
  });
}

// Cập nhật với merge (không ghi đè toàn bộ)
Future<void> updateCustomerMerge(String customerId) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).set({
    'phoneNumber': '0987654321',
    'updatedAt': FieldValue.serverTimestamp(),
  }, SetOptions(merge: true));
}

// Cập nhật array (thêm vào array)
Future<void> addPreference(String customerId, String preference) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).update({
    'preferences': FieldValue.arrayUnion([preference]),
  });
}

// Cập nhật số (tăng/giảm)
Future<void> updateLoyaltyPoints(String customerId, int points) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).update({
    'loyaltyPoints': FieldValue.increment(points),
  });
}
```

2.5.4. DELETE - Xóa Document

```
// Xóa document
Future<void> deleteCustomer(String customerId) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).delete();
}
```

```
// Xóa field (set null)
Future<void> removeField(String customerId, String fieldName) async {
  final db = FirebaseFirestore.instance;

  await db.collection('customers').doc(customerId).update({
    fieldName: FieldValue.delete(),
  });
}
```

2.6. Real-time Updates với StreamBuilder

Firestore hỗ trợ **real-time listeners** - tự động cập nhật UI khi dữ liệu thay đổi.

```
// StreamBuilder - Tự động rebuild khi data thay đổi
StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance
      .collection('menu_items')
      .where('isAvailable', isEqualTo: true)
      .snapshots(),
  builder: (context, snapshot) {
    if (snapshot.hasError) {
      return Text('Error: ${snapshot.error}');
    }

    if (snapshot.connectionState == ConnectionState.waiting) {
      return CircularProgressIndicator();
    }

    final items = snapshot.data!.docs;

    return ListView.builder(
      itemCount: items.length,
      itemBuilder: (context, index) {
        final data = items[index].data() as Map<String, dynamic>;
        return ListTile(
          title: Text(data['name'] ?? ''),
          subtitle: Text('${data['price']} VND'),
        );
      },
    );
  },
)

// Listen to một document cụ thể
StreamBuilder<DocumentSnapshot>(
  stream: FirebaseFirestore.instance
      .collection('customers')
      .doc(customerId)
      .snapshots(),
  builder: (context, snapshot) {
```

```
    if (!snapshot.hasData) {
      return CircularProgressIndicator();
    }

    final data = snapshot.data!.data() as Map<String, dynamic>;
    return Text('Loyalty Points: ${data['loyaltyPoints']}');
  },
)
```

2.7. Query và Filtering

Các toán tử so sánh:

```
// where với các toán tử
.where('price', isEqualTo: 80000)           // Bằng
.where('price', isGreaterThan: 50000)       // Lớn hơn
.where('price', isGreaterThanOrEqualTo: 50000) // Lớn hơn hoặc bằng
.where('price', isLessThan: 100000)         // Nhỏ hơn
.where('price', isLessThanOrEqualTo: 100000)   // Nhỏ hơn hoặc bằng
.where('price', isNotEqualTo: 80000)        // Khác

// where với array
.where('preferences', arrayContains: 'vegetarian')  // Array chứa giá trị
.where('preferences', arrayContainsAny: ['vegetarian', 'spicy']) // Chứa bất kỳ

// where với string
.where('name', isGreaterThan: 'A')          // So sánh string
```

Kết hợp nhiều điều kiện:

```
// AND (mặc định)
.where('category', isEqualTo: 'Main Course')
.where('isAvailable', isEqualTo: true)
.where('price', isLessThan: 100000)

// Lưu ý: Firestore chỉ cho phép 1 range query (>, <, >=, <=) mỗi query
// Nếu cần nhiều range, phải dùng composite index
```

Sắp xếp và giới hạn:

```
.orderBy('price', descending: false)  // Tăng dần
.orderBy('price', descending: true)   // Giảm dần
.orderBy('name').orderBy('price')     // Nhiều field (cần index)
.limit(10)                            // Giới hạn số lượng
.startAt([value])                     // Bắt đầu từ giá trị
.endAt([value])                       // Kết thúc tại giá trị
```

Ví dụ Query phức tạp:

```
// Tìm menu items: category = "Main Course", available, giá < 100000, sắp xếp theo giá
Future<List<Map<String, dynamic>>> searchMenuItems({
  String? category,
  bool? vegetarianOnly,
  bool? spicyOnly,
  double? maxPrice,
}) async {
  final db = FirebaseFirestore.instance;
  Query query = db.collection('menu_items');

  if (category != null) {
    query = query.where('category', isEqualTo: category);
  }

  if (vegetarianOnly == true) {
    query = query.where('isVegetarian', isEqualTo: true);
  }

  if (spicyOnly == true) {
    query = query.where('isSpicy', isEqualTo: true);
  }

  if (maxPrice != null) {
    query = query.where('price', isLessThanOrEqualTo: maxPrice);
  }

  query = query.where('isAvailable', isEqualTo: true);
  query = query.orderBy('price');

  final snapshot = await query.get();

  return snapshot.docs.map((doc) => {
    'id': doc.id,
    ...doc.data() as Map<String, dynamic>
  }).toList();
}
```

2.8. Transactions và Batch Writes

Transaction - Đảm bảo tính nhất quán:

```
// Transaction: Đọc và ghi trong một giao dịch atomic
Future<void> payReservation(String reservationId, String customerId) async {
  final db = FirebaseFirestore.instance;
```

```
  await db.runTransaction((transaction) async {
    // Đọc reservation
    final reservationRef = db.collection('reservations').doc(reservationId);
    final reservationDoc = await transaction.get(reservationRef);
    final reservation = reservationDoc.data()!;

    // Đọc customer
    final customerRef = db.collection('customers').doc(customerId);
    final customerDoc = await transaction.get(customerRef);
    final customer = customerDoc.data()!;

    // Tính toán
    final total = reservation['total'] as double;
    final loyaltyPointsToAdd = (total * 0.01).round();
    final newLoyaltyPoints = (customer['loyaltyPoints'] as int) +
loyaltyPointsToAdd;

    // Cập nhật reservation
    transaction.update(reservationRef, {
      'paymentStatus': 'paid',
      'status': 'completed',
      'updatedAt': FieldValue.serverTimestamp(),
    });

    // Cập nhật customer
    transaction.update(customerRef, {
      'loyaltyPoints': newLoyaltyPoints,
    });
  });
}
```

Batch Write - Ghi nhiều operations cùng lúc:

```
// Batch: Thực hiện nhiều writes cùng lúc (all or nothing)
Future<void> createReservationWithItems(
  String customerId,
  Map<String, dynamic> reservationData,
  List<Map<String, dynamic>> items,
) async {
  final db = FirebaseFirestore.instance;
  final batch = db.batch();

  // Tạo reservation
  final reservationRef = db.collection('reservations').doc();
  batch.set(reservationRef, {
    'customerId': customerId,
    'reservationDate': reservationData['reservationDate'],
    'numberOfGuests': reservationData['numberOfGuests'],
    'status': 'pending',
    'createdAt': FieldValue.serverTimestamp(),
  });
```

```
  // Thêm items vào reservation (subcollection)
  for (var item in items) {
    final itemRef = reservationRef.collection('items').doc();
    batch.set(itemRef, item);
  }

  // Commit batch
  await batch.commit();
}
```

2.9. Security Rules

Security Rules kiểm soát quyền truy cập dữ liệu ở phía server.

Cú pháp cơ bản:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Cho phép đọc/ghi tất cả (CHỈ DÙNG CHO TEST)
    match /{document=**} {
      allow read, write: if request.time < timestamp.date(2024, 12, 31);
    }
  }
}
```

Rules cho Restaurant App:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {

    // Customers: Chỉ đọc/ghi chính mình hoặc admin
    match /customers/{customerId} {
      allow read: if request.auth != null;
      allow create: if request.auth != null;
      allow update, delete: if request.auth != null &&
        (request.auth.uid == customerId || isAdmin());
    }

    // Menu Items: Tất cả đọc được, chỉ admin ghi
    match /menu_items/{itemId} {
      allow read: if true;
      allow write: if isAdmin();
    }

    // Reservations: Đọc/ghi chính mình hoặc admin
```

```
    match /reservations/{reservationId} {
      allow read: if request.auth != null &&
        (resource.data.customerId == request.auth.uid || isAdmin());
      allow create: if request.auth != null &&
        request.resource.data.customerId == request.auth.uid;
      allow update: if request.auth != null &&
        (resource.data.customerId == request.auth.uid || isAdmin());
      allow delete: if request.auth != null &&
        (resource.data.customerId == request.auth.uid || isAdmin());
    }

    // Helper function
    function isAdmin() {
      return request.auth != null &&

get(/databases/$(database)/documents/customers/$(request.auth.uid)).data.isAdmin
== true;
    }
  }
}
```

2.10. Model Classes trong Flutter

Tạo Model classes để làm việc với Firestore dễ dàng hơn.

Customer Model:

```
class CustomerModel {
  final String? customerId;
  final String email;
  final String fullName;
  final String? phoneNumber;
  final String? address;
  final List<String> preferences;
  final int loyaltyPoints;
  final DateTime? createdAt;
  final bool isActive;

  CustomerModel({
    this.customerId,
    required this.email,
    required this.fullName,
    this.phoneNumber,
    this.address,
    this.preferences = const [],
    this.loyaltyPoints = 0,
    this.createdAt,
    this.isActive = true,
  });

  // Từ Firestore Document → Model
```

```
  factory CustomerModel.fromFirestore(DocumentSnapshot doc) {
    final data = doc.data() as Map<String, dynamic>;
    return CustomerModel(
      customerId: doc.id,
      email: data['email'] ?? '',
      fullName: data['fullName'] ?? '',
      phoneNumber: data['phoneNumber'],
      address: data['address'],
      preferences: List<String>.from(data['preferences'] ?? []),
      loyaltyPoints: data['loyaltyPoints'] ?? 0,
      createdAt: (data['createdAt'] as Timestamp?)?.toDate(),
      isActive: data['isActive'] ?? true,
    );
  }

  // Từ Model → Firestore Map
  Map<String, dynamic> toFirestore() {
    return {
      'email': email,
      'fullName': fullName,
      'phoneNumber': phoneNumber,
      'address': address,
      'preferences': preferences,
      'loyaltyPoints': loyaltyPoints,
      'createdAt': createdAt != null ? Timestamp.fromDate(createdAt!) :
FieldValue.serverTimestamp(),
      'isActive': isActive,
    };
  }

  // Copy with
  CustomerModel copyWith({
    String? customerId,
    String? email,
    String? fullName,
    String? phoneNumber,
    String? address,
    List<String>? preferences,
    int? loyaltyPoints,
    DateTime? createdAt,
    bool? isActive,
  }) {
    return CustomerModel(
      customerId: customerId ?? this.customerId,
      email: email ?? this.email,
      fullName: fullName ?? this.fullName,
      phoneNumber: phoneNumber ?? this.phoneNumber,
      address: address ?? this.address,
      preferences: preferences ?? this.preferences,
      loyaltyPoints: loyaltyPoints ?? this.loyaltyPoints,
      createdAt: createdAt ?? this.createdAt,
      isActive: isActive ?? this.isActive,
    );
```

```
    }
  }
```

MenuItem Model:

```
class MenuItemModel {
  final String? itemId;
  final String name;
  final String? description;
  final String category;
  final double price;
  final String? imageUrl;
  final List<String> ingredients;
  final bool isVegetarian;
  final bool isSpicy;
  final int preparationTime;
  final bool isAvailable;
  final double rating;
  final DateTime? createdAt;

  MenuItemModel({
    this.itemId,
    required this.name,
    this.description,
    required this.category,
    required this.price,
    this.imageUrl,
    this.ingredients = const [],
    this.isVegetarian = false,
    this.isSpicy = false,
    this.preparationTime = 0,
    this.isAvailable = true,
    this.rating = 0.0,
    this.createdAt,
  });

  factory MenuItemModel.fromFirestore(DocumentSnapshot doc) {
    final data = doc.data() as Map<String, dynamic>;
    return MenuItemModel(
      itemId: doc.id,
      name: data['name'] ?? '',
      description: data['description'],
      category: data['category'] ?? '',
      price: (data['price'] ?? 0.0).toDouble(),
      imageUrl: data['imageUrl'],
      ingredients: List<String>.from(data['ingredients'] ?? []),
      isVegetarian: data['isVegetarian'] ?? false,
      isSpicy: data['isSpicy'] ?? false,
      preparationTime: data['preparationTime'] ?? 0,
      isAvailable: data['isAvailable'] ?? true,
      rating: (data['rating'] ?? 0.0).toDouble(),
```

```
      createdAt: (data['createdAt'] as Timestamp?)?.toDate(),
    );
  }

  Map<String, dynamic> toFirestore() {
    return {
      'name': name,
      'description': description,
      'category': category,
      'price': price,
      'imageUrl': imageUrl,
      'ingredients': ingredients,
      'isVegetarian': isVegetarian,
      'isSpicy': isSpicy,
      'preparationTime': preparationTime,
      'isAvailable': isAvailable,
      'rating': rating,
      'createdAt': createdAt != null ? Timestamp.fromDate(createdAt!) :
FieldValue.serverTimestamp(),
    };
  }
}
```

2.11. Repository Pattern

Repository Pattern giúp tách biệt logic truy cập dữ liệu khỏi UI.

Customer Repository:

```
class CustomerRepository {
  final FirebaseFirestore _db = FirebaseFirestore.instance;

  // Thêm customer
  Future<String> addCustomer(CustomerModel customer) async {
    final docRef = await _db.collection('customers').add(customer.toFirestore());
    return docRef.id;
  }

  // Lấy customer theo ID
  Future<CustomerModel?> getCustomerById(String customerId) async {
    final doc = await _db.collection('customers').doc(customerId).get();
    if (doc.exists) {
      return CustomerModel.fromFirestore(doc);
    }
    return null;
  }

  // Lấy tất cả customers
  Future<List<CustomerModel>> getAllCustomers() async {
    final snapshot = await _db.collection('customers').get();
    return snapshot.docs
```

```
          .map((doc) => CustomerModel.fromFirestore(doc))
          .toList();
    }

    // Cập nhật customer
    Future<void> updateCustomer(String customerId, CustomerModel customer) async {
      await
  _db.collection('customers').doc(customerId).update(customer.toFirestore());
    }

    // Cập nhật loyalty points
    Future<void> updateLoyaltyPoints(String customerId, int points) async {
      await _db.collection('customers').doc(customerId).update({
        'loyaltyPoints': FieldValue.increment(points),
      });
    }

    // Stream customers (real-time)
    Stream<List<CustomerModel>> streamCustomers() {
      return _db.collection('customers')
          .where('isActive', isEqualTo: true)
          .snapshots()
          .map((snapshot) => snapshot.docs
              .map((doc) => CustomerModel.fromFirestore(doc))
              .toList());
    }
  }
```

---

3. XÂY DỰNG RESTFUL API VỚI .NET FRAMEWORK

3.1. RESTful API là gì?

REST (Representational State Transfer) là một kiến trúc cho phép giao tiếp giữa client và server thông qua HTTP.

Nguyên tắc REST:

- ☑ **Stateless**: Mỗi request độc lập, không lưu trạng thái
- ☑ **Resource-based**: Mọi thứ là resource (danh từ)
- ☑ **HTTP Methods**: GET, POST, PUT, DELETE
- ☑ **Uniform Interface**: URL nhất quán

HTTP Methods:

- **GET**: Lấy dữ liệu (Read)
- **POST**: Tạo mới (Create)
- **PUT**: Cập nhật toàn bộ (Update)
- **PATCH**: Cập nhật một phần (Partial Update)
- **DELETE**: Xóa (Delete)

Ví dụ RESTful URLs:

```
GET    /api/customers           → Lấy danh sách customers
GET    /api/customers/1          → Lấy customer có ID = 1
POST   /api/customers            → Tạo customer mới
PUT    /api/customers/1          → Cập nhật customer ID = 1
DELETE /api/customers/1          → Xóa customer ID = 1
```

3.2. .NET Framework vs .NET Core

.NET Framework:

- Chỉ chạy trên Windows
- Phù hợp với ứng dụng Windows desktop
- **Không khuyến nghị** cho API mới

.NET Core / .NET 5+ (hiện tại là .NET 6/7/8):

- Cross-platform (Windows, Linux, macOS)
- Hiệu năng cao hơn
- **Khuyến nghị** cho API mới

Lưu ý: Đề thi yêu cầu .NET Framework, nhưng trong thực tế nên dùng .NET Core/6/7/8. Tài liệu này sẽ hướng dẫn cả hai.

3.3. Thiết lập Project .NET Framework Web API

Bước 1: Tạo Project

1. Mở Visual Studio
2. File → New → Project
3. Chọn **ASP.NET Web Application (.NET Framework)**
4. Chọn template **Web API**
5. Đặt tên: `RestaurantAPI`
6. Click OK

Bước 2: Cấu trúc Project

```
RestaurantAPI/
├── Controllers/          # API Controllers
│   ├── CustomersController.cs
│   ├── MenuItemsController.cs
│   └── ReservationsController.cs
├── Models/              # Data Models
│   ├── Customer.cs
│   ├── MenuItem.cs
│   └── Reservation.cs
```

```
├── Data/                # Database Context
│   └── ApplicationDbContext.cs
├── Services/            # Business Logic
│   └── CustomerService.cs
├── App_Start/           # Configuration
│   └── WebApiConfig.cs
└── Web.config          # Configuration file
```

Bước 3: Cài đặt NuGet Packages

Mở Package Manager Console và chạy:

```
# Entity Framework (ORM)
Install-Package EntityFramework

# JWT Authentication
Install-Package Microsoft.AspNet.WebApi.Owin
Install-Package Microsoft.Owin.Security.Jwt

# CORS
Install-Package Microsoft.AspNet.WebApi.Cors

# Swagger (API Documentation)
Install-Package Swashbuckle
```

3.4. Database Setup với Entity Framework

Bước 1: Tạo Connection String

Trong `Web.config`:

```
<connectionStrings>
  <add name="DefaultConnection"
       connectionString="Server=localhost;Database=db_exam_2151061234;User
Id=sa;Password=YourPassword;Trusted_Connection=False;MultipleActiveResultSets=True
;"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Bước 2: Tạo Models

Customer.cs:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
```

```
namespace RestaurantAPI.Models
{
    public class Customer
    {
        [Key]
        public int Id { get; set; }

        [Required]
        [EmailAddress]
        [StringLength(255)]
        public string Email { get; set; }

        [Required]
        [StringLength(255)]
        public string Password { get; set; }

        [Required]
        [StringLength(255)]
        public string FullName { get; set; }

        [StringLength(20)]
        public string? PhoneNumber { get; set; }

        [StringLength(500)]
        public string? Address { get; set; }

        public int LoyaltyPoints { get; set; } = 0;

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

        public bool IsActive { get; set; } = true;

        // Navigation properties
        public virtual ICollection<Reservation> Reservations { get; set; }
    }
}
```

MenuItem.cs:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RestaurantAPI.Models
{
    public class MenuItem
    {
        [Key]
        public int Id { get; set; }
```

```
        [Required]
        [StringLength(255)]
        public string Name { get; set; }

        public string? Description { get; set; }

        [Required]
        [StringLength(50)]
        public string Category { get; set; } // "Appetizer", "Main Course", etc.

        [Required]
        [Column(TypeName = "decimal(18,2)")]
        public decimal Price { get; set; }

        [StringLength(500)]
        public string? ImageUrl { get; set; }

        public int PreparationTime { get; set; } // minutes

        public bool IsVegetarian { get; set; } = false;

        public bool IsSpicy { get; set; } = false;

        public bool IsAvailable { get; set; } = true;

        [Column(TypeName = "decimal(3,2)")]
        public decimal Rating { get; set; } = 0.0m;

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

        // Navigation properties
        public virtual ICollection<ReservationItem> ReservationItems { get; set; }
    }
}
```

Reservation.cs:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RestaurantAPI.Models
{
    public class Reservation
    {
        [Key]
        public int Id { get; set; }

        [Required]
```

```
        public int CustomerId { get; set; }

        [Required]
        [StringLength(50)]
        public string ReservationNumber { get; set; }

        [Required]
        public DateTime ReservationDate { get; set; }

        [Required]
        public int NumberOfGuests { get; set; }

        [StringLength(20)]
        public string? TableNumber { get; set; }

        [Required]
        [StringLength(20)]
        public string Status { get; set; } = "pending"; // pending, confirmed,
seated, completed, cancelled, no_show

        public string? SpecialRequests { get; set; }

        [Column(TypeName = "decimal(18,2)")]
        public decimal Subtotal { get; set; } = 0;

        [Column(TypeName = "decimal(18,2)")]
        public decimal ServiceCharge { get; set; } = 0;

        [Column(TypeName = "decimal(18,2)")]
        public decimal Discount { get; set; } = 0;

        [Column(TypeName = "decimal(18,2)")]
        public decimal Total { get; set; } = 0;

        [StringLength(20)]
        public string? PaymentMethod { get; set; } // cash, card, online

        [Required]
        [StringLength(20)]
        public string PaymentStatus { get; set; } = "pending"; // pending, paid,
refunded

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

        // Navigation properties
        [ForeignKey("CustomerId")]
        public virtual Customer Customer { get; set; }

        public virtual ICollection<ReservationItem> ReservationItems { get; set; }
    }
}
```

ReservationItem.cs:

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RestaurantAPI.Models
{
    public class ReservationItem
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public int ReservationId { get; set; }

        [Required]
        public int MenuItemId { get; set; }

        [Required]
        public int Quantity { get; set; }

        [Required]
        [Column(TypeName = "decimal(18,2)")]
        public decimal Price { get; set; } // Giá tại thời điểm đặt

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        // Navigation properties
        [ForeignKey("ReservationId")]
        public virtual Reservation Reservation { get; set; }

        [ForeignKey("MenuItemId")]
        public virtual MenuItem MenuItem { get; set; }
    }
}
```

Table.cs:

```
using System;
using System.ComponentModel.DataAnnotations;

namespace RestaurantAPI.Models
{
    public class Table
    {
        [Key]
        public int Id { get; set; }

        [Required]
```

```
        [StringLength(20)]
        public string TableNumber { get; set; }

        [Required]
        public int Capacity { get; set; }

        public bool IsAvailable { get; set; } = true;

        public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
    }
}
```

Bước 3: Tạo DbContext

ApplicationDbContext.cs:

```
using System.Data.Entity;
using RestaurantAPI.Models;

namespace RestaurantAPI.Data
{
    public class ApplicationDbContext : DbContext
    {
        public ApplicationDbContext() : base("DefaultConnection")
        {
        }

        public DbSet<Customer> Customers { get; set; }
        public DbSet<MenuItem> MenuItems { get; set; }
        public DbSet<Reservation> Reservations { get; set; }
        public DbSet<ReservationItem> ReservationItems { get; set; }
        public DbSet<Table> Tables { get; set; }

        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            // Configure relationships
            modelBuilder.Entity<Reservation>()
                .HasRequired(r => r.Customer)
                .WithMany(c => c.Reservations)
                .HasForeignKey(r => r.CustomerId)
                .WillCascadeOnDelete(false); // ON DELETE RESTRICT

            modelBuilder.Entity<ReservationItem>()
                .HasRequired(ri => ri.Reservation)
                .WithMany(r => r.ReservationItems)
                .HasForeignKey(ri => ri.ReservationId)
                .WillCascadeOnDelete(true); // ON DELETE CASCADE
```

```
            modelBuilder.Entity<ReservationItem>()
                .HasRequired(ri => ri.MenuItem)
                .WithMany(mi => mi.ReservationItems)
                .HasForeignKey(ri => ri.MenuItemId)
                .WillCascadeOnDelete(false); // ON DELETE RESTRICT

            // Unique constraints
            modelBuilder.Entity<Customer>()
                .HasIndex(c => c.Email)
                .IsUnique();

            modelBuilder.Entity<Reservation>()
                .HasIndex(r => r.ReservationNumber)
                .IsUnique();

            modelBuilder.Entity<Table>()
                .HasIndex(t => t.TableNumber)
                .IsUnique();
        }
    }
}
```

Bước 4: Enable Migrations

Trong Package Manager Console:

```
Enable-Migrations
Add-Migration InitialCreate
Update-Database
```

3.5. Tạo Controllers

CustomersController.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using RestaurantAPI.Data;
using RestaurantAPI.Models;
using System.Data.Entity;

namespace RestaurantAPI.Controllers
{
    [RoutePrefix("api/customers")]
```

```
public class CustomersController : ApiController
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: api/customers
    [HttpGet]
    [Route("")]
    public IHttpActionResult GetCustomers()
    {
        var customers = db.Customers
            .Where(c => c.IsActive)
            .Select(c => new
            {
                c.Id,
                c.Email,
                c.FullName,
                c.PhoneNumber,
                c.Address,
                c.LoyaltyPoints,
                c.CreatedAt
            })
            .ToList();

        return Ok(new { success = true, data = customers });
    }

    // GET: api/customers/5
    [HttpGet]
    [Route("{id}")]
    public IHttpActionResult GetCustomer(int id)
    {
        var customer = db.Customers.Find(id);

        if (customer == null || !customer.IsActive)
        {
            return NotFound();
        }

        var result = new
        {
            customer.Id,
            customer.Email,
            customer.FullName,
            customer.PhoneNumber,
            customer.Address,
            customer.LoyaltyPoints,
            customer.CreatedAt
        };

        return Ok(new { success = true, data = result });
    }

    // POST: api/customers
    [HttpPost]
```

```
        [Route("")]
        public IHttpActionResult CreateCustomer([FromBody] Customer customer)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            customer.CreatedAt = DateTime.UtcNow;
            customer.UpdatedAt = DateTime.UtcNow;
            customer.LoyaltyPoints = 0;
            customer.IsActive = true;

            db.Customers.Add(customer);
            db.SaveChanges();

            return CreatedAtRoute("DefaultApi", new { id = customer.Id },
                new { success = true, data = customer });
        }

        // PUT: api/customers/5
        [HttpPut]
        [Route("{id}")]
        public IHttpActionResult UpdateCustomer(int id, [FromBody] Customer
customerData)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var customer = db.Customers.Find(id);

            if (customer == null)
            {
                return NotFound();
            }

            customer.FullName = customerData.FullName;
            customer.PhoneNumber = customerData.PhoneNumber;
            customer.Address = customerData.Address;
            customer.UpdatedAt = DateTime.UtcNow;

            db.SaveChanges();

            return Ok(new { success = true, data = customer });
        }

        // DELETE: api/customers/5
        [HttpDelete]
        [Route("{id}")]
        public IHttpActionResult DeleteCustomer(int id)
        {
            var customer = db.Customers.Find(id);
```

```
            if (customer == null)
            {
                return NotFound();
            }

            // Soft delete
            customer.IsActive = false;
            customer.UpdatedAt = DateTime.UtcNow;
            db.SaveChanges();

            return Ok(new { success = true, message = "Customer deleted
successfully" });
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

MenuItemsController.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using RestaurantAPI.Data;
using RestaurantAPI.Models;
using System.Data.Entity;

namespace RestaurantAPI.Controllers
{
    [RoutePrefix("api/menu-items")]
    public class MenuItemsController : ApiController
    {
        private ApplicationDbContext db = new ApplicationDbContext();

        // GET: api/menu-items
        [HttpGet]
        [Route("")]
        public IHttpActionResult GetMenuItems(
            int page = 1,
            int limit = 10,
            string search = null,
            string category = null,
```

```
            bool? vegetarianOnly = null,
            bool? spicyOnly = null,
            bool? availableOnly = null
        )
        {
            IQueryable<MenuItem> query = db.MenuItems;

            // Search
            if (!string.IsNullOrEmpty(search))
            {
                query = query.Where(m =>
                    m.Name.Contains(search) ||
                    (m.Description != null && m.Description.Contains(search))
                );
            }

            // Filter by category
            if (!string.IsNullOrEmpty(category))
            {
                query = query.Where(m => m.Category == category);
            }

            // Filter by vegetarian
            if (vegetarianOnly.HasValue && vegetarianOnly.Value)
            {
                query = query.Where(m => m.IsVegetarian == true);
            }

            // Filter by spicy
            if (spicyOnly.HasValue && spicyOnly.Value)
            {
                query = query.Where(m => m.IsSpicy == true);
            }

            // Filter by available
            if (availableOnly.HasValue && availableOnly.Value)
            {
                query = query.Where(m => m.IsAvailable == true);
            }

            // Pagination
            var total = query.Count();
            var items = query
                .OrderBy(m => m.Name)
                .Skip((page - 1) * limit)
                .Take(limit)
                .Select(m => new
                {
                    m.Id,
                    m.Name,
                    m.Description,
                    m.Category,
                    m.Price,
                    m.ImageUrl,
```

```
                    m.PreparationTime,
                    m.IsVegetarian,
                    m.IsSpicy,
                    m.IsAvailable,
                    m.Rating
                })
                .ToList();

            return Ok(new
            {
                success = true,
                data = items,
                pagination = new
                {
                    page,
                    limit,
                    total,
                    totalPages = (int)Math.Ceiling((double)total / limit)
                }
            });
        }

        // GET: api/menu-items/5
        [HttpGet]
        [Route("{id}")]
        public IHttpActionResult GetMenuItem(int id)
        {
            var item = db.MenuItems.Find(id);

            if (item == null)
            {
                return NotFound();
            }

            var result = new
            {
                item.Id,
                item.Name,
                item.Description,
                item.Category,
                item.Price,
                item.ImageUrl,
                item.PreparationTime,
                item.IsVegetarian,
                item.IsSpicy,
                item.IsAvailable,
                item.Rating,
                item.CreatedAt
            };

            return Ok(new { success = true, data = result });
        }

        // POST: api/menu-items
```

```
        [HttpPost]
        [Route("")]
        public IHttpActionResult CreateMenuItem([FromBody] MenuItem menuItem)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            menuItem.CreatedAt = DateTime.UtcNow;
            menuItem.UpdatedAt = DateTime.UtcNow;

            db.MenuItems.Add(menuItem);
            db.SaveChanges();

            return CreatedAtRoute("DefaultApi", new { id = menuItem.Id },
                new { success = true, data = menuItem });
        }

        // PUT: api/menu-items/5
        [HttpPut]
        [Route("{id}")]
        public IHttpActionResult UpdateMenuItem(int id, [FromBody] MenuItem
menuItemData)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var menuItem = db.MenuItems.Find(id);

            if (menuItem == null)
            {
                return NotFound();
            }

            menuItem.Name = menuItemData.Name;
            menuItem.Description = menuItemData.Description;
            menuItem.Category = menuItemData.Category;
            menuItem.Price = menuItemData.Price;
            menuItem.ImageUrl = menuItemData.ImageUrl;
            menuItem.PreparationTime = menuItemData.PreparationTime;
            menuItem.IsVegetarian = menuItemData.IsVegetarian;
            menuItem.IsSpicy = menuItemData.IsSpicy;
            menuItem.IsAvailable = menuItemData.IsAvailable;
            menuItem.Rating = menuItemData.Rating;
            menuItem.UpdatedAt = DateTime.UtcNow;

            db.SaveChanges();

            return Ok(new { success = true, data = menuItem });
        }
```

```
        // DELETE: api/menu-items/5
        [HttpDelete]
        [Route("{id}")]
        public IHttpActionResult DeleteMenuItem(int id)
        {
            var menuItem = db.MenuItems.Find(id);

            if (menuItem == null)
            {
                return NotFound();
            }

            // Kiểm tra có trong reservation_items không (reservation chưa
completed)
            var hasActiveReservations = db.ReservationItems
                .Any(ri => ri.MenuItemId == id &&
                    ri.Reservation.Status != "completed" &&
                    ri.Reservation.Status != "cancelled");

            if (hasActiveReservations)
            {
                return BadRequest("Cannot delete menu item that is in active
reservations");
            }

            db.MenuItems.Remove(menuItem);
            db.SaveChanges();

            return Ok(new { success = true, message = "Menu item deleted
successfully" });
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                db.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

3.6. Authentication & Authorization

Bước 1: Tạo DTOs (Data Transfer Objects)

LoginDto.cs:

```
namespace RestaurantAPI.Models
{
```

```
    public class LoginDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }

    public class RegisterDto
    {
        public string Email { get; set; }
        public string Password { get; set; }
        public string FullName { get; set; }
        public string PhoneNumber { get; set; }
        public string Address { get; set; }
    }

    public class LoginResponseDto
    {
        public bool Success { get; set; }
        public string Token { get; set; }
        public object User { get; set; }
        public string StudentId { get; set; } // BẮT BUỘC theo đề thi
    }
}
```

Bước 2: Tạo AuthController

AuthController.cs:

```
using System;
using System.Linq;
using System.Web.Http;
using RestaurantAPI.Data;
using RestaurantAPI.Models;
using System.Security.Cryptography;
using System.Text;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.IdentityModel.Tokens;
using System.Security.Claims;

namespace RestaurantAPI.Controllers
{
    [RoutePrefix("api/auth")]
    public class AuthController : ApiController
    {
        private ApplicationDbContext db = new ApplicationDbContext();
        private const string SECRET_KEY = "your-secret-key-minimum-32-characters-
long"; // Nên lưu trong config

        // POST: api/auth/register
        [HttpPost]
        [Route("register")]
```

```
        public IHttpActionResult Register([FromBody] RegisterDto registerDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            // Kiểm tra email đã tồn tại
            if (db.Customers.Any(c => c.Email == registerDto.Email))
            {
                return BadRequest("Email already exists");
            }

            // Hash password (nên dùng BCrypt hoặc ASP.NET Identity)
            var hashedPassword = HashPassword(registerDto.Password);

            var customer = new Customer
            {
                Email = registerDto.Email,
                Password = hashedPassword,
                FullName = registerDto.FullName,
                PhoneNumber = registerDto.PhoneNumber,
                Address = registerDto.Address,
                LoyaltyPoints = 0,
                IsActive = true,
                CreatedAt = DateTime.UtcNow,
                UpdatedAt = DateTime.UtcNow
            };

            db.Customers.Add(customer);
            db.SaveChanges();

            return Ok(new
            {
                success = true,
                message = "Registration successful",
                data = new
                {
                    customer.Id,
                    customer.Email,
                    customer.FullName
                }
            });
        }

        // POST: api/auth/login
        [HttpPost]
        [Route("login")]
        public IHttpActionResult Login([FromBody] LoginDto loginDto)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
```

```
            var hashedPassword = HashPassword(loginDto.Password);
            var customer = db.Customers
                .FirstOrDefault(c => c.Email == loginDto.Email &&
                                     c.Password == hashedPassword &&
                                     c.IsActive);

            if (customer == null)
            {
                return Unauthorized();
            }

            // Tạo JWT Token
            var token = GenerateJwtToken(customer);

            // Response theo yêu cầu đề thi (BẮT BUỘC có student_id)
            return Ok(new LoginResponseDto
            {
                Success = true,
                Token = token,
                User = new
                {
                    customer.Id,
                    customer.Email,
                    customer.FullName,
                    customer.LoyaltyPoints
                },
                StudentId = "2151061234" // HARDCODE theo Mã Sinh Viên của bạn
            });
        }

        // GET: api/auth/me
        [HttpGet]
        [Route("me")]
        [Authorize] // Cần authentication
        public IHttpActionResult GetCurrentUser()
        {
            var customerId = int.Parse(User.Identity.Name);
            var customer = db.Customers.Find(customerId);

            if (customer == null)
            {
                return NotFound();
            }

            return Ok(new
            {
                success = true,
                data = new
                {
                    customer.Id,
                    customer.Email,
                    customer.FullName,
                    customer.PhoneNumber,
```

```
                    customer.Address,
                    customer.LoyaltyPoints,
                    customer.CreatedAt
                }
            });
        }

        // Helper methods
        private string HashPassword(string password)
        {
            // Đơn giản: SHA256 (NÊN DÙNG BCrypt trong production)
            using (var sha256 = SHA256.Create())
            {
                var hashedBytes =
sha256.ComputeHash(Encoding.UTF8.GetBytes(password));
                return Convert.ToBase64String(hashedBytes);
            }
        }

        private string GenerateJwtToken(Customer customer)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(SECRET_KEY);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new[]
                {
                    new Claim(ClaimTypes.Name, customer.Id.ToString()),
                    new Claim(ClaimTypes.Email, customer.Email),
                    new Claim("FullName", customer.FullName)
                }),
                Expires = DateTime.UtcNow.AddDays(7),
                SigningCredentials = new SigningCredentials(
                    new SymmetricSecurityKey(key),
                    SecurityAlgorithms.HmacSha256Signature
                )
            };

            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);
        }
    }
}
```

Bước 3: Tạo JWT Authentication Middleware

JwtAuthenticationAttribute.cs:

```
using System;
using System.IdentityModel.Tokens.Jwt;
```

```
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Security.Claims;
using System.Threading;
using System.Threading.Tasks;
using System.Web.Http;
using System.Web.Http.Filters;
using Microsoft.IdentityModel.Tokens;

namespace RestaurantAPI.Filters
{
    public class JwtAuthenticationAttribute : Attribute, IAuthenticationFilter
    {
        private const string SECRET_KEY = "your-secret-key-minimum-32-characters-
long";

        public bool AllowMultiple => false;

        public async Task AuthenticateAsync(HttpAuthenticationContext context,
CancellationToken cancellationToken)
        {
            var request = context.Request;
            var authorization = request.Headers.Authorization;

            if (authorization == null || authorization.Scheme != "Bearer")
            {
                return; // Không có token, sẽ trả về 401
            }

            if (string.IsNullOrEmpty(authorization.Parameter))
            {
                context.ErrorResult = new AuthenticationFailureResult("Missing
token", request);
                return;
            }

            var token = authorization.Parameter;
            var principal = await ValidateToken(token);

            if (principal == null)
            {
                context.ErrorResult = new AuthenticationFailureResult("Invalid
token", request);
                return;
            }

            context.Principal = principal;
        }

        public async Task ChallengeAsync(HttpAuthenticationChallengeContext
context, CancellationToken cancellationToken)
        {
            var challenge = new
```

```
System.Net.Http.Headers.AuthenticationHeaderValue("Bearer");
            context.Result = new AddChallengeOnUnauthorizedResult(challenge,
context.Result);
        }

        private Task<ClaimsPrincipal> ValidateToken(string token)
        {
            try
            {
                var tokenHandler = new JwtSecurityTokenHandler();
                var key = System.Text.Encoding.ASCII.GetBytes(SECRET_KEY);

                var validationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new SymmetricSecurityKey(key),
                    ValidateIssuer = false,
                    ValidateAudience = false,
                    ClockSkew = TimeSpan.Zero
                };

                var principal = tokenHandler.ValidateToken(token,
validationParameters, out SecurityToken validatedToken);
                return Task.FromResult(principal);
            }
            catch
            {
                return Task.FromResult<ClaimsPrincipal>(null);
            }
        }
    }

    public class AuthenticationFailureResult : IHttpActionResult
    {
        public string ReasonPhrase { get; }
        public HttpRequestMessage Request { get; }

        public AuthenticationFailureResult(string reasonPhrase, HttpRequestMessage
request)
        {
            ReasonPhrase = reasonPhrase;
            Request = request;
        }

        public Task<HttpResponseMessage> ExecuteAsync(CancellationToken
cancellationToken)
        {
            return Task.FromResult(Execute());
        }

        private HttpResponseMessage Execute()
        {
            var response = new HttpResponseMessage(HttpStatusCode.Unauthorized)
            {
```

```
                RequestMessage = Request,
                ReasonPhrase = ReasonPhrase
            };

            return response;
        }
    }

    public class AddChallengeOnUnauthorizedResult : IHttpActionResult
    {
        private readonly System.Net.Http.Headers.AuthenticationHeaderValue
_challenge;
        private readonly IHttpActionResult _innerResult;

        public
AddChallengeOnUnauthorizedResult(System.Net.Http.Headers.AuthenticationHeaderValue
challenge, IHttpActionResult innerResult)
        {
            _challenge = challenge;
            _innerResult = innerResult;
        }

        public async Task<HttpResponseMessage> ExecuteAsync(CancellationToken
cancellationToken)
        {
            var response = await _innerResult.ExecuteAsync(cancellationToken);

            if (response.StatusCode == HttpStatusCode.Unauthorized)
            {
                response.Headers.WwwAuthenticate.Add(_challenge);
            }

            return response;
        }
    }
}
```

Bước 4: Áp dụng Authentication

Thêm `[JwtAuthentication]` hoặc `[Authorize]` vào các controller/action cần authentication:

```
[RoutePrefix("api/reservations")]
[JwtAuthentication] // Áp dụng cho toàn bộ controller
public class ReservationsController : ApiController
{
    // Tất cả actions đều cần authentication

    [HttpPost]
    [Route("")]
    public IHttpActionResult CreateReservation([FromBody] ReservationDto dto)
    {
```

```
        // ...
    }
}
```

3.7. CORS Configuration

Cho phép Flutter app gọi API từ domain khác.

WebApiConfig.cs:

```
using System.Web.Http;

namespace RestaurantAPI
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // CORS
            config.EnableCors();

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Web.config:

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="Access-Control-Allow-Origin" value="*" />
      <add name="Access-Control-Allow-Headers" value="Content-Type, Authorization"
/>
      <add name="Access-Control-Allow-Methods" value="GET, POST, PUT, DELETE,
OPTIONS" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

3.8. Error Handling

Global Exception Handler:

GlobalExceptionHandler.cs:

```
using System;
using System.Net;
using System.Net.Http;
using System.Web.Http.Filters;

namespace RestaurantAPI.Filters
{
    public class GlobalExceptionFilter : ExceptionFilterAttribute
    {
        public override void OnException(HttpActionExecutedContext context)
        {
            var exception = context.Exception;

            var response = new
HttpResponseMessage(HttpStatusCode.InternalServerError)
            {
                Content = new StringContent(new
                {
                    success = false,
                    message = "An error occurred while processing your request",
                    error = exception.Message
                }.ToString()),
                ReasonPhrase = "Internal Server Error"
            };

            context.Response = response;
        }
    }
}
```

Áp dụng trong `WebApiConfig.cs`:

```
config.Filters.Add(new GlobalExceptionFilter());
```

3.9. Response Format nhất quán

Tạo BaseController:

```
using System.Web.Http;

namespace RestaurantAPI.Controllers
{
    public class BaseApiController : ApiController
    {
```

```
        protected IHttpActionResult Success(object data = null, string message =
null)
        {
            return Ok(new
            {
                success = true,
                message = message,
                data = data
            });
        }

        protected IHttpActionResult Error(string message, int statusCode = 400)
        {
            return Content((System.Net.HttpStatusCode)statusCode, new
            {
                success = false,
                message = message
            });
        }
    }
}
```

Sử dụng:

```
public class CustomersController : BaseApiController
{
    [HttpGet]
    public IHttpActionResult GetCustomers()
    {
        var customers = db.Customers.ToList();
        return Success(customers);
    }
}
```

---

4. SO SÁNH FIRESTORE VS SQL DATABASE

4.1. Bảng so sánh

Tiêu chí	Firestore	SQL Database
Loại	NoSQL Document	Relational
Cấu trúc	Collections/Documents	Tables/Rows
Schema	Flexible (không cố định)	Fixed (cần định nghĩa trước)
Relationships	Không có JOIN, tự xử lý	Foreign Keys, JOIN
Query	Hạn chế (1 range query)	Mạnh mẽ (nhiều điều kiện)

Tiêu chí	Firestore	SQL Database
Real-time	Có sẵn	Phải tự implement
Offline	Hỗ trợ sẵn	Không
Scalability	Tự động scale	Cần cấu hình
Chi phí	Pay per use	Fixed/VPS
Phù hợp	Mobile apps, Real-time	Web apps, Complex queries

4.2. Khi nào dùng Firestore?

☑ **Nên dùng Firestore khi:**

- Ứng dụng mobile (Flutter, React Native)
- Cần real-time updates
- Schema thay đổi thường xuyên
- Dữ liệu không phức tạp
- Cần offline support

4.3. Khi nào dùng SQL Database?

☑ **Nên dùng SQL khi:**

- Web API backend
- Queries phức tạp (JOIN, GROUP BY, v.v.)
- Dữ liệu có quan hệ chặt chẽ
- Cần ACID transactions
- Có team backend chuyên nghiệp

---

5. BEST PRACTICES VÀ TIPS

5.1. Firestore Best Practices

1. Tổ chức dữ liệu tốt:

```
// ☑ TỐT: Flatten structure
customers/{customerId}
reservations/{reservationId} với customerId field

// ❌ XẤU: Nested quá sâu
customers/{customerId}/reservations/{reservationId}/items/{itemId}
```

2. Sử dụng Composite Index:

Khi query nhiều điều kiện, tạo index trong Firebase Console.

3. Pagination:

```
// Sử dụng startAfter cho pagination
final lastDoc = snapshot.docs.last;
final nextPage = await db.collection('menu_items')
    .startAfterDocument(lastDoc)
    .limit(10)
    .get();
```

4. Batch Operations:

Dùng batch/transaction cho nhiều writes liên quan.

5. Security Rules:

Luôn viết Security Rules, không để test mode trong production.

5.2. RESTful API Best Practices

1. RESTful URLs:

```
☑ GET    /api/customers
☑ GET    /api/customers/1
☑ POST   /api/customers
☑ PUT    /api/customers/1
☑ DELETE /api/customers/1

❌ GET /api/getCustomers
❌ POST /api/createCustomer
```

2. HTTP Status Codes:

- `200 OK`: Thành công
- `201 Created`: Tạo mới thành công
- `400 Bad Request`: Request không hợp lệ
- `401 Unauthorized`: Chưa đăng nhập
- `403 Forbidden`: Không có quyền
- `404 Not Found`: Không tìm thấy
- `500 Internal Server Error`: Lỗi server

3. Response Format:

```
{
  "success": true,
  "data": { ... },
```

```
  "message": "Optional message"
}
```

4. Error Handling:

```
{
  "success": false,
  "message": "Error message",
  "errors": {
    "email": ["Email is required"]
  }
}
```

5. Pagination:

```
{
  "success": true,
  "data": [ ... ],
  "pagination": {
    "page": 1,
    "limit": 10,
    "total": 100,
    "totalPages": 10
  }
}
```

5.3. Tips cho Đề Thi

Firebase 05:

1. ☑ Tạo Firebase project với tên chứa Mã Sinh Viên
2. ☑ Package name: `com.example.<MaSinhVien>`
3. ☑ AppBar hiển thị Mã Sinh Viên
4. ☑ Model classes với fromFirestore/toFirestore
5. ☑ Repository pattern
6. ☑ StreamBuilder cho real-time
7. ☑ Error handling đầy đủ

Web API 05:

1. ☑ Folder: `web_api_<MaSinhVien>`
2. ☑ Database: `db_exam_<MaSinhVien>`
3. ☑ Login response có `student_id` (hardcode)
4. ☑ Response format nhất quán
5. ☑ Authentication với JWT

6. ☑ CORS đã cấu hình
7. ☑ Validation đầy đủ
8. ☑ Error handling toàn cục

📝 TÓM TẮT

Firebase Firestore:

- NoSQL document database
- Real-time, offline support
- Phù hợp mobile apps
- Flexible schema

RESTful API với .NET:

- Standard HTTP methods
- JWT authentication
- Entity Framework ORM
- Response format nhất quán

Lưu ý:

- Firestore: Dùng cho Flutter app (đề Firebase 05)
- SQL + .NET API: Dùng cho backend (đề Web API 05)
- Cả hai đều cần authentication, validation, error handling

Chúc các bạn học tập tốt và làm bài thi thành công! 🚀