Smart Task Manager - Dart Code Sample

Project Structure

```
lib/
├── models/
│   └── task.dart
├── services/
│   └── api_service.dart
├── repositories/
│   └── task_repository.dart
├── controllers/
│   └── task_controller.dart
├── exceptions/
│   └── custom_exceptions.dart
└── main.dart
```

models/task.dart

```
class Task {
  final String id;
  final String title;
  final String description;
  final DateTime createdAt;
  bool isCompleted;

  Task({
    required this.id,
    required this.title,
    required this.description,
    required this.createdAt,
    this.isCompleted = false,
  });

  factory Task.fromJson(Map<String, dynamic> json) => Task(
        id: json['id'],
        title: json['title'],
        description: json['description'],
        createdAt: DateTime.parse(json['createdAt']),
        isCompleted: json['isCompleted'],
```

```
    );

  Map<String, dynamic> toJson() => {
        'id': id,
        'title': title,
        'description': description,
        'createdAt': createdAt.toIso8601String(),
        'isCompleted': isCompleted,
      };

  @override
  String toString() => '$title (Completed: $isCompleted)';
}
```

exceptions/custom_exceptions.dart

```
class NetworkException implements Exception {
  final String message;
  NetworkException(this.message);

  @override
  String toString() => 'NetworkException: $message';
}

class TaskNotFoundException implements Exception {
  final String taskId;
  TaskNotFoundException(this.taskId);

  @override
  String toString() => 'TaskNotFoundException: No task found with ID $taskId';
}
```

services/api_service.dart

```
import 'dart:async';
import 'dart:math';
import '../models/task.dart';
import '../exceptions/custom_exceptions.dart';

class ApiService {
  final Random _rng = Random();
```

```
  final Map<String, Task> _mockDB = {};

  Future<List<Task>> fetchAllTasks() async {
    await _simulateLatency();
    return _mockDB.values.toList();
  }

  Future<Task> fetchTask(String id) async {
    await _simulateLatency();
    if (!_mockDB.containsKey(id)) throw TaskNotFoundException(id);
    return _mockDB[id]!;
  }

  Future<void> createTask(Task task) async {
    await _simulateLatency();
    _mockDB[task.id] = task;
  }

  Future<void> updateTask(Task task) async {
    await _simulateLatency();
    if (!_mockDB.containsKey(task.id)) throw TaskNotFoundException(task.id);
    _mockDB[task.id] = task;
  }

  Future<void> deleteTask(String id) async {
    await _simulateLatency();
    if (!_mockDB.containsKey(id)) throw TaskNotFoundException(id);
    _mockDB.remove(id);
  }

  Future<void> _simulateLatency() async {
    final int delay = _rng.nextInt(800) + 200;
    await Future.delayed(Duration(milliseconds: delay));
    if (_rng.nextDouble() < 0.05) throw NetworkException("Simulated network failure.");
  }
}
```

repositories/task_repository.dart

```
import '../models/task.dart';
import '../services/api_service.dart';
```

```dart
import '../exceptions/custom_exceptions.dart';

class TaskRepository {
  final ApiService _apiService;

  final Map<String, Task> _cache = {};

  TaskRepository(this._apiService);

  Future<List<Task>> getAllTasks({bool useCache = true}) async {
    if (useCache && _cache.isNotEmpty) {
      return _cache.values.toList();
    }

    final tasks = await _apiService.fetchAllTasks();
    for (final task in tasks) {
      _cache[task.id] = task;
    }
    return tasks;
  }

  Future<Task> getTaskById(String id) async {
    if (_cache.containsKey(id)) return _cache[id]!;

    final task = await _apiService.fetchTask(id);
    _cache[id] = task;
    return task;
  }

  Future<void> addTask(Task task) async {
    await _apiService.createTask(task);
    _cache[task.id] = task;
  }

  Future<void> completeTask(String id) async {
    final task = await getTaskById(id);
    task.isCompleted = true;
    await _apiService.updateTask(task);
    _cache[id] = task;
  }
```

```
  Future<void> deleteTask(String id) async {
    await _apiService.deleteTask(id);
    _cache.remove(id);
  }

  void clearCache() {
    _cache.clear();
  }
}
```

controllers/task_controller.dart

```
import '../models/task.dart';
import '../repositories/task_repository.dart';

class TaskController {
  final TaskRepository _repository;

  TaskController(this._repository);

  Future<void> createSampleTask(String title, String description) async {
    final task = Task(
      id: DateTime.now().millisecondsSinceEpoch.toString(),
      title: title,
      description: description,
      createdAt: DateTime.now(),
    );
    await _repository.addTask(task);
  }

  Future<void> markTaskComplete(String id) async {
    await _repository.completeTask(id);
  }

  Future<void> deleteTask(String id) async {
    await _repository.deleteTask(id);
  }

  Future<List<Task>> listTasks() async {
    return await _repository.getAllTasks();
```

```
  }
}
```

main.dart

```
import 'controllers/task_controller.dart';
import 'repositories/task_repository.dart';
import 'services/api_service.dart';
import 'exceptions/custom_exceptions.dart';

void main() async {
  final apiService = ApiService();
  final repo = TaskRepository(apiService);
  final controller = TaskController(repo);

  try {
    print('\n== Creating Tasks ==');
    await controller.createSampleTask("Write Dart sample", "Impress tech giant HRs.");
    await controller.createSampleTask("Review pull requests", "Prioritize Dart PRs.");
    await controller.createSampleTask("Prepare for Google interview", "Study system
design.");

    print('\n== Task List ==');
    final tasks = await controller.listTasks();
    for (var task in tasks) {
      print(task);
    }

    print('\n== Marking First Task Complete ==');
    await controller.markTaskComplete(tasks.first.id);

    print('\n== Final Task List ==');
    final updatedTasks = await controller.listTasks();
    for (var task in updatedTasks) {
      print(task);
    }

  } on NetworkException catch (e) {
    print('Network error occurred: \$e');
  } on Exception catch (e) {
    print('General error: \$e');
```

```
  }
}
```