Marketplace Builder Hackathon 2025 - Day 3: API Integration and Data Migration Documentation

# Table of Contents

1. Overview  
2. Objectives  
3. Prerequisites  
4. Environment Setup  
5. API Integration  
 - Understanding the API  
 - Authentication & Authorization  
 - Endpoints and Usage  
 - Error Handling  
6. Data Migration  
 - Data Sources  
 - Migration Strategies  
 - Migration Tools and Scripts  
 - Validation and Testing  
7. Implementation Steps  
8. Troubleshooting  
9. Conclusion  
10. References

## Overview

This document provides detailed instructions and best practices for integrating external APIs and performing data migration as part of Day 3 of the Marketplace Builder Hackathon 2025. Participants will learn how to connect their marketplace application with third-party APIs, handle authentication, manage endpoints, and execute robust data migration strategies to ensure seamless data transfer.

## Objectives

- Integrate External APIs: Connect the marketplace application to external services, fetch or post data using RESTful API calls.  
- Data Migration: Migrate existing data into the new application’s format or from legacy systems to a modern database.  
- Error Handling and Testing: Implement proper error handling for API calls and validate data integrity during migration.

## Prerequisites

- Basic Knowledge of REST APIs: Understanding HTTP methods (GET, POST, PUT, DELETE) and status codes.  
- Familiarity with JavaScript/TypeScript: Most examples are provided using JavaScript or TypeScript.  
- Database Fundamentals: Knowledge of SQL or NoSQL databases, as applicable.  
- Environment Setup: Node.js, a code editor (e.g., VS Code), and Git are recommended.  
- Existing Project Setup: Ensure you have set up your project repository, and all necessary dependencies are installed.

## Environment Setup

1. Clone the Repository:

git clone https://github.com/panaverse/learn-nextjs.git

cd learn-nextjs/HACKATHONS/Marketplace\_Builder\_Hackathon\_2025

2. Install Dependencies:

npm install or yarn install

3. Configuration: Create and configure environment variables as needed (e.g., API keys, database URIs). For example, create a .env file:

API\_KEY=your\_api\_key\_here

DATABASE\_URI=your\_database\_connection\_string\_here

## API Integration

### Understanding the API

Before integration, thoroughly review the API documentation provided by the third-party service.  
Identify the base URL and endpoints that will be used in your application.

### Authentication & Authorization

API Keys/OAuth: Depending on the API, you may need to use API keys, OAuth tokens, or other authentication methods.

Example (Using API Key):

const apiKey = process.env.API\_KEY;  
const headers = {  
 "Authorization": `Bearer ${apiKey}`,  
 "Content-Type": "application/json",  
};

### Endpoints and Usage

GET Requests: Fetch data from the API.

fetch("https://api.example.com/products", { headers })  
 .then(response => response.json())  
 .then(data => console.log(data))  
 .catch(error => console.error("Error:", error));

POST Requests: Send data to the API.

fetch("https://api.example.com/products", {  
 method: "POST",  
 headers,  
 body: JSON.stringify({ name: "New Product", price: 29.99 }),  
})  
 .then(response => response.json())  
 .then(data => console.log(data))  
 .catch(error => console.error("Error:", error));

### Error Handling

Response Checks: Always check response status codes.  
Retry Mechanisms: Implement retry logic if requests fail due to network issues.  
Logging: Log errors for debugging and monitoring.

## Data Migration

### Data Sources

Identify the current location and format of legacy data.

### Migration Strategies

ETL (Extract, Transform, Load): Outline steps to extract data, transform it into the required format, and load it into the new database.  
Incremental vs. Bulk Migration: Decide whether to perform a one-time full migration or an incremental migration to minimize downtime.

### Migration Tools and Scripts

Custom Scripts: Write scripts (e.g., using Node.js, Python) to automate the migration process.

const migrateData = async () => {  
 const legacyData = await fetchLegacyData();  
 const transformedData = transformData(legacyData);  
 await loadDataToDatabase(transformedData);  
};

### Validation and Testing

Data Integrity: Verify that all data has been accurately migrated.  
Automated Tests: Write tests to compare the legacy and migrated data.  
Rollback Strategy: Always have a rollback plan in case something goes wrong.

## Implementation Steps

1. Set Up Environment: Configure API keys and database connections.  
2. Develop API Integration: Implement functions for GET, POST, PUT, DELETE operations. Handle authentication, headers, and error responses.  
3. Create Data Migration Scripts: Write scripts to extract and transform data. Execute and validate the data migration.  
4. Testing and Debugging: Test API endpoints with tools like Postman. Run automated tests for data migration. Debug any issues and refine error handling.  
5. Documentation and Review: Update code comments and documentation. Review the implementation with team members.

## Troubleshooting

API Errors: Ensure your API key/token is valid.  
Data Mismatch: Validate the data types and formats between source and target.  
Network Issues: Implement retry logic. Use network debugging tools to diagnose connectivity issues.

## Conclusion

Day 3 of the hackathon focuses on integrating external APIs and migrating data into your marketplace application. By following this guide, you should be able to set up a reliable API integration, perform data migrations with confidence, and ensure that your application can handle data from both external and legacy sources efficiently.

## References

[API Provider Documentation](#) – Link to the external API documentation.  
[Database Migration Best Practices](#) – Reference material on data migration.  
[Error Handling in JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control\_flow\_and\_error\_handling)