Independent Skill Development Mission

Independent Skill Development Mission NXT
EMPOWERING FUTURES, SHAPING TOMORROW

ISDM (INDEPENDENT SKILL DEVELOPMENT MISSION

DEPLOYING BACKEND ON HEROKU/RENDER

CHAPTER 1: INTRODUCTION TO BACKEND DEPLOYMENT

1.1 What is Backend Deployment?

Backend deployment is the process of making a **Node.js/Express.js API** available online, so it can be accessed by frontend applications and external users.

1.2 Why Deploy on Heroku or Render?

✅ **Heroku** – Cloud platform offering **easy deployment,** managed databases, and scaling options.
✅ **Render** – Modern cloud platform with **free hosting, automatic deployments, and SSL certificates**.

CHAPTER 2: DEPLOYING BACKEND ON HEROKU

2.1 Prerequisites

✓ **Git Installed** – Check with git --version
✓ **Heroku CLI Installed** – Download from https://devcenter.heroku.com/articles/heroku-cli
✓ **Heroku Account** – Sign up at https://signup.heroku.com/

2.2 Initialize Git Repository

ISDM-NXT

1. Navigate to your backend folder:

sh

CopyEdit

cd your-backend-folder

2. Initialize a Git repository:

sh

CopyEdit

git init

git add .

git commit -m "Initial commit"

2.3 Login to Heroku

sh

CopyEdit

heroku login

This opens a browser window. Click **Login** to authenticate.

2.4 Create a New Heroku App

sh

CopyEdit

heroku create your-app-name

If no name is provided, Heroku will generate a random name.

2.5 Add Heroku Remote Repository

sh

ISDM-NXT

CopyEdit

git remote -v

You should see something like:

perl

CopyEdit

heroku https://git.heroku.com/your-app-name.git (fetch)

heroku https://git.heroku.com/your-app-name.git (push)

2.6 Deploy Code to Heroku

sh

CopyEdit

git push heroku master

2.7 Set Environment Variables on Heroku

If your project uses a .env file, configure variables manually:

sh

CopyEdit

heroku config:set MONGO_URI=your_mongo_connection_string

heroku config:set JWT_SECRET=your_secret_key

2.8 Scale the Application

Ensure at least **one instance** is running:

sh

CopyEdit

heroku ps:scale web=1

ISDM-NXT

2.9 Open the Deployed App

sh

CopyEdit

heroku open

✅ **Your backend is now live on Heroku!**

CHAPTER 3: DEPLOYING BACKEND ON RENDER

3.1 Prerequisites

✓ **GitHub Repository** – Push your backend code to GitHub.
✓ **Render Account** – Sign up at https://render.com/

3.2 Create a New Web Service on Render

1. Go to https://dashboard.render.com/.
2. Click **New Web Service** and choose **GitHub Repository**.
3. Select your backend repository and click **Connect**.

3.3 Configure Deployment Settings

- **Build Command:**

sh

CopyEdit

npm install

- **Start Command:**

sh

CopyEdit

node server.js

ISDM-NXT

3.4 Set Environment Variables

Go to **Environment Variables** in Render and add:

ini

CopyEdit

MONGO_URI=your_mongo_connection_string

JWT_SECRET=your_secret_key

3.5 Deploy the Service

1. Click **Deploy**.
2. Wait for **logs to show "Live"** status.

3.6 Check the Live API

Copy the **Render URL** and test with:

arduino

CopyEdit

https://your-app.onrender.com/api/test

✅ **Your backend is now live on Render!**

CHAPTER 4: TESTING THE DEPLOYED BACKEND

4.1 Check API Endpoints with Postman

1. Open Postman.
2. Send a **GET request** to:

arduino

CopyEdit

ISDM-NXT

https://your-app.herokuapp.com/api/test

3. Expected Response:

json

CopyEdit

{ "message": "API is working!" }

4.2 Update Code and Redeploy

For **Heroku**:

sh

CopyEdit

git add .

git commit -m "Updated API"

git push heroku master

For **Render**:

- **Automatic redeployment** on GitHub push.
- Manually trigger deployment from **Render Dashboard**.

CHAPTER 5: DEBUGGING & COMMON ISSUES

Issue	Solution
Application Error on Heroku	Run heroku logs --tail to check errors.
MongoDB connection failed	Ensure MONGO_URI is set correctly in .env.

ISDM-NXT

CORS issues	Install and enable cors middleware in Express.
Render deployment stuck	Check logs in **Render Dashboard** for errors.

Conclusion

✅ **Deployed a backend API using Heroku & Render.**
✅ **Configured environment variables securely.**
✅ **Tested API endpoints with Postman.**

🚀 **Next Steps:**

- Add **JWT Authentication**.
- Implement **MongoDB Atlas for cloud storage**.
- Deploy a **React frontend to connect with the backend**!

By following this guide, you can **successfully deploy and scale backend applications online!** 🎉 🔥

ISDM-NXT

Deployment & Hosting: Deploying Frontend on Netlify/Vercel

Chapter 1: Introduction to Frontend Deployment

1.1 Why Deploy a Frontend?

After developing a frontend application using **React.js**, it needs to be hosted online for users to access. **Netlify and Vercel** are two popular services that provide **fast, free, and reliable** hosting for frontend applications.

✅ **Benefits of Deploying on Netlify & Vercel**

- **Easy CI/CD integration** – Automatically deploys from GitHub/GitLab.
- **Free tier available** – Provides generous free hosting for personal projects.
- **Serverless functions** – Supports backend logic without needing a separate server.
- **Custom domains & HTTPS** – Secure sites with SSL certificates included.
- **Global CDN (Content Delivery Network)** – Improves performance with edge caching.

Chapter 2: Preparing Your React App for Deployment

2.1 Build the React App

Before deploying, generate an **optimized production build** using:

```
npm run build
```

ISDM-NxT

This creates a /build folder containing **optimized JavaScript, CSS, and HTML files**.

✅ **Verify the Build Locally**

Use **serve** to test the build before deployment:

npm install -g serve

serve -s build

Visit **http://localhost:3000/** to check if everything is working.

CHAPTER 3: DEPLOYING REACT APP ON NETLIFY

3.1 Option 1: Deploy via Netlify CLI

Step 1: Install Netlify CLI

npm install -g netlify-cli

Step 2: Login to Netlify

netlify login

This opens a browser for authentication.

Step 3: Deploy the Build Folder

netlify deploy --prod --dir=build

✅ **Your app is now live!** 🎉

3.2 Option 2: Deploy via GitHub (Recommended)

Step 1: Push Your React App to GitHub

git init

git add .

```
git commit -m "Initial commit"

git branch -M main

git remote add origin https://github.com/your-username/your-repo.git

git push -u origin main
```

✅ Ensure your project is **public** on GitHub.

Step 2: Connect GitHub to Netlify

1. Sign up or log in at Netlify.
2. Click **New Site from Git** → Select **GitHub** → Choose your repo.
3. Configure settings:
   - **Build Command:** npm run build
   - **Publish Directory:** build/
4. Click **Deploy Site**.

✅ **Your React app is now deployed on Netlify!**

3.3 Setting Up a Custom Domain on Netlify

Step 1: Change the Default Netlify Subdomain

1. Go to **Site Settings** → **Domain Management**.
2. Click **Edit Site Name** to customize the URL.

Step 2: Connect a Custom Domain

1. Buy a domain from **Namecheap, GoDaddy, or Google Domains**.

ISDM-NXT

2. Update your **DNS settings** to point to Netlify.
3. Enable **HTTPS (SSL)** for security.

✅ Your React app now runs on a custom domain! 🎉

Chapter 4: Deploying React App on Vercel

4.1 Option 1: Deploy via Vercel CLI

Step 1: Install Vercel CLI

npm install -g vercel

Step 2: Login to Vercel

vercel login

This opens a browser for authentication.

Step 3: Deploy the Build Folder

vercel

✅ Your app is now live on **Vercel's global CDN!** 🎉

4.2 Option 2: Deploy via GitHub (Recommended)

Step 1: Push React App to GitHub

If you haven't already, push your React app to GitHub using:

git add .

git commit -m "Deploying React app"

git push origin main

Step 2: Connect GitHub to Vercel

ISDM-NXT

1. Go to Vercel and log in.
2. Click **New Project** → Select **GitHub** → Choose your repo.
3. Configure settings:
    - **Build Command:** npm run build
    - **Output Directory:** build/
4. Click **Deploy**.

✅ Your app is now deployed on **Vercel's high-speed servers**. 🎉

4.3 Setting Up a Custom Domain on Vercel

Step 1: Get a Free Vercel Subdomain

Vercel provides a **free .vercel.app subdomain**.

Step 2: Connect a Custom Domain

1. Go to **Project Settings** → **Domains**.
2. Add a **custom domain** and update DNS settings.
3. Enable **SSL (HTTPS)** for security.

✅ **Your React app now has a custom domain with Vercel!**

Chapter 5: Continuous Deployment (CI/CD) for Auto Updates

5.1 What is Continuous Deployment?

CI/CD ensures your **frontend updates automatically** when you push changes to GitHub.

5.2 Enable Auto Deployment on Netlify

1. Go to **Deploy Settings → Build & Deploy**.
2. Enable **Continuous Deployment** for automatic updates.

5.3 Enable Auto Deployment on Vercel

1. Go to **Project Settings → Git Integration**.
2. Enable **Auto Deploy on Push**.

✅ **Now, any code change is automatically deployed!** 🎉

CHAPTER 6: OPTIMIZING YOUR REACT APP FOR DEPLOYMENT

6.1 Reduce Bundle Size

Install webpack-bundle-analyzer to analyze and optimize the build size.

npm install --save-dev webpack-bundle-analyzer

Modify package.json:

```
"scripts": {
  "analyze": "source-map-explorer build/static/js/*.js"
}
```

Run the analyzer:

npm run analyze

✅ **Identify and remove unnecessary dependencies.**

6.2 Enable Lazy Loading

Lazy load components using React's **lazy() and Suspense**.

```
import { lazy, Suspense } from "react";
```

```
const Dashboard = lazy(() => import("./Dashboard"));


function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Dashboard />
    </Suspense>
  );
}
```

✅ **Improves page load speed by loading components on demand.**

6.3 Enable Gzip Compression

Netlify and Vercel automatically **compress files** for better performance.
To verify compression, check your **Network Tab** in Chrome DevTools.

6.4 Use a CDN for Static Assets

Upload large images & videos to **Cloudinary** or **AWS S3** instead of keeping them inside the React app.

✅ **Faster load times & reduced hosting costs.**

ISDM-NXT

Chapter 7: Hands-on Practice & Assignments

Exercise 1: Deploy a React App on Netlify

1. **Create a simple React app** (create-react-app my-app).
2. **Push the project to GitHub**.
3. **Deploy on Netlify** via GitHub integration.
4. **Add a custom domain** to your Netlify site.

Exercise 2: Deploy a React App on Vercel

1. **Push your React app to GitHub**.
2. **Deploy using Vercel's GitHub integration**.
3. **Enable automatic deployments** on push.
4. **Optimize your build for faster loading**.

Conclusion

🎉 You have successfully learned how to:

✅ Deploy **React.js frontend** on **Netlify & Vercel**.

✅ Use **GitHub for continuous deployment**.

✅ Optimize performance with **lazy loading & bundle analysis**.

✅ Set up **custom domains & HTTPS** for security.

🚀 **Next Steps:**

- Deploy a **full-stack MERN app** (Frontend on Vercel, Backend on Heroku).
- Integrate **serverless functions** for API handling.
- Monitor **performance with Lighthouse & Google PageSpeed Insights**.

ISDM-NXT

Happy Deploying! 🎉 🚀

ISDM-NXT

Deployment & Hosting: Deploying Frontend on Netlify/Vercel

Chapter 1: Introduction to Frontend Deployment

1.1 Why Deploy a Frontend?

After developing a frontend application using **React.js**, it needs to be hosted online for users to access. **Netlify and Vercel** are two popular services that provide **fast, free, and reliable** hosting for frontend applications.

✅ **Benefits of Deploying on Netlify & Vercel**

- **Easy CI/CD integration** – Automatically deploys from GitHub/GitLab.
- **Free tier available** – Provides generous free hosting for personal projects.
- **Serverless functions** – Supports backend logic without needing a separate server.
- **Custom domains & HTTPS** – Secure sites with SSL certificates included.
- **Global CDN (Content Delivery Network)** – Improves performance with edge caching.

Chapter 2: Preparing Your React App for Deployment

2.1 Build the React App

Before deploying, generate an **optimized production build** using:

```
npm run build
```

ISDM-NxT

This creates a /build folder containing **optimized JavaScript, CSS, and HTML files**.

✅ **Verify the Build Locally**

Use **serve** to test the build before deployment:

npm install -g serve

serve -s build

Visit **http://localhost:3000/** to check if everything is working.

CHAPTER 3: DEPLOYING REACT APP ON NETLIFY

3.1 Option 1: Deploy via Netlify CLI

Step 1: Install Netlify CLI

npm install -g netlify-cli

Step 2: Login to Netlify

netlify login

This opens a browser for authentication.

Step 3: Deploy the Build Folder

netlify deploy --prod --dir=build

✅ **Your app is now live!** 🎉

3.2 Option 2: Deploy via GitHub (Recommended)

Step 1: Push Your React App to GitHub

git init

git add .

git commit -m "Initial commit"

git branch -M main

git remote add origin https://github.com/your-username/your-repo.git

git push -u origin main

✅ Ensure your project is **public** on GitHub.

Step 2: Connect GitHub to Netlify

1. Sign up or log in at Netlify.
2. Click **New Site from Git** → Select **GitHub** → Choose your repo.
3. Configure settings:
    - **Build Command:** npm run build
    - **Publish Directory:** build/
4. Click **Deploy Site**.

✅ **Your React app is now deployed on Netlify!**

3.3 Setting Up a Custom Domain on Netlify

Step 1: Change the Default Netlify Subdomain

1. Go to **Site Settings** → **Domain Management**.
2. Click **Edit Site Name** to customize the URL.

Step 2: Connect a Custom Domain

1. Buy a domain from **Namecheap, GoDaddy, or Google Domains**.

ISDM-NXT

2. Update your **DNS settings** to point to Netlify.
3. Enable **HTTPS (SSL)** for security.

✅ Your React app now runs on a custom domain! 🎉

Chapter 4: Deploying React App on Vercel

4.1 Option 1: Deploy via Vercel CLI

Step 1: Install Vercel CLI

npm install -g vercel

Step 2: Login to Vercel

vercel login

This opens a browser for authentication.

Step 3: Deploy the Build Folder

vercel

✅ Your app is now live on **Vercel's global CDN!** 🎉

4.2 Option 2: Deploy via GitHub (Recommended)

Step 1: Push React App to GitHub

If you haven't already, push your React app to GitHub using:

git add .

git commit -m "Deploying React app"

git push origin main

Step 2: Connect GitHub to Vercel

ISDM-NXT

1. Go to Vercel and log in.
2. Click **New Project** → Select **GitHub** → Choose your repo.
3. Configure settings:
    - **Build Command:** npm run build
    - **Output Directory:** build/
4. Click **Deploy**.

✅ Your app is now deployed on **Vercel's high-speed servers**. 🎉

4.3 Setting Up a Custom Domain on Vercel

Step 1: Get a Free Vercel Subdomain

Vercel provides a **free .vercel.app subdomain**.

Step 2: Connect a Custom Domain

1. Go to **Project Settings** → **Domains**.
2. Add a **custom domain** and update DNS settings.
3. Enable **SSL (HTTPS)** for security.

✅ **Your React app now has a custom domain with Vercel!**

Chapter 5: Continuous Deployment (CI/CD) for Auto Updates

5.1 What is Continuous Deployment?

CI/CD ensures your **frontend updates automatically** when you push changes to GitHub.

5.2 Enable Auto Deployment on Netlify

1. Go to **Deploy Settings** → **Build & Deploy**.

2. Enable **Continuous Deployment** for automatic updates.

5.3 Enable Auto Deployment on Vercel

1. Go to **Project Settings → Git Integration**.
2. Enable **Auto Deploy on Push**.

✅ **Now, any code change is automatically deployed!** 🎉

Chapter 6: Optimizing Your React App for Deployment

6.1 Reduce Bundle Size

Install webpack-bundle-analyzer to analyze and optimize the build size.

npm install --save-dev webpack-bundle-analyzer

Modify package.json:

"scripts": {

"analyze": "source-map-explorer build/static/js/*.js"

}

Run the analyzer:

npm run analyze

✅ **Identify and remove unnecessary dependencies.**

6.2 Enable Lazy Loading

Lazy load components using React's **lazy() and Suspense**.

import { lazy, Suspense } from "react";

ISDM-NXT

```
const Dashboard = lazy(() => import("./Dashboard"));


function App() {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Dashboard />
    </Suspense>
  );
}
```

✅ **Improves page load speed by loading components on demand.**

6.3 Enable Gzip Compression

Netlify and Vercel automatically **compress files** for better performance.
To verify compression, check your **Network Tab** in Chrome DevTools.

6.4 Use a CDN for Static Assets

Upload large images & videos to **Cloudinary** or **AWS S3** instead of keeping them inside the React app.

✅ **Faster load times & reduced hosting costs.**

Chapter 7: Hands-on Practice & Assignments

ISDM-NXT

Exercise 1: Deploy a React App on Netlify

1. **Create a simple React app** (create-react-app my-app).
2. **Push the project to GitHub**.
3. **Deploy on Netlify** via GitHub integration.
4. **Add a custom domain** to your Netlify site.

Exercise 2: Deploy a React App on Vercel

1. **Push your React app to GitHub**.
2. **Deploy using Vercel's GitHub integration**.
3. **Enable automatic deployments** on push.
4. **Optimize your build for faster loading**.

Conclusion

🎉 You have successfully learned how to:

✅ Deploy **React.js frontend** on **Netlify & Vercel**.

✅ Use **GitHub for continuous deployment**.

✅ Optimize performance with **lazy loading & bundle analysis**.

✅ Set up **custom domains & HTTPS** for security.

🚀 **Next Steps:**

- Deploy a **full-stack MERN app** (Frontend on Vercel, Backend on Heroku).
- Integrate **serverless functions** for API handling.
- Monitor **performance with Lighthouse & Google PageSpeed Insights**.

Happy Deploying! 🎉 🚀

ISDM-NXT

Automating Deployment with GitHub Actions: A Complete Guide

Chapter 1: Introduction to GitHub Actions

1.1 What is GitHub Actions?

GitHub Actions is a **CI/CD (Continuous Integration/Continuous Deployment) automation tool** that enables developers to **build, test, and deploy** their applications **directly from GitHub**.

1.2 Why Use GitHub Actions for Deployment?

✅ **Automates the deployment process** – No need for manual deployments.
✅ **Ensures consistency** – Deployments follow the same steps every time.
✅ **Enables testing** – Run unit and integration tests before deploying.
✅ **Integrates with cloud services** – Deploy to **Heroku, Render, Netlify, Vercel, AWS, or Firebase**.

1.3 How Does GitHub Actions Work?

- **Workflows** – Automated processes triggered by GitHub events (push, pull request).
- **Jobs** – Steps executed within a workflow.
- **Actions** – Predefined tasks used in workflows.

Chapter 2: Setting Up GitHub Actions for MERN Deployment

2.1 Prerequisites

ISDM-NxT

✓ A **GitHub repository** with a **MERN application**.
✓ A **Heroku/Render (backend) and Netlify/Vercel (frontend) account**.
✓ Environment variables for database connection (MONGO_URI) and authentication (JWT_SECRET).

CHAPTER 3: AUTOMATING BACKEND DEPLOYMENT (EXPRESS & MONGODB)

3.1 Deploying Backend to Heroku

Step 1: Create a GitHub Actions Workflow

Inside the **backend** repository, create a .github/workflows/deploy.yml file:

```
name: Deploy Backend to Heroku

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
```

ISDM-NXT

```
- name: Checkout Repository
  uses: actions/checkout@v2

- name: Set up Node.js
  uses: actions/setup-node@v2
  with:
    node-version: '18'

- name: Install Dependencies
  run: npm install

- name: Deploy to Heroku
  uses: akhileshns/heroku-deploy@v3.12.12
  with:
    heroku_api_key: ${{ secrets.HEROKU_API_KEY }}
    heroku_app_name: "mern-ecommerce-backend"
    heroku_email: "your-email@example.com"
    usedocker: false
```

Step 2: Set Up Secrets in GitHub

1. Go to your **GitHub repository → Settings → Secrets and Variables → Actions**.
2. Add the following secrets:

ISDM-NXT

- **HEROKU_API_KEY** – Found in your Heroku account under API keys.
- **HEROKU_APP_NAME** – The name of your Heroku app (e.g., mern-ecommerce-backend).

Step 3: Push Changes to GitHub

git add .

git commit -m "Automated deployment setup"

git push origin main

✅ **Now, GitHub Actions will automatically deploy your backend to Heroku on every push!** 🎉

3.2 Deploying Backend to Render

If using **Render** instead of Heroku, modify .github/workflows/deploy.yml:

```
name: Deploy Backend to Render


on:
  push:
    branches:
      - main


jobs:
  deploy:
    runs-on: ubuntu-latest
```

ISDM-NXT

```
steps:
- name: Checkout Repository
  uses: actions/checkout@v2

- name: Set up Node.js
  uses: actions/setup-node@v2
  with:
    node-version: '18'

- name: Install Dependencies
  run: npm install

- name: Deploy to Render
  env:
    RENDER_API_KEY: ${{ secrets.RENDER_API_KEY }}
  run: |
    curl -X POST "https://api.render.com/v1/services/{your-service-id}/deploys" \
    -H "Authorization: Bearer $RENDER_API_KEY"
```

✅ **Render will now deploy the backend whenever you push changes!**

ISDM-NXT

CHAPTER 4: AUTOMATING FRONTEND DEPLOYMENT (REACT.JS)

4.1 Deploying Frontend to Netlify

Step 1: Create .github/workflows/frontend.yml in the Frontend Repository

```
name: Deploy Frontend to Netlify

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v2

    - name: Install Dependencies
      run: npm install

    - name: Build React App
```

ISDM-NXT

```
    run: npm run build


  - name: Deploy to Netlify
    uses: netlify/actions/cli@master
    with:
      args: deploy --prod
    env:
      NETLIFY_AUTH_TOKEN: ${{ secrets.NETLIFY_AUTH_TOKEN }}
      NETLIFY_SITE_ID: ${{ secrets.NETLIFY_SITE_ID }}
```

Step 2: Set Up Secrets in GitHub

1. Go to **GitHub → Settings → Secrets and Variables → Actions**.
2. Add:
   - **NETLIFY_AUTH_TOKEN** – Found in Netlify **User Settings → Applications**.
   - **NETLIFY_SITE_ID** – Found in **Netlify Dashboard → Site Settings**.

✅ **Now, every push to main triggers a new Netlify deployment!** 🎉

4.2 Deploying Frontend to Vercel

Step 1: Create .github/workflows/frontend.yml for Vercel

```
name: Deploy Frontend to Vercel
```

```
on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Repository
      uses: actions/checkout@v2

    - name: Install Dependencies
      run: npm install

    - name: Build React App
      run: npm run build

    - name: Deploy to Vercel
      uses: amondnet/vercel-action@v20
      with:
```

ISDM-NXT

```
vercel-token: ${{ secrets.VERCEL_TOKEN }}

vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}

vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
```

Step 2: Set Up Secrets in GitHub

1. Add the following secrets in GitHub:
    - **VERCEL_TOKEN** – Found in Vercel **Account Settings → API Tokens**.
    - **VERCEL_ORG_ID** – Found in **Vercel Settings → Team**.
    - **VERCEL_PROJECT_ID** – Found in **Vercel Project Settings**.

✅ **Now, the frontend is automatically deployed to Vercel on every push!** 🎉

CHAPTER 5: AUTOMATING TESTS WITH GITHUB ACTIONS

5.1 Running Unit Tests Before Deployment

Modify .github/workflows/deploy.yml to **run Jest tests** before deployment:

```
- name: Run Tests

  run: npm test
```

✅ **If tests fail, deployment is stopped!**

Chapter 6: Full CI/CD Pipeline Overview

Component	Hosting Service	GitHub Action Workflow
Backend	Heroku/Render	deploy.yml
Frontend	Netlify/Vercel	frontend.yml
Testing	Jest	Runs before deployment

Chapter 7: Hands-on Practice & Assignments

Exercise 1: Implement CI/CD for a Different Cloud Provider

- Deploy to **AWS Lambda, DigitalOcean, or Firebase** instead of Heroku.

Exercise 2: Add Database Migrations in CI/CD

- Use **MongoDB Atlas** with **automated schema updates**.

Exercise 3: Set Up CI/CD for Microservices

- Deploy **backend, authentication, and admin services separately**.

Conclusion

🚀 **GitHub Actions automates the deployment of your MERN app to cloud services.**

✅ **Automated CI/CD reduces manual effort and ensures error-free releases.**

✅ **Full-stack deployment integrates backend (Heroku/Render) and frontend (Netlify/Vercel).**

🎉 **Congratulations! You now have a production-ready CI/CD pipeline for your MERN project!** 🚀

ISDM-NxT

CI/CD & DevOps: CI/CD Pipelines for MERN Applications

Chapter 1: Introduction to CI/CD for MERN Applications

1.1 What is CI/CD?

CI/CD (**Continuous Integration & Continuous Deployment**) is a **DevOps practice** that automates the process of building, testing, and deploying applications.

✅ **CI (Continuous Integration)**

- Developers push code to a **Git repository (GitHub, GitLab, Bitbucket)**.
- The CI system automatically **runs tests & checks for errors**.

✅ **CD (Continuous Deployment/Delivery)**

- Code changes are **automatically deployed** to staging or production servers.
- Ensures that the **latest stable version** of the app is live.

1.2 Why Use CI/CD in a MERN App?

🚀 Automates **code testing & deployment**.
⚡ Reduces **manual errors & improves efficiency**.
📈 Enables **faster release cycles** for new features.
✅ Ensures that the app **remains stable** with every update.

Chapter 2: Setting Up a CI/CD Pipeline for MERN Applications

2.1 CI/CD Workflow for MERN Applications

- **Step 1:** Developer pushes code to **GitHub/GitLab**.
- **Step 2: CI server** (GitHub Actions, GitLab CI/CD) automatically builds & tests the app.
- **Step 3:** If tests pass, the code is **deployed to staging or production**.

CHAPTER 3: SETTING UP A CI/CD PIPELINE WITH GITHUB ACTIONS

3.1 What is GitHub Actions?

GitHub Actions automates software workflows, including:

✅ **Running tests** before deployment.
✅ **Building the frontend (React.js)**.
✅ **Deploying the backend (Node.js, Express, MongoDB)**.

Chapter 4: Creating a CI/CD Pipeline for a MERN App

4.1 Adding a GitHub Actions Workflow

Create a .github/workflows/ci-cd.yml file in your MERN project repository.

```
name: CI/CD Pipeline for MERN App

on:
  push:
    branches:
      - main

jobs:
```

ISDM-NXT

```
build:
 runs-on: ubuntu-latest

 steps:
  - name: Checkout Repository
   uses: actions/checkout@v3

  - name: Setup Node.js
   uses: actions/setup-node@v3
   with:
    node-version: 18

  - name: Install Backend Dependencies
   run: |
    cd backend
    npm install

  - name: Install Frontend Dependencies
   run: |
    cd frontend
    npm install
```

ISDM-NXT

```
    - name: Run Tests
      run: |
        cd backend
        npm test
        cd ../frontend
        npm test

    - name: Build React App
      run: |
        cd frontend
        npm run build

    - name: Deploy to Heroku (Backend)
      if: success()
      env:
        HEROKU_API_KEY: ${{ secrets.HEROKU_API_KEY }}
        HEROKU_APP_NAME: my-mern-backend
      run: |
        cd backend
        git remote add heroku
https://git.heroku.com/$HEROKU_APP_NAME.git
        git push heroku main
```

ISDM-NXT

```
- name: Deploy to Netlify (Frontend)
  if: success()
  run: |
    cd frontend
    npm install -g netlify-cli
    netlify deploy --prod --dir=build --auth=${{ secrets.NETLIFY_AUTH_TOKEN }}
```

✅ **How it Works:**

- Runs on **every push to the main branch**.
- **Builds & tests** the backend and frontend.
- **Deploys the backend to Heroku**.
- **Deploys the frontend to Netlify**.

Chapter 5: Storing API Keys Securely

5.1 Using GitHub Secrets for Sensitive Data

1. Go to your **GitHub Repository → Settings → Secrets and variables → Actions**.
2. Add the following secrets:
   - HEROKU_API_KEY: Your **Heroku API Key**.
   - NETLIFY_AUTH_TOKEN: Your **Netlify authentication token**.

✅ **Why use GitHub Secrets?**

🔒 Prevents exposing **sensitive credentials** in your code.

⚡ Allows automated deployment **without manual authentication**.

Chapter 6: Implementing CI/CD with GitLab CI/CD

6.1 Add a .gitlab-ci.yml File for GitLab Pipelines

```
stages:
 - test
 - build
 - deploy


test_backend:
 stage: test
 script:
  - cd backend
  - npm install
  - npm test


test_frontend:
 stage: test
 script:
  - cd frontend
  - npm install
  - npm test
```

ISDM-NXT

```
build_frontend:
  stage: build
  script:
    - cd frontend
    - npm run build

deploy_backend:
  stage: deploy
  script:
    - cd backend
    - git push heroku main
  only:
    - main

deploy_frontend:
  stage: deploy
  script:
    - cd frontend
    - netlify deploy --prod --dir=build
  only:
    - main
```

✅ **This GitLab CI/CD Pipeline:**

ISDM-NXT

1. **Runs tests** before deployment.
2. **Builds React frontend**.
3. **Deploys backend & frontend** only if tests pass.

Chapter 7: Automating Deployments to Heroku & Netlify

7.1 Automate Backend Deployment to Heroku

heroku login

heroku create my-mern-backend

heroku config:set MONGO_URI="your-mongodb-uri"

git push heroku main

7.2 Automate Frontend Deployment to Netlify

npm install -g netlify-cli

netlify login

netlify deploy --prod --dir=frontend/build

✅ **Now, every push to GitHub automatically deploys the latest version!** 🎉

Chapter 8: Monitoring and Debugging CI/CD Pipelines

8.1 View Deployment Logs in GitHub Actions

1. Go to **GitHub Repository → Actions**.
2. Click on the **latest workflow run** to see logs.
3. Check for **failed steps** and fix errors.

8.2 Check Deployment Logs in Heroku

heroku logs --tail

✅ **Fix any issues preventing deployment.**

Chapter 9: Best Practices for CI/CD Pipelines

✅ **Run Tests Before Deployment**

- Ensure that **only error-free code** is deployed.

✅ **Use Environment Variables**

- Store API keys & database credentials **securely**.

✅ **Enable Auto Rollbacks**

- If deployment fails, **restore the previous version**.

✅ **Monitor Performance After Deployment**

- Use **New Relic, Prometheus, or Datadog** to track server health.

✅ **Enable Slack/Email Notifications**

- Send alerts if **CI/CD pipeline fails**.

Chapter 10: Hands-on Practice & Assignments

Exercise 1: Set Up GitHub Actions for Your MERN App

1. Add **GitHub Actions workflow** (ci-cd.yml).
2. Ensure the **pipeline runs on push** to the main branch.
3. Store **API keys securely** using GitHub Secrets.

Exercise 2: Deploy Backend & Frontend Automatically

1. Deploy the **backend to Heroku** using GitHub Actions.
2. Deploy the **frontend to Netlify** using GitHub Actions.
3. Verify that **every commit auto-updates the live site**.

Exercise 3: Debug & Improve CI/CD Pipeline

1. View the **CI/CD logs** in GitHub Actions.
2. Fix any **deployment failures**.
3. Enable **Slack or Email notifications** for failed deployments.

Conclusion

🎉 You have successfully learned:

✅ **How CI/CD pipelines work for MERN apps**
✅ **Automated testing, building, and deployment**
✅ **Using GitHub Actions & GitLab CI/CD**
✅ **Deploying React & Node.js apps seamlessly**

🚀 **Next Steps:**

- Add **Docker support** for containerized deployments.
- Implement **Blue-Green Deployment** for zero downtime.
- Use **Kubernetes & AWS ECS** for scaling!

Happy Deploying! 🎉 🚀

ISDM-NXT

CAPSTONE PROJECT: BUILD A FULL-STACK MERN E-COMMERCE APP

PROJECT OVERVIEW

In this capstone project, you will **build a full-stack E-Commerce application** using the **MERN (MongoDB, Express.js, React.js, Node.js) stack**. The application will allow users to browse products, add them to the cart, and make purchases.

PROJECT FEATURES

1. User Authentication

- ✅ **Register/Login with JWT authentication**
- ✅ **Role-based access (Admin/User)**
- ✅ **Secure user session management**

2. Product Management

- ✅ **Admin can add, update, and delete products**
- ✅ **Users can browse available products**
- ✅ **Product categories and search functionality**

3. Shopping Cart & Orders

- ✅ **Users can add/remove products from the cart**
- ✅ **Checkout and order placement**
- ✅ **Order tracking and history**

4. Payment Integration

- ✅ **Stripe or PayPal integration for online payments**
- ✅ **Cash on Delivery (COD) option**

5. Admin Dashboard

✅ **Manage users, products, and orders**
✅ **View sales reports and analytics**

6. Deployment & Hosting

✅ **Backend deployed on Heroku/Render**
✅ **Frontend deployed on Netlify/Vercel**
✅ **CI/CD pipeline setup with GitHub Actions**

Step 1: Setting Up the Backend (Node.js & Express.js)

1.1 Initialize a Node.js Project

Create a project folder and initialize Node.js:

mkdir mern-ecommerce

cd mern-ecommerce

npm init -y

1.2 Install Dependencies

npm install express mongoose dotenv cors jsonwebtoken bcryptjs cookie-parser multer stripe

- **express** – Backend framework.
- **mongoose** – MongoDB ODM.
- **dotenv** – Manages environment variables.
- **cors** – Handles Cross-Origin Resource Sharing.
- **jsonwebtoken (JWT)** – Secure user authentication.
- **bcryptjs** – Encrypts passwords.
- **cookie-parser** – Handles user authentication cookies.

ISDM-NXT

- **multer** – Manages file uploads.
- **stripe** – Payment integration.

1.3 Create server.js (Express Server Setup)

```
require("dotenv").config();

const express = require("express");

const mongoose = require("mongoose");

const cors = require("cors");


const app = express();

const PORT = process.env.PORT || 5000;


// Middleware

app.use(cors());

app.use(express.json());


// MongoDB Connection

mongoose.connect(process.env.MONGO_URI, { useNewUrlParser: true, useUnifiedTopology: true })

  .then(() => console.log("MongoDB Connected"))

  .catch(err => console.error(err));


// Routes
```

ISDM-NXT

```
app.get("/", (req, res) => {
  res.send("E-Commerce API is running...");
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

✅ **Now, your backend is connected to MongoDB.**

1.4 Setting Up Environment Variables (.env)

```
PORT=5000
MONGO_URI=mongodb+srv://<username>:<password>@cluster.mongodb.net/ecommerce
JWT_SECRET=mysecretkey123
STRIPE_SECRET_KEY=sk_test_4eC39HqLyjWDarjtT1zdp7dc
```

Step 2: Implementing User Authentication

2.1 Create User Schema (models/User.js)

```
const mongoose = require("mongoose");

const UserSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
```

```
  password: { type: String, required: true },
  isAdmin: { type: Boolean, default: false }
});

module.exports = mongoose.model("User", UserSchema);
```

2.2 Implement User Registration (routes/authRoutes.js)

```
const express = require("express");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const User = require("../models/User");

const router = express.Router();

// Register
router.post("/register", async (req, res) => {
  try {
    const hashedPassword = await bcrypt.hash(req.body.password, 10);
    const newUser = new User({ ...req.body, password: hashedPassword });
    await newUser.save();
    res.status(201).json({ message: "User registered successfully" });
  } catch (err) {
```

ISDM-NXT

```
    res.status(500).json({ error: err.message });
  }
});

// Login
router.post("/login", async (req, res) => {
  try {
    const user = await User.findOne({ email: req.body.email });
    if (!user) return res.status(404).json({ error: "User not found" });

    const isMatch = await bcrypt.compare(req.body.password,
user.password);
    if (!isMatch) return res.status(401).json({ error: "Invalid
credentials" });

    const token = jwt.sign({ id: user._id, isAdmin: user.isAdmin },
process.env.JWT_SECRET);
    res.json({ token });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});
```

ISDM-NxT

```
module.exports = router;
```

✅ **User authentication is now working!**

Step 3: Product Management

3.1 Create Product Schema (models/Product.js)

```
const mongoose = require("mongoose");

const ProductSchema = new mongoose.Schema({
  name: { type: String, required: true },
  price: { type: Number, required: true },
  description: { type: String },
  category: { type: String, required: true },
  image: { type: String }
});

module.exports = mongoose.model("Product", ProductSchema);
```

3.2 Implement CRUD Operations (routes/productRoutes.js)

```
const express = require("express");
const Product = require("../models/Product");

const router = express.Router();
```

ISDM-NXT

```
// Create Product
router.post("/", async (req, res) => {
  try {
    const newProduct = new Product(req.body);
    await newProduct.save();
    res.status(201).json(newProduct);
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// Get All Products
router.get("/", async (req, res) => {
  const products = await Product.find();
  res.json(products);
});

// Get Product by ID
router.get("/:id", async (req, res) => {
  const product = await Product.findById(req.params.id);
  if (!product) return res.status(404).json({ error: "Product not found" });
  res.json(product);
```

ISDM-NXT

```
});

module.exports = router;
```

✅ **Product API is now working!**

Step 4: Setting Up the React Frontend

4.1 Install React & Dependencies

```
npx create-react-app ecommerce-frontend
cd ecommerce-frontend
npm install axios react-router-dom
```

4.2 Fetch Products (src/components/ProductList.js)

```
import React, { useEffect, useState } from "react";
import axios from "axios";

function ProductList() {
  const [products, setProducts] = useState([]);

  useEffect(() => {
    axios.get("http://localhost:5000/products")
      .then(res => setProducts(res.data))
      .catch(err => console.error(err));
  }, []);
```

ISDM-NXT

```
  return (
    <div>
      <h2>Products</h2>
      {products.map(product => (
        <div key={product._id}>
          <h3>{product.name}</h3>
          <p>{product.description}</p>
          <p>${product.price}</p>
        </div>
      ))}
    </div>
  );
}

export default ProductList;
```

✅ **Frontend now fetches products from the backend!**

Final Steps: Deployment & CI/CD Setup

✅ **Deploy Backend to Heroku/Render**
✅ **Deploy Frontend to Netlify/Vercel**
✅ **Setup GitHub Actions for CI/CD**

ISDM-NXT

CONCLUSION

🎯 **Congratulations! You built a full-stack MERN e-commerce app.**

🔹 **Implemented authentication, product management, and payment integration.**

🔹 **Deployed the app using modern CI/CD tools.**

🚀 **Next Steps:**

- Implement **admin dashboard**.
- Add **search and filtering**.
- Integrate **payment processing with Stripe/PayPal**.

By completing this capstone, you have gained **hands-on experience in full-stack web development!** 🎉🔥

ISDM-NXT

Assignment Solution & Step-by-Step Guide: Implement Payment Gateway in a Web App

Objective

In this assignment, we will integrate a **payment gateway (Razorpay or Stripe)** into a **MERN-based web application**. The application will:

- ✅ Allow users to make payments securely.
- ✅ Generate payment orders dynamically.
- ✅ Process transactions and confirm successful payments.

Step 1: Setting Up the Backend (Node.js & Express.js)

1.1 Initialize a Node.js Project

1. Open a terminal and create a new project folder:

sh

CopyEdit

mkdir payment-app

cd payment-app

2. Initialize a Node.js project:

sh

CopyEdit

npm init -y

This creates a package.json file for managing dependencies.

1.2 Install Required Dependencies

ISDM-NXT

sh

CopyEdit

npm install express dotenv cors body-parser razorpay stripe

- **express** – Backend framework to handle API requests.
- **dotenv** – Manages environment variables securely.
- **cors** – Allows frontend and backend communication.
- **body-parser** – Parses incoming JSON requests.
- **razorpay/stripe** – SDKs for handling payments.

1.3 Create .env File for API Keys

Create a .env file in the backend folder:

env

CopyEdit

RAZORPAY_KEY_ID=your_razorpay_key_id

RAZORPAY_KEY_SECRET=your_razorpay_key_secret

STRIPE_SECRET_KEY=your_stripe_secret_key

PORT=5000

1.4 Set Up Express Server

Create server.js and set up the server:

js

CopyEdit

require("dotenv").config();

const express = require("express");

ISDM-NXT

```
const cors = require("cors");


const app = express();
app.use(cors());
app.use(express.json());


app.get("/", (req, res) => {
   res.send("Payment Gateway Integration API is Running...");
});


const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

✅ **The backend is now set up and ready to handle payment requests.**

Step 2: Integrating Razorpay Payment Gateway

2.1 Create Payment Route for Razorpay

Create a routes/paymentRoutes.js file and add:

js

CopyEdit

```
const express = require("express");
const Razorpay = require("razorpay");
```

```
require("dotenv").config();


const router = express.Router();


const razorpay = new Razorpay({
  key_id: process.env.RAZORPAY_KEY_ID,
  key_secret: process.env.RAZORPAY_KEY_SECRET,
});


// Create a Payment Order
router.post("/create-order", async (req, res) => {
  try {
    const options = {
      amount: req.body.amount * 100, // Convert amount to paisa
      currency: "INR",
      receipt: `receipt_${Date.now()}`,
      payment_capture: 1,
    };


    const order = await razorpay.orders.create(options);
    res.status(200).json({ success: true, order });
  } catch (error) {
```

ISDM-NXT

```
    res.status(500).json({ success: false, message: error.message });
  }
});

module.exports = router;
```

✅ **This API will generate an order when the frontend requests payment.**

2.2 Integrate the Payment Route in server.js

Modify server.js to use the payment route:

```
js
CopyEdit
const paymentRoutes = require("./routes/paymentRoutes");
app.use("/api/payment", paymentRoutes);
```

✅ **Now, the backend is ready to create payment orders using Razorpay.**

Step 3: Setting Up the Frontend with React.js

3.1 Create a React App

```
sh
CopyEdit
npx create-react-app payment-frontend
cd payment-frontend
```

3.2 Install Required Dependencies

sh

CopyEdit

npm install axios

- **axios** – Used for API requests.

3.3 Create a Razorpay Payment Component

Inside src/components/, create Payment.js:

jsx

CopyEdit

```
import React from "react";
import axios from "axios";

function Payment() {
  const initiatePayment = async () => {
    try {
      const { data } = await
axios.post("http://localhost:5000/api/payment/create-order", {
        amount: 500, // Amount in INR
      });

      const options = {
        key: "your_razorpay_key_id",
        amount: data.order.amount,
        currency: "INR",
```

ISDM-NXT

```
            name: "Your Store",
            description: "Test Payment",
            order_id: data.order.id,
            handler: function (response) {
              alert("Payment Successful! Payment ID: " +
response.razorpay_payment_id);
            },
            prefill: {
              name: "John Doe",
              email: "johndoe@example.com",
              contact: "9876543210",
            },
            theme: {
              color: "#3399cc",
            },
          };

          const rzp = new window.Razorpay(options);
          rzp.open();
        } catch (error) {
          console.error("Error initiating payment", error);
        }
      };
```

ISDM-NXT

```
  return (
    <div>
      <h2>Make a Payment</h2>
      <button onClick={initiatePayment}>Pay ₹500</button>
    </div>
  );
}

export default Payment;
```

✅ **When the "Pay ₹500" button is clicked, Razorpay's checkout UI will open.**

Step 4: Adding Stripe Payment Gateway (Alternative Method)

4.1 Create Payment Route for Stripe in Backend

Modify paymentRoutes.js to include Stripe:

```
js
CopyEdit
const stripe = require("stripe")(process.env.STRIPE_SECRET_KEY);

// Create Stripe Payment Session
router.post("/stripe-checkout", async (req, res) => {
  try {
```

```
    const session = await stripe.checkout.sessions.create({
      payment_method_types: ["card"],
      line_items: [
        {
          price_data: {
            currency: "usd",
            product_data: { name: "Sample Product" },
            unit_amount: 2000,
          },
          quantity: 1,
        },
      ],
      mode: "payment",
      success_url: "http://localhost:3000/success",
      cancel_url: "http://localhost:3000/cancel",
    });

    res.json({ id: session.id });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

ISDM-NXT

✅ **This API will generate a payment session for Stripe.**

4.2 Implement Stripe Checkout in React

Install Stripe SDK:

sh

CopyEdit

```
npm install @stripe/stripe-js
```

Modify Payment.js to handle Stripe payments:

jsx

CopyEdit

```
import React from "react";
import { loadStripe } from "@stripe/stripe-js";
import axios from "axios";

const stripePromise = loadStripe("your_stripe_public_key");

function Payment() {
  const handlePayment = async () => {
    const { data } = await
axios.post("http://localhost:5000/api/payment/stripe-checkout");
    const stripe = await stripePromise;
    stripe.redirectToCheckout({ sessionId: data.id });
  };
```

ISDM-NXT

```
  return <button onClick={handlePayment}>Pay with Stripe</button>;
}

export default Payment;
```

✅ **Clicking "Pay with Stripe" redirects the user to Stripe's secure checkout page.**

Step 5: Deploying the Payment Web App

5.1 Deploy Backend to Heroku

1. Install Heroku CLI:

```
sh
CopyEdit
npm install -g heroku
```

2. Initialize Git and push the code to Heroku:

```
sh
CopyEdit
git init
git add .
git commit -m "Initial commit"
heroku create payment-app
git push heroku master
```

✅ **Your backend is now live on Heroku.**

ISDM-NXT

5.2 Deploy Frontend to Netlify

1. Build the React app:

sh

CopyEdit

npm run build

2. Upload the build/ folder to Netlify.
   ✅ **Your frontend is now live on Netlify.**

CONCLUSION

✅ **Integrated Razorpay and Stripe into a MERN-based web app.**
✅ **Implemented backend API for payment handling.**
✅ **Built a frontend React component to initiate payments.**
✅ **Deployed the full application to Heroku and Netlify.**

🚀 **Next Steps:**

- Store **transaction history** in MongoDB.
- Implement **email receipts** after successful payments.
- Secure payments using **JWT authentication**.

By following this guide, you can now **integrate payment gateways into any web application!** 🎉 🔥