**Requirements Document for Dynamic Website Conversion**

**Project Overview**

**This project aims to convert static web pages (index.html and homepage.html) into dynamic, database-driven pages using .NET for the backend and React for the frontend. The new system will allow content to be managed dynamically through a robust API and user-friendly React interface, deployed as a containerized application on Azure Kubernetes Service (AKS) with routing through AWS Route53.**

**Functional Requirements**

**1. Content Management**

* **Navigation Menu:**
  + **Dynamically manage navigation links and their URLs via a .NET-based API.**
  + **Display navigation links on all pages through React components.**
* **Sections and Content Blocks:**
  + **Define sections for blogs, services, and other content dynamically via the API.**
  + **Include a title, description, and optional image for each section.**
  + **Allow ordering of sections.**
* **Media Management:**
  + **Upload and manage images and other media files using the API.**
  + **Render media dynamically in React components.**

**2. Dynamic Page Rendering**

* **Homepage:**
  + **Display a dynamic navigation bar based on API data.**
  + **Render content sections such as blogs, services, and portfolio items dynamically using React.**
* **Additional Pages:**
  + **Enable creation of additional pages (e.g., contact, about, services) via API endpoints.**

**3. Deployment and Containerization**

* **Containerization:**
  + **Containerize the .NET backend and React frontend applications using Docker.**
  + **Include Dockerfile and docker-compose.yaml for local development and production deployment.**
* **Deployment on AKS:**
  + **Deploy the containerized applications on Azure Kubernetes Service (AKS).**
  + **Use Helm charts for Kubernetes resource management.**
  + **Ensure autoscaling and high availability.**

**4. DNS and Routing**

* **Domain Management:** 
  + **Route DNS for www.stshadow.com managed by AWS Route53 to Azure-hosted services.**
  + **Configure appropriate DNS records (e.g., A, CNAME) and SSL certificates.**

**5. User Interface and Experience**

* **Use React for a responsive and interactive frontend experience.**
* **Reuse existing static assets (CSS, JavaScript, images) and integrate them with React components.**
* **Ensure compatibility with desktop, tablet, and mobile devices.**

**6. API Design**

* **Develop RESTful APIs using .NET to:** 
  + **Manage navigation menus.**
  + **Handle section content creation, updates, and retrieval.**
  + **Upload and manage media files.**

**Non-Functional Requirements**

**1. Performance**

* **Optimize API endpoints to ensure low latency.**
* **Use caching for frequently accessed resources.**

**2. Security**

* **Implement user authentication and authorization for API access.**
* **Sanitize inputs to prevent SQL injection and cross-site scripting (XSS).**
* **Serve the site over HTTPS in production.**

**3. Maintainability**

* **Use a modular architecture for both .NET and React components.**
* **Write clear and reusable code with comments and documentation.**
* **Use version control (e.g., Git) for tracking changes.**

**4. Compatibility**

* **Ensure compatibility with modern browsers (Chrome, Firefox, Edge, Safari).**
* **Support .NET 6+ and the latest version of React.**

**System Architecture**

**1. Database**

* **Use SQL Server for production and SQLite for development.**
* **Tables:** 
  + **NavigationMenu: Stores navigation link names and URLs.**
  + **SectionContent: Stores titles, descriptions, and associated media.**
  + **MediaAssets: Stores file paths for uploaded images.**

**2. Backend (.NET)**

* **Develop a RESTful API using ASP.NET Core.**
* **Handle data storage, validation, and business logic.**

**3. Frontend (React)**

* **Use React for the user interface with state management (e.g., Redux or Context API).**
* **Fetch data from the .NET API dynamically.**

**4. Static and Media Files**

* **Organize static assets (CSS, JS, images) under the public/ directory in React.**
* **Serve uploaded media files from a designated Azure Blob Storage or API endpoint.**

**5. Containerization and Deployment**

* **Docker:**
  + **Create separate Dockerfile files for .NET and React applications.**
  + **Include docker-compose.yaml for multi-container setups.**
* **Azure Kubernetes Service:**
  + **Deploy the containerized applications using AKS.**
  + **Utilize Helm charts for Kubernetes configuration.**
  + **Enable autoscaling and load balancing for high availability.**
* **DNS Configuration:**
  + **Use AWS Route53 to route www.stshadow.com to Azure services.**
  + **Configure secure HTTPS routing with SSL/TLS certificates.**

**Development Plan**

**1. Environment Setup**

* **Install .NET SDK, Node.js, and Docker.**
* **Set up a development environment for .NET and React.**
* **Initialize new .NET and React projects.**

**2. Backend Development**

* **Define models for NavigationMenu, SectionContent, and MediaAssets.**
* **Implement API endpoints for managing content.**
* **Add authentication and authorization middleware.**

**3. Frontend Development**

* **Set up React with state management and routing.**
* **Create components for navigation menus, sections, and media.**
* **Integrate API calls to fetch and render data dynamically.**

**4. Containerization**

* **Write Dockerfile files for .NET and React applications.**
* **Create docker-compose.yaml for local and production configurations.**
* **Test the containers locally.**

**5. Deployment**

* **Deploy the containerized applications to AKS using Helm charts.**
* **Configure DNS routing through AWS Route53.**
* **Set up HTTPS certificates.**

**6. Testing**

* **Test API endpoints and React components locally.**
* **Perform integration testing on AKS.**
* **Write unit tests for critical backend and frontend functionalities.**

**Deliverables**

1. **Fully functional dynamic website:**
   * **Dynamically rendered React frontend with data fetched from .NET APIs.**
   * **Additional pages managed via the API.**
2. **Containerization:**
   * **Dockerfile for .NET backend and React frontend.**
   * **Docker Compose configuration for multi-container environments.**
3. **Deployment:**
   * **Applications deployed on Azure Kubernetes Service.**
   * **DNS routing for www.stshadow.com through AWS Route53.**
   * **Secure HTTPS configuration.**
4. **Documentation:**
   * **Installation and deployment guide.**
   * **API documentation.**
   * **Admin user manual.**
5. **Source code repository with version control.**

**Timeline**

| **Phase** | **Estimated Duration** |
| --- | --- |
| **Environment Setup** | **2 days** |
| **Backend Development** | **5 days** |
| **Frontend Development** | **5 days** |
| **Containerization** | **3 days** |
| **Deployment** | **3 days** |
| **Testing** | **2 days** |
| **Total** | **20 days** |

**Assumptions and Constraints**

* **Existing assets (images, CSS, JS) are available and functional.**
* **Development will follow best practices for .NET, React, and Docker.**
* **Deployment will use Azure services with routing through AWS Route53.**

**Risks and Mitigation**

| **Risk** | **Mitigation Strategy** |
| --- | --- |
| **Inconsistent content formatting** | **Use backend validation and React error handling.** |
| **Deployment issues** | **Test deployment in a staging environment.** |
| **Performance bottlenecks** | **Optimize API queries and use React lazy loading.** |
| **Compatibility problems** | **Test on multiple browsers and devices.** |
| **DNS misconfiguration** | **Double-check DNS records and SSL setup.** |

**Conclusion**

**This project will transform the static website into a modern, dynamic web application. Leveraging .NET for the backend and React for the frontend ensures scalability, maintainability, and an enhanced user experience. Containerization and deployment on AKS provide high availability, while AWS Route53 ensures proper routing for the domain www.stshadow.com. The outlined requirements and plan provide a roadmap for successful implementation.**