Detailed Architecture and Code for MVP Development Using Microsoft Technology Platform

1. Architecture Overview

For the MVP of the pet social network, we will utilize the Microsoft Azure cloud platform, leveraging its wide array of services to build a scalable, secure, and performant application. The architecture will follow a microservices design pattern, ensuring that different parts of the application can be developed, deployed, and scaled independently. This setup will also facilitate the integration of AI/ML capabilities into the platform.

Cloud Platform: Microsoft Azure

- **Compute:** Azure App Services for hosting web apps and APIs, Azure Functions for serverless computing, and Azure Kubernetes Service (AKS) for container orchestration.
- **Storage:** Azure Blob Storage for media files, Azure SQL Database for relational data, and Azure Cosmos DB for non-relational data (e.g., JSON).
- **Database:** Azure SQL Database for structured data, Redis Cache for fast data retrieval.
- **AI/ML Services:** Azure Machine Learning for building, training, and deploying ML models.
- **Authentication:** Azure AD B2C for managing user authentication and identity.
- **APIs:** Azure API Management for secure API exposure.
- **Front-End Framework:** React.js for UI development, with Azure Static Web Apps for hosting.
- **Back-End Framework:** .NET Core with ASP.NET Core Web API for building RESTful services.
- **DevOps:** Azure DevOps for CI/CD pipeline and monitoring.

2. Detailed Architecture Components

1. **User Profiles for Pets and Owners**
    - **Backend Microservice**: User Service
        - *Functions*: Manages CRUD operations for user and pet profiles.
        - *Technology*: .NET Core with ASP.NET Core Web API, Azure SQL Database for persistent storage.
        - *Endpoints*:
            - POST /api/users: Create a new user profile.
            - POST /api/pets: Create a new pet profile associated with a user.
            - GET /api/users/{userId}: Retrieve user profile details.
            - GET /api/pets/{petId}: Retrieve pet profile details.

        - PUT /api/users/{userId}: Update user profile.
        - PUT /api/pets/{petId}: Update pet profile.
        - DELETE /api/users/{userId}: Delete user profile.
        - DELETE /api/pets/{petId}: Delete pet profile.
    - **Database Schema**:

sql

```
CREATE TABLE Users (
   UserId INT PRIMARY KEY IDENTITY(1,1),
   Name NVARCHAR(100),
   Email NVARCHAR(100) UNIQUE,
   PasswordHash NVARCHAR(255),
   ProfilePictureUrl NVARCHAR(255),
   CreatedAt DATETIME DEFAULT GETDATE(),
   UpdatedAt DATETIME
);

CREATE TABLE Pets (
   PetId INT PRIMARY KEY IDENTITY(1,1),
   UserId INT FOREIGN KEY REFERENCES Users(UserId),
   Name NVARCHAR(100),
   Breed NVARCHAR(100),
   Age INT,
   Bio NVARCHAR(MAX),
   ProfilePictureUrl NVARCHAR(255),
   HealthRecords NVARCHAR(MAX),  -- Stored as JSON
   CreatedAt DATETIME DEFAULT GETDATE(),
   UpdatedAt DATETIME
);
```

    - **AI Integration**:

        - **ML Model**: Predictive health analytics for pets.
        - **Implementation**:
            - Use Azure Machine Learning to train and deploy a model predicting potential health issues based on breed, age, and health history.
            - Serve the model via Azure Functions:

csharp

```
[FunctionName("PredictPetHealth")]
public async Task<IActionResult> Run(
   [HttpTrigger(AuthorizationLevel.Function, "post", Route = null)] HttpRequest req,
   ILogger log)
{
   string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
   var data = JsonConvert.DeserializeObject<PetData>(requestBody);

   var prediction = await PredictHealthRisk(data);

   return new OkObjectResult(new { HealthRisk = prediction });
}
```

2. **Advanced Pet Profiles**
    - **Backend Microservice**: Pet Health Service
        - *Functions*: Manages health records, vaccination details, and breed certifications.
        - *Technology*: .NET Core with ASP.NET Core Web API, Azure Cosmos DB for flexible data models, Redis Cache for frequent data retrieval.
        - *Endpoints*:
            - POST /api/pets/{petId}/health: Add health records for a pet.
            - GET /api/pets/{petId}/health: Retrieve health records.
            - PUT /api/pets/{petId}/health: Update health records.
            - DELETE /api/pets/{petId}/health: Delete health records.
    - **Database Schema**:

sql

```
CREATE TABLE PetHealthRecords (
  RecordId INT PRIMARY KEY IDENTITY(1,1),
  PetId INT FOREIGN KEY REFERENCES Pets(PetId),
  VaccinationDetails NVARCHAR(MAX),  -- Stored as JSON
  Allergies NVARCHAR(MAX),  -- Stored as JSON
  MedicalHistory NVARCHAR(MAX),
  CreatedAt DATETIME DEFAULT GETDATE(),
  UpdatedAt DATETIME
);
```

- **AI Integration**:
    - **ML Model**: Recommendations for personalized care.
    - **Implementation**:
        - Deploy models via Azure ML and integrate into the back-end service using Azure Functions.
        - Fetch real-time data via APIs for enhanced predictions.

3. **Front-End Development**
    - **UI/UX Design**:
        - *Technology*: React.js for the user interface, hosted on Azure Static Web Apps.
        - *Components*:
            - **Profile Page**: User and pet profile management with real-time updates.
            - **Health Dashboard**: Visual representation of health records and alerts for upcoming vaccinations.
    - **State Management**:
        - *Technology*: Redux for state management.
        - *Sample Redux State*:

javascript

```
const initialState = {
  user: {},
  pets: [],
  healthRecords: {}
};

function rootReducer(state = initialState, action) {
  switch (action.type) {
    case 'SET_USER':
      return { ...state, user: action.payload };
    case 'ADD_PET':
      return { ...state, pets: [...state.pets, action.payload] };
    case 'SET_HEALTH_RECORDS':
      return { ...state, healthRecords: action.payload };
    default:
      return state;
  }
}
```

- **API Integration**:
  - Use Axios for calling REST APIs:

javascript

```
axios.post('/api/users', userData)
   .then(response => console.log('User created:', response.data))
   .catch(error => console.error('Error creating user:', error));
```

4. **DevOps and CI/CD Pipeline**
   - **Continuous Integration**: Managed through Azure DevOps, ensuring code quality and automated testing.
   - **Continuous Deployment**: Pipeline setup in Azure DevOps for deploying services to Azure App Services and AKS.
   - **Monitoring and Logging**: Azure Monitor and Azure Application Insights for real-time performance tracking and diagnostics.

- **Sample CI/CD Pipeline Configuration**:

yaml

```yaml
trigger:
  branches:
    include:
      - main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: UseDotNet@2
  inputs:
    packageType: 'sdk'
    version: '6.x.x'

- script: |
    dotnet build --configuration Release
    dotnet test --configuration Release
  displayName: 'Build and Test'

- task: AzureWebApp@1
  inputs:
    azureSubscription: 'Your Azure Subscription'
    appName: 'your-app-name'
    package: '$(System.DefaultWorkingDirectory)/**/*.zip'
  displayName: 'Deploy to Azure WebApp'
```

Conclusion

This detailed architecture leverages Microsoft Azure's robust cloud services, combining .NET Core for back-end development with Azure's AI/ML capabilities, providing a scalable and secure solution. The technical architecture supports future expansion, ensuring that the platform is prepared for growth, while also being efficient and cost-effective for the MVP phase.