Indiana University Southeast

RS-9

Jonathan Roberts

Sebastian Burman

Xiaokun Li

Capstone No Kill Louisville SMS Check-In System

Professor Finkbine

03/23/2024

**Programmer Manual for No-Kill Louisville Digital Check-In System**

**1. Vision Statement**

Our vision is to provide No-Kill Louisville with a digital check-in system that epitomizes efficiency, security, and user-friendliness. By automating the pet food pickup process, we aim to facilitate a smooth operation that saves time for both clients and volunteers, ultimately allowing the shelter to serve the community and its animals better.

**2. Introduction**

Welcome to the No-Kill Louisville digital check-in system programmer's manual. This document is designed to be a definitive technical guide for software developers tasked with the system's development, maintenance, or future enhancements.

**3. Component Overview**

The system comprises several interconnected components, each responsible for a distinct set of functionalities within the overall architecture.

a. SMS Reception Component

Description:  
Interacts with Twilio to receive and process SMS check-ins. It decodes messages, validates client input, and triggers check-in procedures.

Functionality:

* Receives SMS through Twilio webhooks.
* Validates SMS format against expected patterns.
* Parses out check-in information such as client ID and time of arrival.
* Logs invalid attempts and alerts administrators as necessary.

Key Methods:

* ReceiveMessage(): Captures incoming messages.
* ValidateFormat(string message): Checks the message against a regex pattern.
* ParseCheckInDetails(string message): Extracts and returns structured check-in data.

Technologies Used:

* .NET Core for the API endpoint.
* Twilio API for SMS reception.

Interfaces:

* RESTful endpoints for webhook reception.
* Database connection for logging attempts.

Data Handling:

* Input validation to prevent injection attacks.
* Encrypted storage of sensitive information.

**4. Tool Overview**

Each tool and language selected contributes to the robustness and maintainability of the system.

a. .NET Core

Why .NET Core:

* Cross-platform support, crucial for a system that may be deployed on various OSes.
* High performance for handling concurrent requests efficiently.
* Strong security features for building a system that handles sensitive information.

Key Features:

* Asynchronous programming patterns.
* Dependency injection for easier testing and maintenance.
* Entity Framework Core for ORM.

Best Practices:

* Utilize async/await for I/O-bound code.
* Keep business logic separate from controllers for a clean MVC architecture.
* Regularly update dependencies to maintain security and performance enhancements.

**5. Project Repository**

The project's repository is structured for ease of access and clarity.

a. Software

* Directory Structure:
  + /src: Source code for the system.
  + /src/WebAPI: The .NET Core WebAPI project files.
  + /src/ClientApp: The front-end client application.
* Branching:
  + main: Stable release branch.
  + development: Active development branch.
  + Feature branches are named according to the feature (feature/sms-processing).

b. Test Cases

* Located in /tests.
* Unit tests are structured by the component (/tests/WebAPIUnitTests).
* Integration tests (/tests/WebApiIntegrationTests).

c. Documentation

* Located in /docs.
* Includes this programmer manual (/docs/programmer\_manual.md).
* API documentation (/docs/api\_documentation.md).

d. Test Platform Description

* Found in /docs/test\_platform.md.
* Includes descriptions of test environments, setups, and tools.

e. Test Scripts

* Automated test scripts are stored in /tests/scripts.
* Contains scripts for running unit and integration tests.

**6. Installation for New Install**

Detailed steps are documented in /docs/installation\_guide.md.

* Prerequisites: .NET Core Runtime, SQL server, appropriate environment variables.
* Clone the repository.
* Execute dotnet restore to install dependencies.
* Apply database migrations using dotnet ef database update.
* Start the application with dotnet run.

**7. Installation for New Platform (preserves data from previous)**

Transitioning to a new platform while preserving data requires careful migration.

* Ensure the target platform supports .NET Core.
* Backup the current database and config files.
* Install the software on the new platform as per the installation guide.
* Restore the database and config files.
* Perform thorough testing to ensure system integrity.

**8. Further Development Statement**

With more time, enhancements would include:

* Developing a mobile application for enhanced accessibility.
* Implementing machine learning to predict peak times and manage resources.
* Integrating with more scheduling systems to broaden our reach.

This manual should empower you to understand the system's intricacies and empower you to develop, deploy, and enhance the system confidently.