Indiana University Southeast

RF-4

Jonathan Roberts

Sebastian Burman

Xiaokun Li

Capstone No Kill Louisville SMS Check-In System

Professor Finkbine

11/24/2023

### Software Design for Digital Check-In Service at No-Kill Louisville Animal Shelter

### Data Design

1. Client Information

* **Structure:** Client
  + **Attributes:**
    - ClientID (String): A unique identifier for each client.
    - Name (String): The client's full name.
    - ContactNumber (String): The client's contact number.
  + **Example:**

public class Client  
{  
 public string ClientID { get; set; }  
 public string Name { get; set; }  
 public string ContactNumber { get; set; }  
}

2. Scheduled Pickups

* **Structure:** Appointment
  + **Attributes:**
    - AppointmentID (String): Unique identifier for the pickup.
    - ClientID (String): Linked to the Client's ID.
    - ScheduledDateTime (DateTime): The scheduled date and time for the pickup.
    - PickupItems (List<String>): List of items to be picked up.
  + **Example:**

public class Appointment  
{  
 public string AppointmentID { get; set; }  
 public string ClientID { get; set; }  
 public DateTime ScheduledDateTime { get; set; }  
 public List<string> PickupItems { get; set; }  
}

3. SMS Messages

* **Structure:** SMSMessage
  + **Attributes:**
    - MessageID (String): A unique identifier for the message.
    - FromNumber (String): The sender's phone number.
    - Content (String): The actual text content of the SMS.
    - ReceivedAt (DateTime): The timestamp when the message was received.
  + **Example:**

public class SMSMessage  
{  
 public string MessageID { get; set; }  
 public string FromNumber { get; set; }  
 public string Content { get; set; }  
 public DateTime ReceivedAt { get; set; }  
}

4. Notifications

* **Structure:** Notification
  + **Attributes:**
    - NotificationID (String): Unique identifier for the notification.
    - Message (String): The content of the notification.
    - CreatedOn (DateTime): Timestamp when the notification was created.
  + **Example:**

public class Notification  
{  
 public string NotificationID { get; set; }  
 public string Message { get; set; }  
 public DateTime CreatedOn { get; set; }  
}

Data Relationships

* **Client to Scheduled Pickups:** One-to-Many relationship. Each client can have multiple pickups scheduled.
* **SMS Messages:** Not directly linked to Client or ScheduledPickup but processed to extract relevant information.
* **Notifications:** Linked to ScheduledPickup and possibly Client, indicating the status of the pickup process.

Data Storage and Retrieval

* **Database:** A relational database like PostgreSQL or a NoSQL database like MongoDB can be used.
* **ORM (Object-Relational Mapping):** Entity Framework for .NET Core for data manipulation and retrieval.
* **Data Access Layer:** Implement repositories and services to abstract and handle database interactions.

### Security and Performance Considerations

* **Data Security:** Encrypt sensitive data in the database and use secure connections for data access.
* **Scalability:** Design the database schema and queries for efficiency, especially under high load.
* **Backup and Recovery:** Implement regular backups and a strategy for data recovery.

A screenshot of a computer

Description automatically generated

### 

### Architecture Design

#### 1. System Overview

* **Components:**
  + **SMS Reception Component:** Handles incoming SMS messages via Twilio.
  + **API Communication Component:** Interacts with the Acuity scheduling system.
  + **Notification Dispatch Component:** Manages notifications to shelter volunteers.
  + **Database:** Stores client information, scheduled pickups, and other relevant data.
  + **WebAPI Server:** The central hub for processing, logic implementation, and API interactions.

#### 2. Data Flow

* **SMS Reception to API Communication:**
  + SMS messages received by Twilio are forwarded to the WebAPI server.
  + The server processes and validates the SMS content, extracting necessary information like ClientID and ScheduledTime.
  + Validated data is then passed to the API Communication Component.
* **API Communication to Database Interaction:**
  + The API Communication Component interacts with the Acuity scheduling system, confirming pickup details.
  + Responses from Acuity, along with any processed data, are stored in the database for record-keeping and further action.
* **Database to Notification Dispatch:**
  + Once a pickup is confirmed, the Notification Dispatch Component retrieves relevant information from the database.
  + It then formats and sends notifications to the shelter volunteers via chosen channels (email, SMS, dashboard).

#### 3. Component Details

* **SMS Reception Component:**
  + Interfaces with Twilio through webhooks.
  + Responsible for initial processing of SMS messages.
* **API Communication Component:**
  + Manages all interactions with the Acuity scheduling system's API.
  + Responsible for sending requests and handling responses.
* **Notification Dispatch Component:**
  + Gathers necessary information for notifying volunteers.
  + Uses various channels (e.g., email, SMS) to send notifications.
* **Database:**
  + Stores all persistent data such as client information, scheduled pickups, received messages, and notifications.
* **WebAPI Server:**
  + Hosts the backend logic.
  + Serves as the intermediary for all component interactions.
  + Ensures secure and efficient data handling and processing.

#### 4. Security and Reliability

* **Secure API Endpoints:** Implement authentication and encryption for all API endpoints.
* **Data Encryption:** Ensure that sensitive data, both in transit and at rest, is encrypted.
* **Error Handling:** Robust error handling mechanisms to gracefully manage exceptions and maintain system stability.
* **Scalability:** Design components to handle varying loads, ensuring the system remains responsive and efficient.

#### 5. Deployment Strategy

* **Cloud Hosting:** Utilize cloud services like AWS or Azure for hosting the WebAPI server and database.
* **Containerization:** Use Docker for containerizing the application, ensuring easy deployment and scalability.
* **Continuous Integration/Continuous Deployment (CI/CD):** Implement CI/CD pipelines for streamlined testing and deployment.

A diagram of a diagram

Description automatically generated

#### 

### Interface Design

#### Internal Interfaces

##### SMS to API Gateway Interface

* **Protocol:** RESTful API over HTTP.
* **Data Format:** JSON.
* **Process Flow:**
  + Twilio receives an SMS and triggers a webhook.
  + The webhook sends a POST request to the designated endpoint on the .NET Core WebAPI server.
  + The request body contains JSON data with key details from the SMS, such as the sender's number and message content.
* **Security:**
  + Use HTTPS to secure data in transit.
  + Implement validation to ensure only Twilio requests are processed.
* **Example JSON Structure:**
* {  
   "From": "+15551234567",  
   "Body": "Check-in message content",  
   "ReceivedAt": "2023-11-24T12:00:00Z"  
  }

##### API to Notification System Interface

* **Protocol:** Internal method call within the .NET Core application.
* **Data Transfer:** In-memory data objects.
* **Process Flow:**
  + The .NET Core application processes the request, extracts, and validates data.
  + It then invokes a method to the Notification Dispatch Module with the processed information.
* **Security:** As this is an internal call, standard application security practices apply.
* **Data Object Example:**
* public class ClientCheckInInfo  
  {  
   public string ClientID { get; set; }  
   public DateTime ScheduledTime { get; set; }  
  }

#### External Interfaces

##### Client SMS Interface

* **Protocol for Clients:** Clients send an SMS to a designated Twilio phone number.
* **SMS Format:** A predefined format, such as: “Check-in [ClientID] at [HH:MM]”.
* **Validation:** The .NET Core application validates the format and extracts necessary information.
* **Error Handling:** If the SMS format is incorrect, send an automated reply via Twilio informing the client of the correct format.

##### Volunteer Notification Interface

* **Method of Notification:** Notifications can be sent via email, SMS, or a web-based dashboard.
* **Notification Content:** Information about the client's arrival and pickup schedule.
* **Format:** Structured text with key details, e.g., “Client [ClientID] has arrived for their scheduled pickup at [ScheduledTime].”
* **Delivery Mechanism:** For emails/SMS, use an SMTP server or Twilio's SMS service. For dashboard notifications, use real-time technologies like SignalR for .NET Core to push updates to the dashboard.

### Security and Reliability

* **Authentication and Authorization:** Implement proper authentication for the WebAPI endpoints, possibly using OAuth or API keys.
* **Data Encryption:** Ensure all data in transit and at rest is encrypted.
* **Error Handling:** Robust error handling in the .NET Core application to manage exceptions and provide meaningful feedback to both the Twilio webhook and the dashboard.
* **Logging and Monitoring:** Implement logging to monitor the health and performance of the application and interfaces.

### Procedural Design

#### 1. SMS Reception and Processing

* **Receive SMS:**
  + **Description:** The Twilio service receives an SMS from a client and triggers a webhook to the .NET Core WebAPI server.
  + **Endpoint:** POST /api/sms/receive
  + **Request Body:** Contains the sender's number and message content in JSON format.
* **Validate and Parse SMS:**
  + **Function:** ParseAndValidateSMS(Json smsData)
  + **Process:**
    - Validate the format of the SMS.
    - Extract relevant information such as ClientID and ScheduledTime.
  + **Error Handling:** If the format is incorrect, send an automated response via Twilio informing the client of the correct format.
* **Send to API Gateway:**
  + **Function:** SendToApiGateway(ClientCheckInInfo checkInInfo)
  + **Process:**
    - Formats the data into an API request for the Acuity scheduling system.
    - Handles any API authentication required.

#### 2. API Request Handling

* **Receive and Process API Request:**
  + **Function:** ProcessApiRequest(ClientCheckInInfo checkInInfo)
  + **Process:**
    - Send a request to the Acuity API to confirm the client's scheduled pickup.
    - Handle the API response, including success or error messages.
* **Log and Monitor:**
  + **Function:** LogApiRequestAndResponse(ApiRequest request, ApiResponse response)
  + **Purpose:** Logs all requests and responses for monitoring and debugging purposes.

#### 3. Notification Dispatch

* **Prepare Notification:**
  + **Function:** PrepareVolunteerNotification(ApiResponse apiResponse)
  + **Process:**
    - Based on the response from the Acuity API, prepare a notification message for the shelter volunteers.
* **Dispatch Notification:**
  + **Function:** DispatchNotification(Notification notification)
  + **Channels:**
    - Email, SMS (via Twilio), or a web-based real-time dashboard (using technologies like SignalR).
  + **Process:**
    - Sends the prepared notification through the selected channels.

#### Pseudo-code Example: SMS Processing Procedure

FUNCTION ProcessIncomingSMS(Json smsData)  
 VALIDATED\_DATA = ParseAndValidateSMS(smsData)  
 IF VALIDATED\_DATA IS NOT NULL THEN  
 SendToApiGateway(VALIDATED\_DATA)  
 ELSE  
 SendErrorResponse(smsData.From)  
 ENDIF  
END FUNCTION  
  
FUNCTION ParseAndValidateSMS(Json smsData)  
 // Implement format checks and data extraction  
 // RETURN extracted data or NULL if invalid  
END FUNCTION  
  
FUNCTION SendToApiGateway(ClientCheckInInfo checkInInfo)  
 // Format and send request to Acuity API  
 // Log request and handle response  
END FUNCTION

### Design Considerations

* **Error Handling and Logging:** Implement robust error handling to gracefully manage and log any unexpected issues.
* **Scalability:** Design procedures to efficiently handle varying loads, ensuring performance under high-volume scenarios.
* **Security:** Incorporate security best practices, especially in data handling and API interactions.
* **Maintainability:** Write clear, well-commented code to ensure maintainability, especially important for a team of junior programmers.

A screenshot of a computer

Description automatically generated

**Although all of us have involved in designing every aspect of this system, but we do have individual areas to focus on detailed below.**

**Individual Contributions**

**Xiaokun Li:**

● **System Integration:** Li will primarily focus on integrating the check-in system with the Acuity scheduling system using the API. This will involve thorough understanding of API functionalities, data mapping, and ensuring seamless data transfer between the systems.

● **Data Management:** Li will be responsible for overseeing the data storage and data flow aspects, ensuring that the data from the check-in system is accurately stored, updated, and synchronized with the shelter's database.

● **Quality Assurance:** Li will play a critical role in testing the integrated system, identifying potential issues, and ensuring that the check-in system meets the highest quality standards before its deployment.

**Jonathan Roberts:**

● **Twilio Integration**: Jonathan will focus on integrating the Twilio text message service into the check-in system, enabling the system to send real-time notifications and updates to users. This will involve setting up the necessary communication protocols and ensuring the smooth functioning of the messaging service.

● **User Experience:** Jonathan will be responsible for designing an intuitive and user-friendly interface for the check-in system, ensuring that users can easily navigate through the check-in process and receive clear and concise notifications via the Twilio service.

● **Documentation:** Jonathan will oversee the documentation process, ensuring that all system functionalities, user guides, and technical documentation are well-documented and readily available for future reference and troubleshooting.

**Sebastian Burman:**

● **.NET Core Developer:** Sebastian will focus on developing the core functionalities of the check-in system using .NET Core Web API. This will involve designing and implementing the necessary backend processes, data handling functionalities, and communication protocols for the system.

● **Technical Lead:** Sebastian will be the main point of contact for resolving any technical issues or concerns related to the check-in system. This will involve providing timely support to team members, addressing any technical challenges, and ensuring the smooth operation of the system throughout its lifecycle.

● **Project Coordinator:** Sebastian will take on the responsibility of coordinating tasks, managing timelines, and ensuring effective communication between team members. Sebastian will also be responsible for tracking project milestones and ensuring that the project progresses according to the established timeline.

# Team contact

1. **Team lead:** Sebastian Burman
   * Contact: sjburman@iu.edu
   * Responsibilities: Overall project coordination
2. **Developer:** Sebastian Burman, Xiaokun Li, Jonathan Roberts
   * Contact: sjburman@iu.edu
   * Responsibilities: Backend development