Software Design Document for Animal Care Utility App

Version 2.0 approved

Prepared by Argentieri, Marco
Gudino, Francisco
Guizarnotegui, Joshua
Hanson, Sean
Placencia Jr., Juan

Los Angeles County Animal Care and Control

December 2, 2018
8. User Interface
   8.1. Overview of User Interface
   8.2. Screen Frameworks or Images
   8.3. User Interface Flow Model

9. Database Design
10. Requirements Validation and Verification
11. Glossary
12. References
## Revision History

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Reason For Changes</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team</td>
<td>9/25/18</td>
<td>Initial Creation</td>
<td>1.0</td>
</tr>
<tr>
<td>Team</td>
<td>12/2/18</td>
<td>First Draft Polish</td>
<td>2.0</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Purpose

The Animal Care and Utility Application is designed for and intended for use by Los Angeles County Animal Care and Control. The main purpose of the application is to help shorten lines and wait times at care centers. This shall be achieved by providing simple information to the user and employ the use of a ticketing system to receive help from a clerk when required.

1.2 Document Conventions

Not applicable.

1.3 Intended Audience and Reading Suggestions

The intended audience for this documentation should have some technical knowledge of the inner workings of software development. It is not required but it would help to fully understand everything presented. It is directed towards developers, project managers, testers and documentation writers.

1.4 System Overview

The Animal Care and Utility Application will provide users with access to a FAQ and Ticketing system. The application is laid out with a homepage that includes access to a search functionality and quick access buttons to other pages. Each page is also accompanied by a side menu in order to return to prior pages or navigate to other pages of the application. The general flow of the application involves receiving input from the user in the form of a question search, the application will then attempt to return an answer. If an answer is unable to be found the user will be prompted, if on-site, if they would like to submit a ticket for help.
2. Design Considerations

2.1 Assumptions and Dependencies

- Node js platform, Microsoft SQL database
- Front end OS: ios and android, Back end OS: independent
- Possibilities: Machine learning

2.2 General Constraints

- Hardware/end-user environment
  - User will need access to a smart device (i.e., phone or tablet).
  - Internet access.
- Data repository and distribution requirements
  - Back end should have significant storage for general information and ticketing data.
- Security requirements
  - Application needs to be secured according Los Angeles County standards.
- Network communications.
  - User needs constant and stable network connection.

2.3 Goals and Guidelines

- Primary goal: get information to users as quickly as possible
- The KISS principle ("Keep it simple stupid!")
- The software has a mandatory delivery date that must be met by May 2019
- The product should work, look, and feel like an existing product

2.4 Development Methods

Agile Development. Further reading can be found at https://en.wikipedia.org/wiki/Agile_software_development.

3. Architectural Strategies

- Technologies used - Nativescript, Vue, Loopback, SQL, Javascript, Typescript.
- Future plans - Machine Learning, Web application.
- External databases - Microsoft SQL, Vuex.
• Communication mechanisms - HTTP.

4. System Architecture

- Touch Screen
- GPS

D.F.D. Level 0

Animal Care Utility App A.C.U.A.

- 2.1 Main Module
- 2.2 Search Module
- 2.3 GPS Module
- 2.4 Ticketing Module
- 2.5 Speech to Text Module
- 2.6 Machine Learning Module

- Phone Screen
Level 1 Data Flow Diagrams (DFD) and Control Flow Diagrams (CFD) should probably go here.

Describe how the higher-level components collaborate with each other in order to achieve the required results. Don't forget to provide some sort of rationale for choosing this particular decomposition of the system (perhaps discussing other proposed decompositions and why they were rejected). Feel free to make use of design patterns, either in describing parts of the architecture (in pattern format), or for referring to elements of the architecture that employ them. Diagrams that describe a particular component or subsystem in detail should be included within the particular subsection that describes that component or subsystem.
5. Policies and Tactics

5.1 Choice of which specific products used

- Nativescript - Allows us to do mobile development across Android and iOS.
- Vue - Used for the front end. Development is similar to web development for future porting options.
- Loopback - Used for the front end. Creates an API service to handle object relational mapping between the front end and database.
- Microsoft SQL - database that the Animal Care and Control company is using.
- Javascript - Main programming language for front and back end
- Typescript - Javascript with static type to help development.
- VS Code - IDE that specializes in Vue, Javascript, and Typescript files.
- Node js - Platform used to run back end service and front end development.

5.2 Plans for ensuring requirements traceability

    Unit testing.

5.3 Plans for testing the software

    Unit testing.

5.4 Coding guidelines and conventions

    ESLint and TSlint are used to follow a coding style and guideline

5.5 How to build the System’s deliverables

    The front end code will be compiled through Nativescript’s Sidekick application.
    The back end code will be initialized on a cloud hosting and deployment site.

5.6 Database Abstraction

    Loopback allows for the use of any database via connector libraries.
6. Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

This is where Level 2 (or lower) DFD’s will go. If there are any additional detailed component diagrams, models, user flow diagrams or flowcharts they may be included here.

6.x Name of Component (Module)

6.x.1 Responsibilities

The primary responsibilities and/or behavior of this component. What does this component accomplish? What roles does it play? What kinds of services does it provide to its clients? For some components, this may need to refer back to the requirements specification.

6.x.2 Constraints

Any relevant assumptions, limitations, or constraints for this component. This should include constraints on timing, storage, or component state, and might include rules for interacting with this component (encompassing preconditions, post conditions, invariants, other constraints on input or output values and local or global values, data formats and data access, synchronization, exceptions, etc.)

6.x.3 Composition

A description of the use and meaning of the subcomponents that are a part of this component.

6.x.4 Uses/Interactions

A description of this components collaborations with other components. What other components is this entity used by? What other components does this entity use (this would include any side-effects this entity might have on other parts of the system)? This concerns the method of interaction as well as the interaction itself. Object-oriented designs should include a description of any known or anticipated subclasses, superclass’s, and metaclasses.

6.x.5 Resources
A description of any and all resources that are managed, affected, or needed by this entity. Resources are entities external to the design such as memory, processors, printers, databases, or a software library. This should include a discussion of any possible race conditions and/or deadlock situations, and how they might be resolved.

6.x.6 Interface/Exports

The set of services (classes, resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

Much of the information that appears in this section is not necessarily expected to be kept separate from the source code. In fact, much of the information can be gleaned from the source itself (especially if it is adequately commented). This section should not copy or reproduce information that can be easily obtained from reading the source code (this would be an unwanted and unnecessary duplication of effort and would be very difficult to keep up-to-date). It is recommended that most of this information be contained in the source (with appropriate comments for each component, subsystem, module, and subroutine). Hence, it is expected that this section will largely consist of references to or excerpts of annotated diagrams and source code.
7. Detailed Lower level Component Design

Other lower-level Classes, components, subcomponents, and assorted support files are to be
described here. You should cover the reason that each class exists (i.e. its role in its package; for
complex cases, refer to a detailed component view.) Use numbered subsections below (i.e.
“7.1.3 The ABC Package”). Note that there isn't necessarily a one-to-one correspondence
between packages and components.

7.x Name of Class or File

7.x.1 Classification
The kind of component, such as a subsystem, class, package, function, file, etc.

7.x.2 Processing Narrative (PSPEC)
A process specification (PSPEC) can be used to specify the processing details

7.x.3 Interface Description

7.x.4 Processing Detail

7.x.4.1 Design Class Hierarchy
Class inheritance: parent or child classes.

7.x.4.2 Restrictions/Limitations

7.x.4.3 Performance Issues

7.x.4.4 Design Constraints

7.x.4.5 Processing Detail For Each Operation
8. Database Design

The following are tables that we have designed for the company. We will also be using a tables from them, such as a person table, animal table, and kennel table.

**Ticket**

Function: This table is used to insert a ticket into the system. A ticket will reference a person via their phone number and will contain a subject about the question, the question, and the location where the ticket was requested.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type (size) [default value]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TICKET_ID</td>
<td>varchar(10)</td>
<td>Ticket Number in form of XXXXXX</td>
</tr>
<tr>
<td>PHONE_AREA_CODE</td>
<td>varchar(3)</td>
<td>Reference to a person’s phone area code</td>
</tr>
<tr>
<td>PHONE_NUMBER</td>
<td>varchar(7)</td>
<td>Reference to a person’s phone number</td>
</tr>
<tr>
<td>Time_In</td>
<td>datetime</td>
<td>The system automatically fills in the current date and time when a record is inputed.</td>
</tr>
<tr>
<td>SUBJECT</td>
<td>varchar(255)</td>
<td>The subject of the question.</td>
</tr>
<tr>
<td>INFO</td>
<td>varchar(255)</td>
<td>Detailed question</td>
</tr>
<tr>
<td>GEO_LOCATION</td>
<td>varchar(255)</td>
<td>Location where the person submitted the ticket</td>
</tr>
</tbody>
</table>
**FAQ**

Function: This table is used to store faqs and insert new questions into the system. A faq will contain the question, the answer, a frequency counter, and check for if it has an answer.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type (size) [default value]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUESTION_ID</td>
<td>varchar(10)</td>
<td>Question ID in form of XXXXXX</td>
</tr>
<tr>
<td>QUESTION</td>
<td>varchar(255)</td>
<td>A frequently asked question that can be answered without an employee’s explanation</td>
</tr>
<tr>
<td>ANSWER</td>
<td>varchar(255)</td>
<td>A detailed explanation in response to the question</td>
</tr>
<tr>
<td>COUNT</td>
<td>Integer</td>
<td>A frequency counter to determine how popular an answered question is or how demanding an unanswered question is.</td>
</tr>
<tr>
<td>IS_ANSWERED</td>
<td>Boolean [false]</td>
<td>A condition to determine if this is a new question without an answer (Needs to be filled in by company manually) or an answered question.</td>
</tr>
</tbody>
</table>
9. User Interface

9.1 Overview of User Interface

The UI will be laid out with a search bar at the top (on pages that use it) followed by quick access buttons to other features. Other areas of their application will be listed in a similar fashion, with text fields at the top of the page and large buttons for any prompts or navigation. Each page shall have a sidebar menu that has access links to other areas of the application.

9.2 Screen Frameworks or Images

These can be mockups or actual screenshots of the various UI screens and popups.

9.3 User Interface Flow Model

A discussion of screen objects and actions associated with those objects. This should include a flow diagram of the navigation between different pages.
## 10. Requirements Validation and Verification

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Design</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main App Design</td>
<td>UI</td>
<td>Vue allows for modular components</td>
</tr>
<tr>
<td>Search</td>
<td>UI and API request</td>
<td>Front end will request access to filtered data from the back end and display data once received</td>
</tr>
<tr>
<td>GPS</td>
<td>mobile component</td>
<td>Core library in Nativescript</td>
</tr>
<tr>
<td>Ticketing</td>
<td>UI and API request</td>
<td>Front end will send data to the back end and display data once received</td>
</tr>
<tr>
<td>Speech-to-text</td>
<td>mobile component</td>
<td>Core library in Nativescript</td>
</tr>
<tr>
<td>Machine Learning</td>
<td>API request</td>
<td>Front end will send data to the back end and display data once received</td>
</tr>
</tbody>
</table>
11. Glossary

No defined terms in this revision of the document.

12. References