System and Software Architecture Description (SSAD)

Farmworkers Safety App

Team 09

TEAM MEMBER NAME	ROLES
Juan Andrade	Project Manager Life Cycle Planner Developer
Theerapat Chawannakul	System Architect Developer
Fereshteh Khorzani	Independent Verification and Validation Quality Focal Point
Basir Navab	Life Cycle Planner Developer
Vahagen Sinanian	Operational Concept Developer Developer
David Tasky	Independent Verification and Validation Quality Focal Point

04/19/2017

VERSION HISTORY

Date	Author	Version	Changes Made	Rationale
10/12/2016	Shobhit Agarwal	1.0	Sections 1 and 2 diagrams added	Draft for the FCR ARB Submission
10/16/2016	Viraj Sahai	1.1	Sections 1 and 2 completed	Final copy for FCR ARB Submission
10/17/2016	Shobhit Agarwal	1.2	Adding section 2.2, adding header and footers, and formatting the document	Final copy for FCR ARB submission
11/26/2016	Viraj Sahai	2.0	Added section 3 and 4	Draft of final copy for DCR TRR ARB
12/02/2016	Juan Andrade	2.1	Completed all sections	Final copy for DCR TRR ARB
12/05/2016	Viraj Sahai	2.2	Finalized the document	Final copy for upload
02/12/2017	Theerapat Chawannakul	3.0	Changed members of the team and updated architecture of system	Final copy for Re-baselined ARB
04/19/2017	Juan Andrade	3.1	Updated Process Realization Diagrams and Added Class Diagram	Updated Copy for As Built Package.

Table of Contents

Table of Tables

Table of Figures

1. Introduction

1.1 Purpose of the SSAD

The purpose of this SSAD is to document the design of the Farmworker Safety App. The SSAD will be a guiding tool for analysis and detailed description of the system architecture. It would serve as a reference for the future software developer/maintainer to understand and enhance the design in the future.

1.2 Status of the SSAD

-Sections 1 and 2 have been completed for Foundations Commitment Package.
-All other sections will be completed at a later date.
-Naming conventions are likely to change in further revisions (refer to section 2.2 System Analysis Rationale for further details).

2. System Analysis

2.1 System Analysis Overview

The main purpose of the Farm Worker App is to provide notifications to farmworkers and farmers about the temperature levels if it exceeds a threshold so that they can take breaks and save themselves from heat related stress. The app will also collect feedback from the farmers regarding their health conditions so that they can be monitored for symptoms of heat related stress. It will also include emergency services like one touch call to CalOsha, 911 and other emergency contacts. It may also provide a platform for contractors and farmers to search for available farmworkers looking for jobs.

2.1.1 System Context

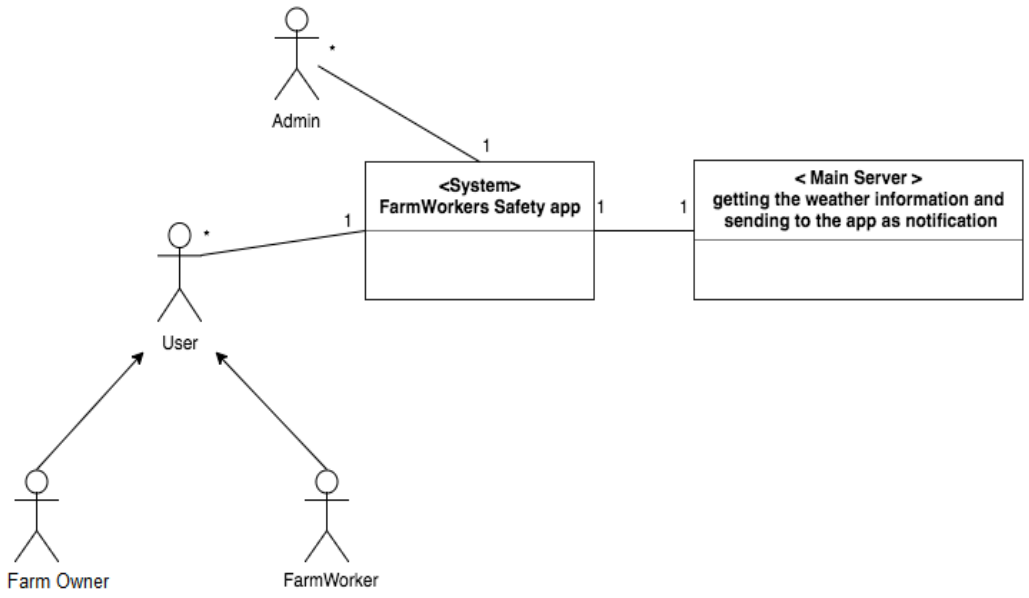

Figure 1: System Context Diagram

Table 1: Actors Summary

Actor	Description	Responsibilities
Admin	Administrators to manage the system and functionalities	Control user access rights; manage data; post content; manage users
User (Farm Owner)	Users that own farm	Manage farmworkers; register new farmworkers to the farm; hire farmworkers
User (Farm Worker)	Users that work on the farms	Submit feedback; take quizzes; update location; get himself educated; take preventive measures

2.1.2 Artifacts & Information

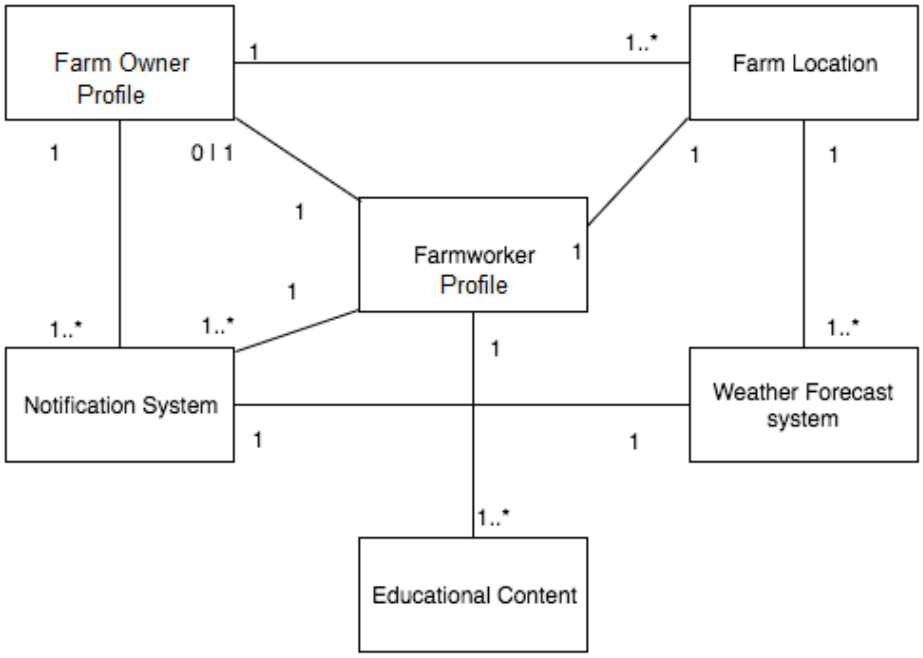

Figure 2: Artifacts and Information Diagram

Table 2: Artifacts and Information Summary

Artifact	Purpose
Farmworker Profile	Contains information regarding the farmworkers
Farmer Profile	Contains information regarding the farmer/contractor
Farm Location	Location of the farm at which the farmworker is working
Notification System	Provides notifications to the farmworkers regarding temperature, quizzes and feedback
Educational Content	Videos and Quizzes to educate the farmworkers regarding heat related stress and prevention practices
Weather Forecast System	Fetches weather information to be able to send notifications to the farmworkers

2.1.3 Behavior

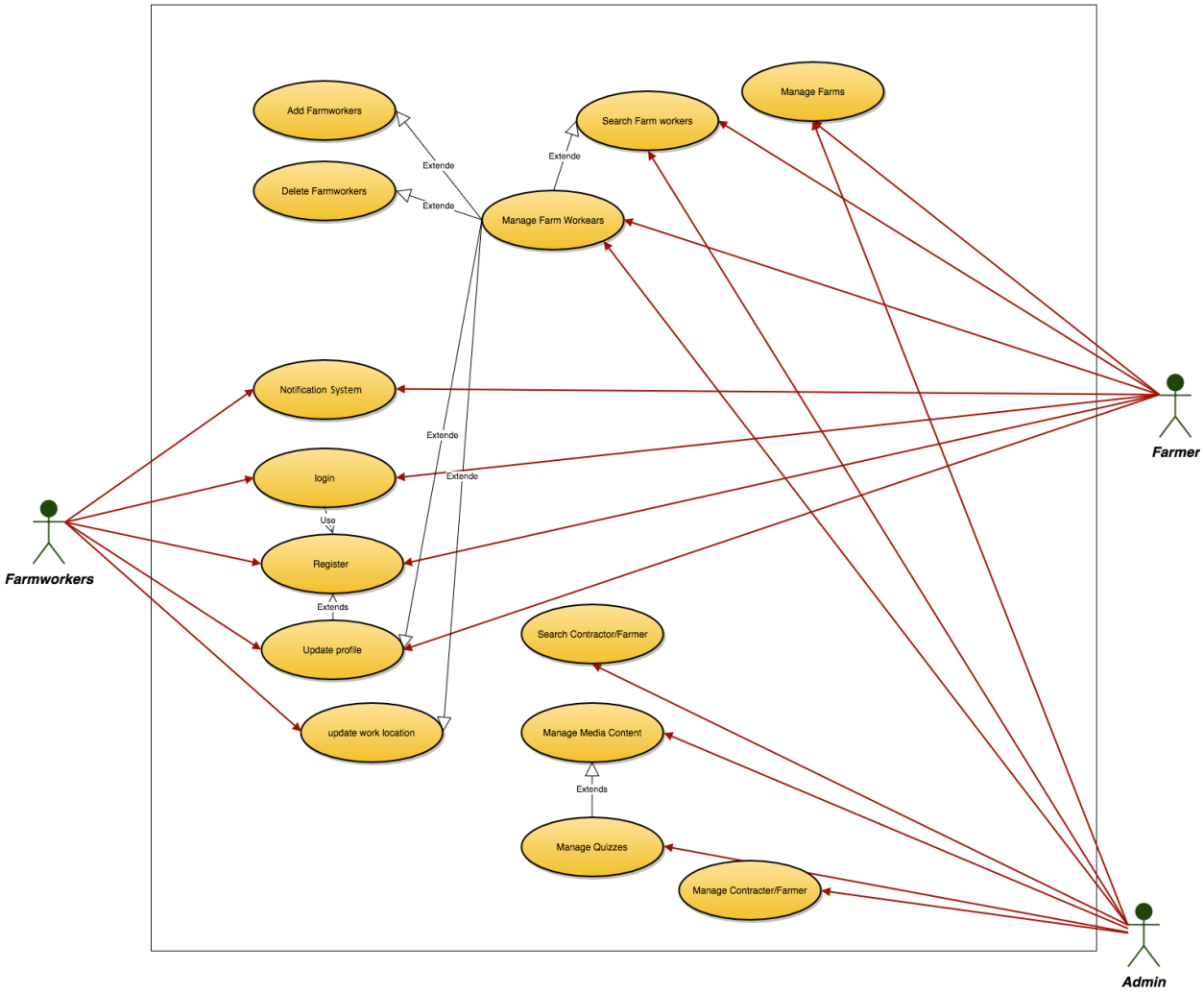

Figure 3: Process Diagram

2.1.3.1 Authentication

2.1.3.1.1 Login

Table 3: Process Description-login

Identifier	login
Purpose	Determine if a person logging in to the system can be authenticated, and, if so, what the person's privileges are as a user.
Requirements	Authorization and Authentication
Development Risks	None
Pre-conditions	System database is properly initialized. Login functionality is successfully implemented.
Post-conditions	If user exists in the system the user shall be logged in with appropriate access rights

Table 4: Typical Course of Action-login: successful

Seq#	Actor's Action	System's Response
1	User enter username and password and click enter	
2		Check whether username and password is true.
3		Redirect user to user's main page

Table 5: Alternate Course of Action-login: fail

Seq#	Actor's Action	System's Response
1	User enter username and password and click enter	
2		Check whether username and password is true.
3		Return error message like "user name or password incorrect" or "user doesn't exist"

2.1.3.1.1 Logout

Table 6: Process Description-logout

Identifier	logout
Purpose	Logout the user from the system
Requirements	Authorization and Authentication
Development Risks	None
Pre-conditions	User is a valid system user User is logged in to the system and the session exists.
Post-conditions	The login session is terminated.

Table 7: Typical Course of Action-logout: successful

Seq#	Actor's Action	System's Response
1	User clicks on the logout button	
2		User is logged out of the system and a message is displayed saying " You have been successfully logged"

Table 8: Alternate Course of Action-logout: fail

Seq#	Actor's Action	System's Response
1	User logs out an expired session	
2		Return session expired message and return to log in page

2.1.3.2 Registration

2.1.3.2.1 User Registration

Table 9: Process Description-Registration

Identifier	Registration
Purpose	Register user to the system
Requirements	Internet connection on user end. Registration webpage.
Development Risks	None
Pre-conditions	User is not an already registered user

	User has loaded the registration page
Post-conditions	User is registered to the system

Table 10: Typical Course of Action-Registration: successful

Seq#	Actor's Action	System's Response
1	User fills in the registration form	
2		User is registered into the database
3		Redirect user to the main profile page

Table 11: Alternate Course of Action-Registration: fail

Seq#	Actor's Action	System's Response
1	Already a registered user tries re-registering	
2		Return a "user already registered" error

2.1.3.3 Profile Management

2.1.3.3.1 Update Profile

Table 12: Process Description-Update Profile

Identifier	Update profile
Purpose	Update personal user profile
Requirements	User should be registered
Development Risks	None
Pre-conditions	User should be logged in
Post-conditions	Profile data is updated in the database

Table 13: Typical Course of Action-Update Profile: successful

Seq#	Actor's Action	System's Response
1	User updates the required field	
2		User data is updated in the database and a message is displayed as "Profile Updated"

Table 14: Alternate Course of Action-Update Profile: fail

Seq#	Actor's Action	System's Response
1	User updates unique data like email to one that is already registered	
2		Return a "Update failed. Email already registered to another user" error

2.1.3.3.2 Update Location

Table 15: Process Description-Update Location

Identifier	Update work location
Purpose	Update location of user so as to be able to send notifications
Requirements	User should be registered
Development Risks	None
Pre-conditions	User should be logged in
Post-conditions	Location data is updated in the database

Table 16: Typical Course of Action-Update Location: successful

Seq#	Actor's Action	System's Response
1	User updates the location using place name or zip-code	
2		Location data is updated in the database and a message is displayed as "Location Updated"

Table 17: Alternate Course of Action-Update Location: fail

Seq#	Actor's Action	System's Response
1	User input invalid place or zip-code	
2		Return a "Invalid Location input" error

2.1.3.4 Farm Management

2.1.3.4.1 Manage Farms

Table 18: Process Description-Manage farms

Identifier	Manage Farms
Purpose	Add/Delete Farms; Update Farm attributes like area, facilities etc.; View farm details and list of employed workers
Requirements	User should be registered as a farmer
Development Risks	None
Pre-conditions	User should be logged in User should be a farmer
Post-conditions	Farm data is updated in the database

Table 19: Typical Course of Action-Manage Farms: successful

Seq#	Actor's Action	System's Response
1	User performs the required changes	
2		Farm data is updated in the database and a message is displayed as "Farm data updated"

Table 20: Alternate Course of Action-Manage Farms: fail

Seq#	Actor's Action	System's Response
1	User inputs invalid data like invalid zip-code	
2		Return a "Update failed. Invalid zip-code" error

2.1.3.5 Farmworker Management

2.1.3.5.1 Add Farmworker

Table 21: Process Description-Add Farmworker

Identifier	Add farmworkers
Purpose	Add Farmworkers to the Farmer's farm
Requirements	User should be registered as a farmer

Development Risks	None
Pre-conditions	User should be logged in User should be a farmer User should add farmworkers to valid farm
Post-conditions	Farmworker data is updated in the database Corresponding update also shows in the Farmworkers profile

Table 22: Typical Course of Action-Add Farmworker: successful

Seq#	Actor's Action	System's Response
1	User adds farmworkers	
2		Farmworker data is updated in the database and a message is displayed as "Farmworker added"

Table 23: Alternate Course of Action-Add Farmworker: fail

Seq#	Actor's Action	System's Response
1	User adds a farmworker who is not registered to the system	
2		Return a "Farmworker not registered" error

2.1.3.5.2 Delete Farmworker

Table 24: Process Description-Delete Farmworker

Identifier	Delete farmworkers
Purpose	Delete Farmworkers from the Farmer's farm
Requirements	User should be registered as a farmer
Development Risks	None
Pre-conditions	User should be logged in User should be a farmer User should delete farmworkers from valid farm
Post-conditions	Farmworker data is updated in the database Corresponding update also shows in the Farmworkers profile

Table 25: Typical Course of Action-Delete Farmworker: successful

Seq#	Actor's Action	System's Response
1	User deletes farmworkers	
2		Farmworker data is updated in the database and a message is displayed as "Farmworker deleted"

Table 26: Alternate Course of Action-Delete Farmworker: fail

Seq#	Actor's Action	System's Response
1	User deleted a farmworker who is not registered to the system	
2		Return a "Farmworker not registered" error

2.1.3.5.3 Search Farmworker

Table 27: Process Description-Search Farmworker

Identifier	Search farmworkers
Purpose	Search Farmworkers in the Farmer's farm to display their info
Requirements	User should be registered as a farmer
Development Risks	None
Pre-conditions	User should be logged in User should be a farmer
Post-conditions	Farmworker is fetched from the database

Table 28: Typical Course of Action-Search Farmworker: successful

Seq#	Actor's Action	System's Response
1	User searches farmworkers	
2		Farmworkers data is fetched from the database and displayed

Table 29: Alternate Course of Action-Search Farmworker: fail

Seq#	Actor's Action	System's Response
1	User searches for farmworker not registered in the system	
2		Return a "Farmworker not found" error

2.1.3.6 User Management

2.1.3.6.1 Manage Users

Table 30: Process Description-Manage Users

Identifier	Manage Contractors/Farmworkers
Purpose	Add/Delete users; Update user data
Requirements	User should be registered as an admin
Development Risks	None
Pre-conditions	User should be logged in User should be an admin
Post-conditions	Farmworker data is updated in the database Corresponding update also shows in the Farmworkers profile

Table 31: Typical Course of Action-Manage Users: successful

Seq#	Actor's Action	System's Response
1	User updates other users' data	
2		Users' data is updated in the database and a message is displayed as "User Updated"

Table 32: Alternate Course of Action-Manage Users: fail

Seq#	Actor's Action	System's Response
1	User updates unique data fields leading to table constraints violations	
2		Return a "User with same data already present" error

2.1.3.6.2 Search Users

Table 33: Process Description-Search Users

Identifier	Search Contractors/Farmworkers
Purpose	Search users
Requirements	User should be registered as an admin
Development Risks	None

Pre-conditions	User should be logged in User should be an admin
Post-conditions	Farmworker data is fetched from the database

Table 34: Typical Course of Action-Search Users: successful

Seq#	Actor's Action	System's Response
1	User searches for other users' data	
2		Users' data is fetched from the database and displayed

Table 35: Alternate Course of Action-Search Users: fail

Seq#	Actor's Action	System's Response
1	User searches for a non-existent user	
2		Return a "User not found" error

2.1.3.7 Content Management

2.1.3.7.1 Manage Media Content

Table 36: Process Description-Manage Media Content

Identifier	Manage Media Content
Purpose	Add/Delete media content
Requirements	User should be registered as an admin
Development Risks	None
Pre-conditions	User should be logged in User should be an admin
Post-conditions	Media content shows as a post to farmworkers Notifications are sent to farmworkers

Table 37: Typical Course of Action-Manage Media Content: successful

Seq#	Actor's Action	System's Response
1	User adds educational video	
2		Video is added and a "Video Added" message is displayed

Table 38: Alternate Course of Action-Manage Media Content: fail

Seq#	Actor's Action	System's Response
1	User adds content in incorrect format	
2		Return a "Unsupported file format" error

2.1.3.7.2 Manage Quizzes

Table 39: Process Description-Manage Quizzes

Identifier	Manage Quizzes
Purpose	Add/Delete Quizzes
Requirements	User should be registered as an admin
Development Risks	None
Pre-conditions	User should be logged in User should be an admin
Post-conditions	Quiz shows as a post to farmworkers Notifications are sent to farmworkers

Table 40: Typical Course of Action-Manage Quizzes: successful

Seq#	Actor's Action	System's Response
1	User adds quiz with corrected url	
2		Quiz is added and a "Quiz Added" message is displayed

Table 41: Alternate Course of Action-Manage Content: fail

Seq#	Actor's Action	System's Response
1	User add quiz with empty url	
2		Return a "Quiz link is required" error

2.1.3.8 Notification System

2.1.3.8.1 Notification System

Table 42: Process Description-Notification System

Identifier	Notification System
Purpose	Send notification to the user
Requirements	User should be registered as a user
Development Risks	None
Pre-conditions	User should be logged in
Post-conditions	Notification is sent to the user and displays in status bar

Table 43: Typical Course of Action-Notification System: successful

Seq#	Actor's Action	System's Response
1		System sends temperature and other notifications
2	Users reacts accordingly	

2.1.4 Modes of Operation

Farmworker app will operate in only one mode, so nothing further need be said of modes of operation.

2.2 System Analysis Rationale

Administrator registration:
Admin registration has not been included explicitly, as this would be created manually for each admin. Number of admins should be limited and should not be shown in the UI. Only existing admins should be able to add new admins.

User type - Farmer and Contractor:
Both these user types have the same functionality and hence are not being shown explicitly. The only difference in terms of the app between these users is that farmers can manage contractors as well, which will not be covered in the near future, and would be handled manually from the backend for the time being.

Admin can send notification:
Admin has the functionality to send manual notification to the users of the app, but this has not been shown in the use case diagram as this would be one of the implicit feature of the admin user.

3. Technology Dependent System Design

3.1 Design Overview

3.1.1 System Structure

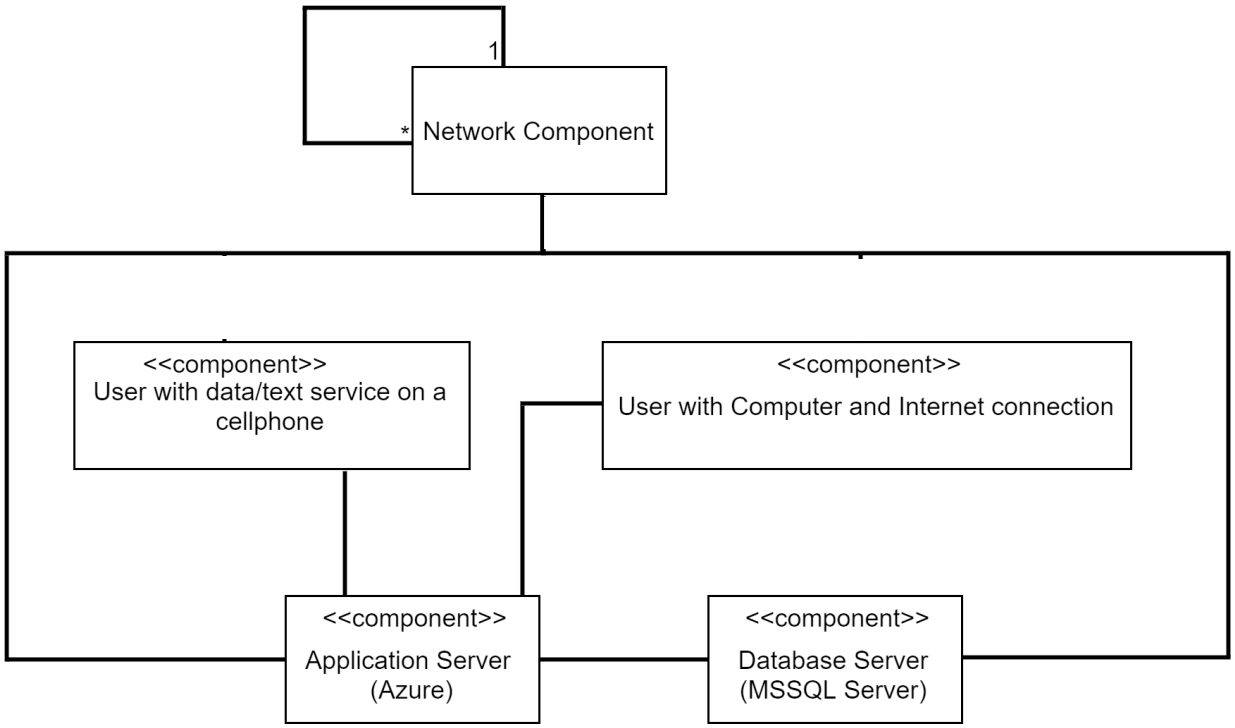

Figure 3: Hardware Component Class Diagram

Table 44: Hardware Component Description

Hardware Component	Description
Farmworker's Computer	User computer with a web browser
Farmworker's Mobile	User mobile device with text (and/or data) service
Application Server	Server used to host the web application with PHP configuration
Database Server	Server used to host the enormous user, farm and weather data
Network Component	Device through which the user's device gains internet access. i.e. Wi-Fi Router

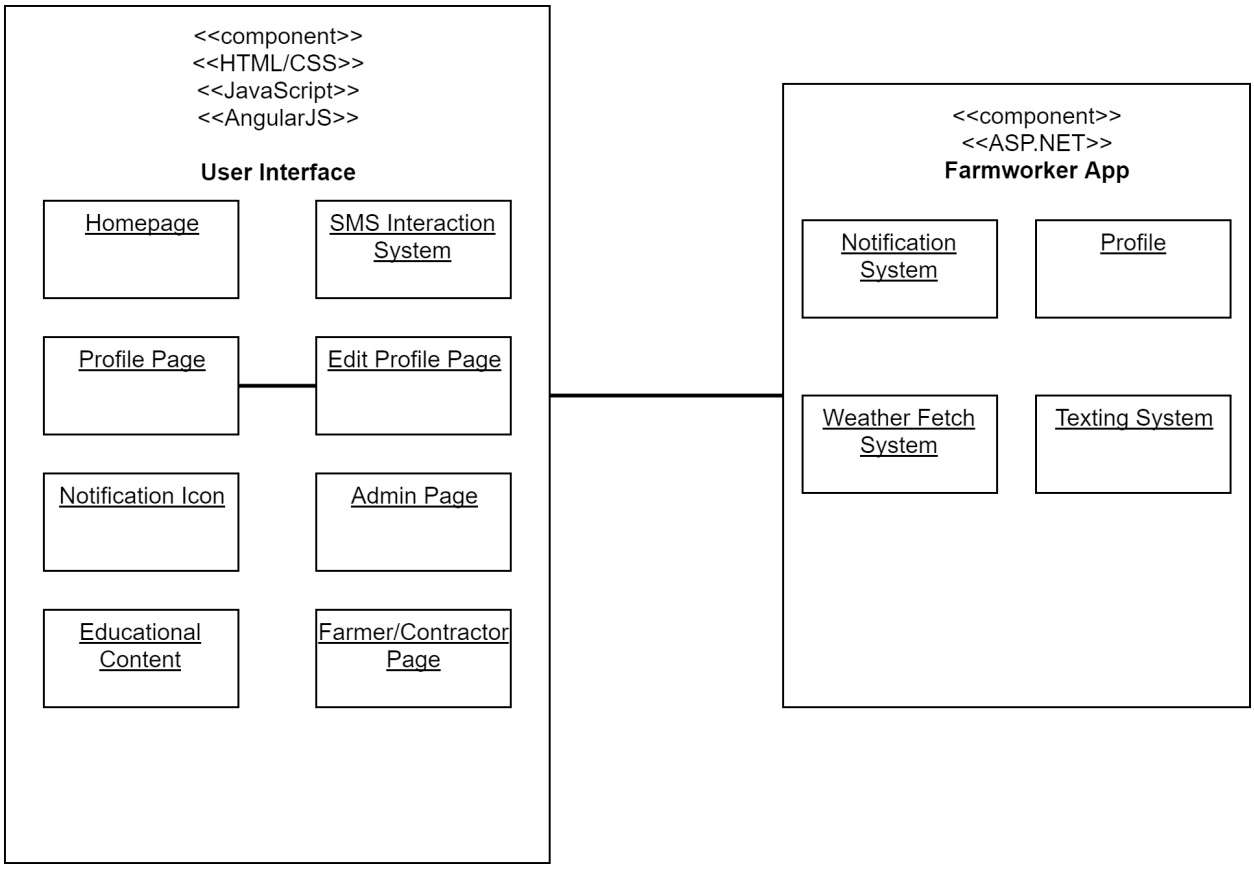

Figure 5: Software Component Class Diagram

Table 65: Software Component Description

These components and descriptions are the same as the technology-independent section except that all components in user interface will be developed with HTML and JavaScript using AngularJS framework, and all components in "Farmworker App" API will be developed with ASP.Net

Software Component	Description
Notification system	The system handles app notification of both temperature events and informing about new educational content.
Profile	The system handles adding/editing/showing users'(Farmworkers and Farmers) profile.
Weather Fetch System	The system will fetch all weather and temperature data by making API calls.
Texting System	The system is responsible for two-way text input and text notification actions.
Home page	Home page consists of current temperature information and new educational content.
SMS Interaction System	This system will use text messages to present user with choices and notifications
Profile page	Profile page allows users' to share their information and show their information to other users.
Edit Profile Page	This will allow users to update their profile data like age, location etc.
Notification icon	Notification icon shows new temperature and educational content notification.
Admin page	The Admin page allows maintainers to manage users, educational contents, and farms.
Educational Content Page	This page will display all the educational content videos and quizzes categorized accordingly.
Farmer/Contractor Page	This page will display details of farmworkers and farms under the farmer/contractor.

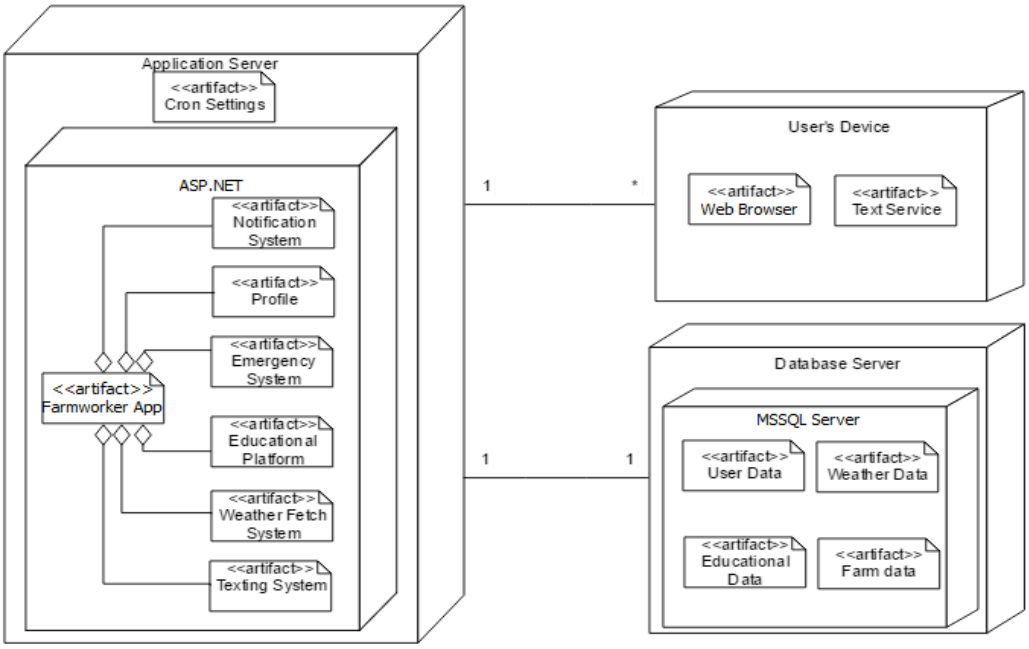

Figure 6: Deployment Diagram


+ QuizQuestions : ICollection<QuizQuestion>
+ UserEducationalContents : ICollection<UserEducationalContent>
+ UserFarms : ICollection<UserFarm>
+ WeatherHistories : ICollection<WeatherHistory>
+ QuizAttemptsUsers : ICollection<Q
+ UserCommunicationPreferences : I
+ UserEducationalContents : ICollect
+ UserEmergencyContacts : ICollecti
+ UserFarms : ICollection<UserFarm
+ QuizAnswers : ICollection<QuizAnswer>
+ IsLatest : Nullable<Int32>
+ UpdateDate : Nullable<DateTime>
+ UserID : Nullable<Int32>
+ UserRoleID : Nullable<Int32>
+ UserPermissions : ICollection<UserPermission>

Table 46: Class Dependency Diagram Description

Class	Type	Description
User	Model	This model extends the Model class from the ASP.NET framework consists on User's data access. (Will be removed in future releases and be replace by its Entity Equivalent)
User	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on User's data access. Facilitates database operations.
Farm	Model	This model extends the Model class from the ASP.NET framework consists on Farm's data access. (Will be removed in future releases and be replace by its Entity Equivalent)
Farm	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on Farm's data access. Facilitates database operations.
EducationalContent	Model	This model extends the Model class from the ASP.NET framework consists on Educational Content's data access. (Will be removed in future releases and be replace by its Entity Equivalent)
EducationalContent	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on Educational Content's data access. Facilitates database operations.
Quiz	Model	This model extends the Model class from the ASP.NET framework consists on Quiz's data access. (Will be removed in future releases and be replace by its Entity Equivalent)
Quiz	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the Quiz's main information data access. Facilitates database operations.
QuizAnswer	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the Quizzes Answers Section data access. Facilitates database operations.
QuizQuestion	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the Quizzes Questions Section data access. Facilitates database operations.
QuizAttemptsUser	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the Quizzes User Responses Section data access. Facilitates database operations.
QuizEducationalContent	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the data access of the Educational Content attached to the

		Quizzes. Facilitates database operations.
Notifications	Model	This model extends the Model class from the ASP.NET framework consists on Notification's necessary data access. (Will be removed in future releases and be replace by its Entity Equivalent)
FarmWorkersAppContext	Entity	This class extends the Entity Framework from the ASP.NET Framework. It is used to create a database context to work with it throughout the application.
LoginCredentials	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the Login Credential's data access. Facilitates database operations.
UserCommunicationPreference	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the User Communication Preferences' data access. Facilitates database operations.
UserEducationalContent	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the User Viewed Educational Contents' data access. Facilitates database operations.
UserEmergencyContacts	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the User Emergency Contact data access. Facilitates database operations.
UserPermission	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the User Security Permissions' data access. Facilitates database operations.
UserSecurityRole	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the User Security Role data access. Facilitates database operations.
UserFarm	Entity	This class extends the Entity Framework from the ASP.NET Framework, consists on the data access of the User relationships with farms. Facilitates database operations.
UserController	Controller	This controller extends the Controller class from the ASP.NET framework handles all requests regarding User's features and functionalities. And prepares the data in a JSON format to be rendered in the front end.
EducationalContentController	Controller	This controller extends the Controller class from the ASP.NET framework handles all requests regarding Educational Content's features and functionalities. And prepares the data in a JSON format to be rendered in the front end.
QuizController	Controller	This controller extends the Controller class from

		the ASP.NET framework handles all requests regarding Quiz's features and functionalities. And prepares the data in a JSON format to be rendered in the front end.
FarmController	Controller	This controller extends the Controller class from the ASP.NET framework handles all requests regarding Farm's features and functionalities. And prepares the data in a JSON format to be rendered in the front end.
EducationalContentController	Controller	This controller extends the Controller class from the ASP.NET framework handles all requests regarding Educational Content Controller's features and functionalities. And prepares the data in a JSON format to be rendered in the front end.
NotificationsController	Controller	This controller extends the Controller class from the ASP.NET framework handles all requests regarding Notifications' features and functionalities. And prepares the data in a JSON format to be rendered in the front end.
Login Credentials Controller	Controller	This controller is included to manage the user's session activities. Such as Login, Logout, Session Persistent Data, etc. And prepares the data in a JSON format to be rendered in the front end.

ocess Realization

ToList<EducationalContent>

Create List<EducationalContent>

Click Here to See Full Size Diagram

Figure 10: On-Demand SMS Farm Registration & Educational Content Request Process Realization Diagram

3.2 Design Rationale

As it can be observed on Figure 14, on the design class diagram, we decided to the implement Model-View-Controller architecture, and APS.NET was selected as our framework to reach this goal. In order to successfully implement MVC to that, in the technology dependent design class diagram, all the controllers must extend ASP.NET's Controller Class, which is provided by the framework, and all the models must extend ASP.NET'S Model Class, which is also provided by the framework of choice. We also decided to use AngularJS as a frontend framework because it is one of most popular framework for web application. Moreover, the framework provides a lot of support for user interface rendering and testing.

4. Architectural Styles, Patterns, and Frameworks

Table 47: Architectural Styles, Patterns, and Frameworks

Name	Description	Benefits, Costs, and Limitations
3-tier architecture	Separate the application into 3 tiers: a presentation tier using with a web browser/text messages, a logic tier with a ASP.Net server, and a data tier with MSSQL.	- Using web browser allows the user to use any devices that they want, resulting in no need to maintain users' devices. - Update through text messages makes the system more robust and platform independent. - Any change made in the application in the server will be automatically presented to the users. - There is no cost for adopting the architecture. - Need to acquire servers for application and database.
MVC	Models, Views, and Controller pattern	- MVC has separate layers for each specific responsibility, resulting in great structures and ease of modifying. - There are no costs for the pattern. - If the application is complex, the model layer will be very complicated. So main idea is to keep the system as modular and as simple as possible.
ASP.NET	ASP.NET is a web application framework based on C#	- ASP.NET has very good knowledge documents, tutorials, and community. - ASP.NET is free for using. - The team needs to study the framework and good practice before developing the application.
AngularJS	AngularJS is a web application framework based on Javascript	- AngularJS provides an easy way to render user interface. - Writing unit tests in AngularJS is easy - It makes the code organized. Thus, maintainers can maintain the system in the future.