We Care : Online Disease Tracker System

Nadini Jayatissa
Department of Computer Science and Engineering, University of Moratuwa
Sri Lanka
nadinij.13@cse.mrt.ac.lk

***Abstract*—The Online Diseases Tracker System is an online web application which intends to help the people with self-diagnosis or self-triage by means of using an algorithm. After analyzing the symptoms, the system provides the predicted diseases along with the probability of the patient having the disease and the doctors to consult for each disease. For a better diagnosis the system provides the patients to keep an up-to-date profile of patient's medical history. Apart from disease tracking the patients can find ideal doctors to consult for a particular disease, can find hospitals where a particular doctor is available and they can contribute information to the system regarding diseases and their symptoms, doctors and hospitals which will be later added to the system database after a thorough inspection by the system administration. For the system administrators the system provides a user friendly platform to interact with the system database and thereby to update the system database.**

Keywords—symptom checker; diagnosis; doctor; disease; hospital; patient; web application; online; Symfony

I. INTRODUCTION

The Online Disease Tracker System is intended to help the registered users with the self-diagnosis after analyzing the symptoms the user entered and the medical profile the user maintains in the system. The patients can contribute information to the system which will be evaluated by the system administrators and if the provided information is marked as success the owner of that information will be sent a response email. Apart from the registered users the system admins have access to the system and the system provides a platform for them to manage the database of the system. As for now the system keeps a database maintained by the system admin through the user interface. But when scaling the system, instead of information from the system database, a web search will be added to retrieve information. The system was developed to make the diagnosis process and the doctor consulting process easier for the patients.

Even though there are many online symptom checkers available, not many of the systems provide more details about what doctors to consult and which hospitals to go for that particular disease and the probability of the user having that disease. Moreover, the predicting process does not take into consideration the prevailing medical conditions of the patient. This system provides solution for all the above mentioned concerns.

The online application was developed to meet the project requirements in providing symptom checker and an information management system. The system displays the predicted diseases for the provided symptoms and the system provides a platform for the system administrators to manage the system database. The Symfony framework [1] has been used for the development of the system where the MVC architecture has been incorporated. The twigs represent the view layer and the controller files which contain the business logic represent the controller layer. The entity files represent the model layer. Entity files have used the doctrine PHP library which made persisting and reading information to and from a database easier. Symfony framework provides reusable PHP libraries which makes the development convenient.

The outcome of the project is the online system ensuring complete protection of data from unauthorized access. All confidential information accesses are subject to user identification and password control.

This paper consists of 6six sections altogether where the Section I is the introduction. The Section II compares the system with the existing other similar systems. Section III includes the system design and system models and Section IV includes the system implementation. Section V and VI incudes system evaluation and the conclusion respectively.

II. LITERATURE REVIEW

Online Symptom checkers prevail to a higher extent thus providing self-diagnosing facilities. The developed system is objected to serve basically the people living in Sri Lanka with the diseases found in Sri Lanka and Sri Lankan doctors and hospitals. Moreover, the system intends to customize the list of predicted diseases for a particular set of symptoms and the probability of the user having the disease depending on the user. This has been implemented by giving the users the facility to keep an up-to-date medical profile.

The symptom checker handled by the WebMD Corporation [2] is a similar system which provides help for patients with self-diagnosis. Nevertheless, this system does not provide any information regarding the doctors and hospitals in-need for those predicted diseases. In addition, WebMD symptom checker does not take into consideration the medical conditions prevailing in the patient and also it lacks the user involvement in the system. Compared to the system in question, the system has no user involvement for the information used in the application and no separate medical profiles for each registered user [2].

The "Isabela" Symptom checker [1] is another popular similar system which provides a platform for patients to do self-diagnosis. This system provides two options for the

users: either they can register in the system and have a better diagnosis process or else they can check their symptoms by providing the relevant information without registering in the system. Although the diseases are predicted and the relevant sources of information will be provided for each disease they do not clearly mention the doctors and hospitals in need for those diseases. Further, "Isabela" symptom checker does not provide the probability associated with each predicted disease and also the users do not have the facility to maintain an up-to-date medical profile [1].

The system was developed using the Symfony framework where MVC architecture has been incorporated. Symfony is integrated with many decoupled and reusable PHP which are used as the standard foundation on which the application has been built on. Doctrine is another library which symphony comes integrated with, to make persisting and reading information to and from the database easier [3].

III. System Models

A. System Requirements

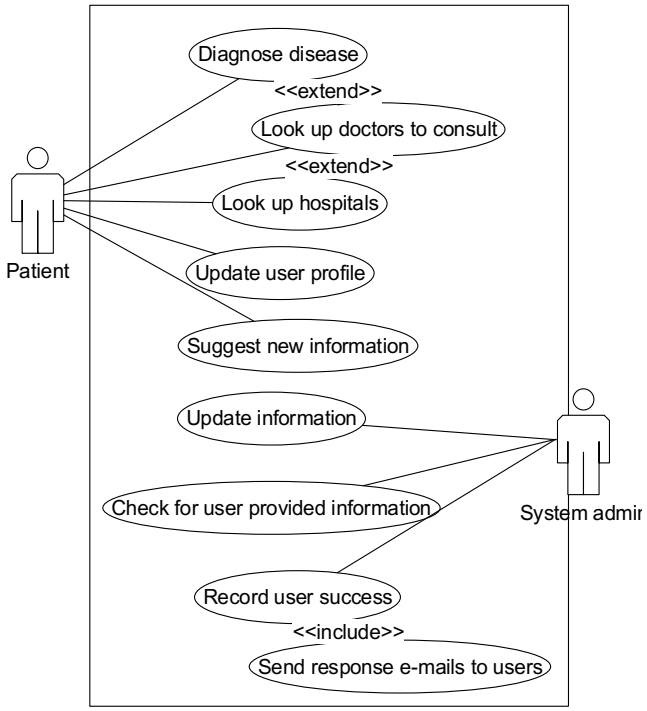

Fig. 1. Use Case Diagram of the System

As shown in Figure 1 the main functional requirements of the system are centered on symptom checking and information management sub systems. There are two types of users in the system, both user types have to first register in the system to have access to the system. The user type 'patient' once logged into the system, can do the self-diagnosis by entering the symptoms. To have more accurate results the users have to update the user profile with the relevant and recent information. Beyond the process of just knowing the predicted diseases the user can find doctors for those diseases and the hospital where those doctors are available. In addition, the users can contribute information to the system. The system admins can log into the system and manage the information in the database. Apart from that they check for the recent information provided by the 'patient' type users and mark those information as success or failure. For the owners whose information were marked as success, a response e-mail will be sent.

The non-functional requirements include performance, security and reliability. The online disease tracker system is available all the time. There shall be no more than 4% down time. The security of the user profiles and personal data have been secured with user identification and passwords. A normal user should be able to use the system at a productive rate after a maximum time of 1 minute and for a power user it will take 30 seconds or less time of learning. The average number of errors done by the experienced users shall not exceed two per hour of system use.

B. System Design

The software architecture of the system is based on the MVC architecture with the view layer, controller layer and the model layer. MVC architecture allows the re-use of business logic across the application lessening the redundancy of code. Moreover, multiple user interfaces can be developed without concerning the codebase. Therefore, the developer can specialize and focus on UIs and the business logic separately. The View Layer contains all of the visible web pages and handles all input from and output to the user. The Controller layer which includes the business logic, handles all of the controlling between the view and model while controlling the access to corresponding user types and the Model layer consists of the data access process and stored procedures contained within provides the persistence required for the system.

The class diagram of the system is shown Figure 2. The two classes, patient and admin, are derived from User class using inheritance concept. Composition and aggregation have been used in many instances. For example, the relationship between Symptom class and Disease class is an aggregation. Disease is the container class and symptom is the contained class. Polymorphism has been used to override some methods in the parent class. (e.g. Redirect_after_login method in User class was overridden in the Patient and Admin classes).

The main activity diagram in the Online Disease Tracker system illustrates in Figure 3. Once the patient was given access to the system, he/she can enter the symptoms to the system. Then the system analyzes the provided symptoms and the relevant details in the user profile and displays the final list of the predicted diseases. Thereafter the patient can select one disease at a time and view the doctors available for treating that disease. Similarly, by selecting one doctor at a time patient can view the hospitals where that particular doctor is available.

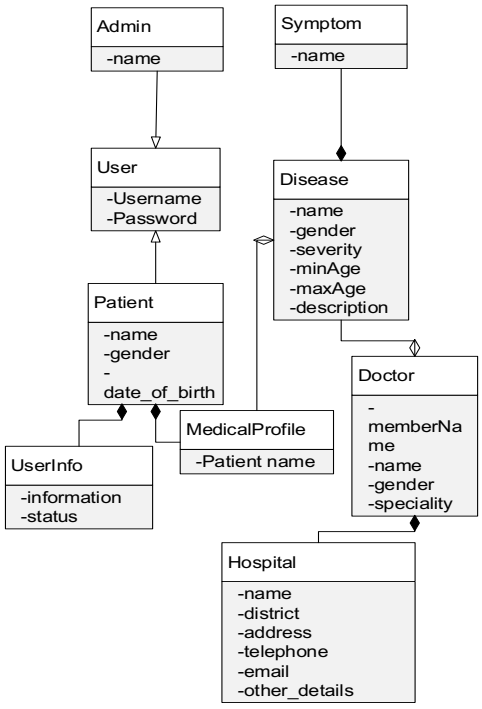

Fig. 2. Class Diagram

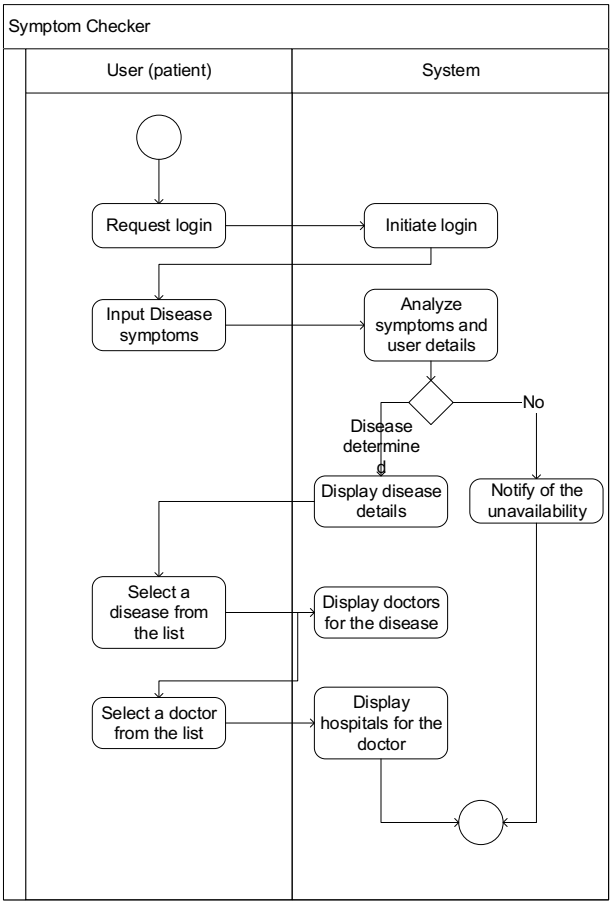

Fig. 3. Activity Diagram of the Symptom Checker

IV. System Implementation

A. Implementation Procedure

The system was developed using the PHP Symfony framework where MVC architecture has been incorporated. Symfony is an object- oriented framework. Therefore, relational database information was mapped to an object model with ORM tool provided by Doctrine. The entity classes created in this process represented the Model layer. The Controller layer comprise of the controllers. Controllers extend the Framework Bundle base controller and use annotations to configure routing, caching and security. The business logic was decoupled from the framework, while, at the same time the controllers and routing was coupled to the framework to get the most out of it.

The templates which represent the View layer were formatted using the Twig templating. The base.html.Twig file was initially formatted by dividing the template into blocks. The other templates extend the base.html.Twig to make formatting easier and make all the templates follow a common style. The forms created in the project were Symfony forms because of the convenience it provides in form handling, form rendering and form submitting. The forms that needed to be reused were created as PHP classes and stored in the "AppBundle/Forms" namespace.

The user login of the system was built on top of the authentication bundled with symfony. Authentication is configured in security.yml, primarily under the firewalls key with the anonymous key enabled. The passwords have been encoded with the bcrypt encoder instead of the traditional SHA-512 hashing encoder. The main advantages of bcrypt are the inclusion of a salt value to protect against rainbow table attacks, and its adaptive nature, which allows making it slower to remain resistant to brute-force search attacks.

The system is developed in a user friendly way such that the system gives feedback for every update action the system admins make via the system interfaces and for patients the information displayed are differentiated as according to the best way it should be. Using Doctrine built-in statements to persist and retrieve data from the database provided a risk free environment. In every other case prepared statements were used in executing queries to make the system completely immune to SQL injection attacks.

Mailing functionality has been added to the system and was implemented using the swiftmailer configuration. Instead of using a regular SMTP server to send emails, Gmail has been used to the send the mails. The Swift Mailer library works by creating, configuring and then sending Swift_Message objects. The "mailer" is responsible for the actual delivery of the message and is accessible via the mailer service [4].

The information about diseases, doctors and hospitals have been stored in separate tables. As the next step the symptoms have been linked to the relevant diseases. Similarly, the available hospitals to relevant doctors and doctors to the diseases for which they should be consulted have also been linked. In addition, the patients registered in

the system update their profiles with their medical histories and that information will be accompanied with the other user details to be used in the process of predicting the diseases.

B. The Algorithm

The Figure 4 shows the main algorithm used in the system is the algorithm used to diagnose the diseases for a particular user for the symptoms he/she entered. Since the system uses data from the system database the algorithm is based on data interactions.

```
user ← user of the session
medical_profile [] ←Ids of diseases in user's
medical profile
symptom_ids [] ←array containing the ids of
user entered symptoms
predicted_diseases [] ←ids of diseases with
the entered symptoms fetched from the database

for each (disease in predicted_diseases[] )
{
    disease.probability = 10

    pre_req[]=disease.prerequisites ← fetched
from the database

if(user.gender != disease.gender)
{
    remove disease form predicted_diseases[]
}

if(user.age is within disease.age_group±5)
{
    dif = |disease.age_group - user.age|
    disease.probability = disease.probability
- dif
}

if(user.age is not within disease.age_group)
{
    remove disease form predicted_diseases[]
}

for each (condition in pre_req[] )
{
  if(condition not in medical_profile[] )
  {
     disease.probability =
disease.probability-1
  }
}
}
```

Fig. 4. Algorithm for disease predicting

In the disease prediction algorithm, initially all the diseases with the provided symptoms will be fetched from the database and will be given a probability of 10 for each disease. Thereafter comparing the relevant criteria of the disease and the user the probability of the disease will be adjusted or that particular disease will be removed from the predicted disease list.

Most of the other algorithms used in the system are primarily based on interactions with the database. Either to fetch data from the database by filtering through different benchmarks or else to persist new data in the database.

C. Main Interfaces

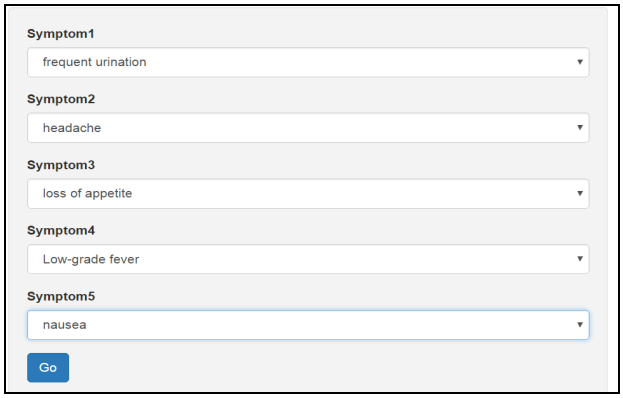

Fig. 5. Enter Symptoms Page

Figure 5 shows the interface where patients can enter the symptoms. They can enter 1 to 10 symptoms by selecting symptoms from the pick lists. Once submitted the form, as displayed in the interface shown in Figure 6. possible diagnoses are displayed to the user.

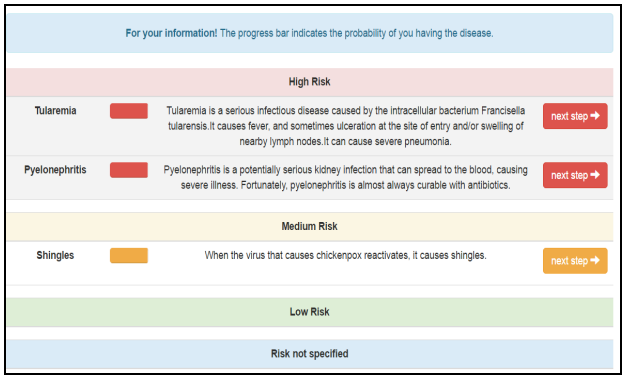

Fig. 6. Possible Diagnoses Page

The "next step" button in the interface shown in Figure 6, takes user to the page where they can find information regarding the doctors for that particular disease. There is a button that would navigate the users to the page which contains information about the hospitals where that doctor can be consulted.

The user interfaces available for the users with the role system admins are primarily to update data in the database. The user interface shown in Figure 7. is for the admins to evaluate the user-provided information. Information is displayed in an order where the most recent one comes first and they have been categorized by the evaluated state of the information. Just by pressing a button admin can mark whether the information is a success or a failure. If the 'success' button is pressed a response email is sent to the user who provided that information.

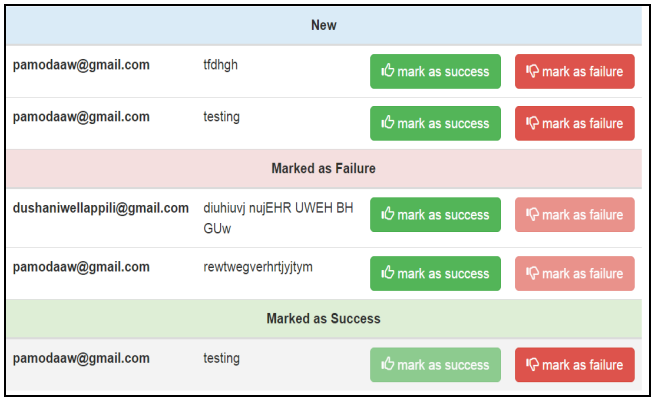

Fig. 7. Evaluate User Provided Info Page

V. System Testing and Analysis

The test plan [5] of the project covers the major components of the system developed. The subsystems in the system including symptom checker sub system and the information management sub system needs to be tested for verification.

The scope of the testing was to cover the system in order to make sure the system is delivering according to the agreed user requirements. UI testing was intended to include checks to make sure all the information is included based on the user requirement specification. Unit tests were intended to test and cover the functionality of each component at the development state. The unit testing has been carried out by the PHPUnit library which comes integrated with Symfony framework.

The performance testing is done in order to test the performance of the system including response times, transaction rates and other time sensitive requirements. The main objective of the performance test is to exercise and test the behavior of functional transactions with respect to the performance measuring requirements. In order to verify the performance, the system should be tested with normal anticipated workload as well as the anticipated worst-case workload. The technique used to achieve the performance testing is an enhanced version of the user test procedures. The test cases developed for the function and business cycle testing are used modifying the data sets to increase the number of iterations that occur in each transaction. The scripts have run on one machine and repeated with multiple clients. In performance profiling testing, Selenium IDE can be used as a test script automation tool. Rational Quantify has been used as an application performance profiling tool. The success criteria varied depending on the number of users and number of transactions. The success criteria for single transaction or single user are the successful emulation of the transaction scripts and workload without any failures.

The security of the system was provided with user authentication through encoding password with bcrypt encoder. And access controlling was done depending on the user role from each and every controller of the system with the use of libraries come bundled with Symfony framework.

In application level security the user is provided with the permissions to access only the functions and data based on their role, patient or admin. The patient has access to a certain set of functions including the symptom checking, whereas the admin has overall accesses including the information management. These accesses are to be tested in the application level security.

VI. Conclusions and Future Work

The developed system provides help for patients with the self-diagnosis or self-triage by means of considering the symptoms they have and other information such as gender, age and medical history of the patient. In addition, patients can contribute information to the system which will be evaluated by the system admins. User friendly interfaces have been created for the system admins to interact with the database and update the needed data in the database. Mailing functionality has been implemented in the system to send response emails to the patients who have contributed reliable information to the system. This system includes user authentication methodologies as well as access control methodologies to allow access to each functionality based on the user type.

As per now the patients can enter symptoms from a pick list. For the future work, instead of a pick list model the system can be improved to have a human model where patients can enter symptoms in simple English and the system do a search for the symptoms with similar phrases in the system.

In conclusion the Online Disease Tracker System provides a significant support for the patients to have a better self-diagnosis compared to other symptoms. Furthermore, the developed system provides more information regarding the disease, probability of user having the disease, doctors and hospitals in need for those diseases, etc. which are not available in similar systems. In addition, managing of the system database has made lot easier by creating user friendly interfaces for the system admins to interact with the database. Thus the developed system provides a complete platform to manage information regarding health matters. For patients the system helps to understand their symptoms better and get to know a rough idea about what the diseases they may be affected with are and what the likelihood of them having each predicted disease is. The significance with this developed system is, beyond the boundaries of just providing the predicted diseases; We Care Online Disease Tracker System provides information for the patients to ease the post-diagnosis process for each predicted disease. Most importantly the system intends to provide assistance for the Sri Lankans by having information confined to Sri Lanka (i.e. diseases common in Sri Lanka, doctors and hospitals in Sri Lanka, etc.). Thus We Care Online Disease Tracker system guides the Sri Lankans for a better self-diagnosis in a successful and productive way making their awareness high and lives easier.

REFERENCES

[1]"Symptom Checker | The one the doctors use", *Symptomchecker.isabelhealthcare.com*, 2016. [Online]. Available: http://symptomchecker.isabelhealthcare.com/home/main. [Accessed: 29- Jun- 2016].

[2]"Symptom Checker from WebMD. Check Your Medical Symptoms.", *WebMD*, 2016. [Online]. Available: http://symptoms.webmd.com/#introView. [Accessed: 29- Jun- 2016].

[3]"Learn Symfony (3.1)", *Symfony.com*, 2016. [Online]. Available: http://symfony.com/doc/current/index.html. [Accessed: 29- Jun- 2016].

[4]"How to Use Gmail to Send Emails (The Symfony CookBook)", *Symfony.com*, 2016. [Online]. Available: http://symfony.com/doc/current/cookbook/email/gmail.html. [Accessed: 29- Jun- 2016].

[5]*"Test Plan, We Care Online Disease Tracker System", We Care Online Disease Tracker System, Test Plan, version 1.1*