CSCI4448: Project Part 4 Spring 2014

Alexia Newgord, Mark Lohaus
snap.colorado.edu

1. *What features were implemented and a class diagram showing the final set of classes and relationships of the system. (This may have changed from what you originally anticipated from earlier submissions)*

Implemented Features:

- All users may:
  - View the purpose of Snap4Life on the home page
  - Submit a contact form to webmaster for questions and concerns
  - Log in and out of web application (authentication/authorization is not fully functional because we do not have a list of users, but currently simulates the user authentication process)
- Emergency responders may:
  - Take/upload image with mobile device
  - Respond to suggested patient questions
  - Use an Android or Apple smartphone to access our mobile-friendly web app
  - View list of submission as well as detailed view when a record is selected
- Hospital Users may:
  - View list of patients as well as detail view of question responses, images, and general info, when selected
- Administrators may:
  - Customize responder questions

Please see class diagram in question 3.

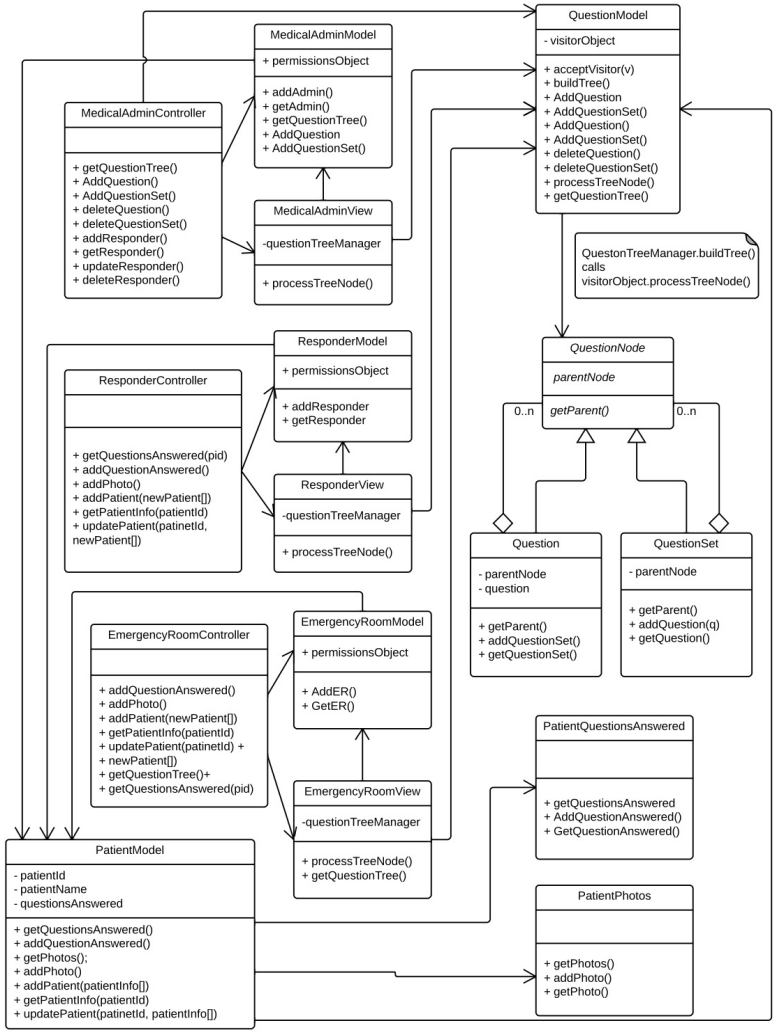
2. *Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?*

In building a tree structure of questions we were able to use the **composite pattern**. There are three places we need to leverage the question tree: When admins build the question tree, when responders answer questions about a patient and when emergency room staff view the answered questions. The html that these questions are wrapped in changes slightly for each of these rolls, so we are using the **visitor pattern** so that each of these three views passes themselves to the question model. The question model then calls that view's method which defines the HTML that is to surround the question.

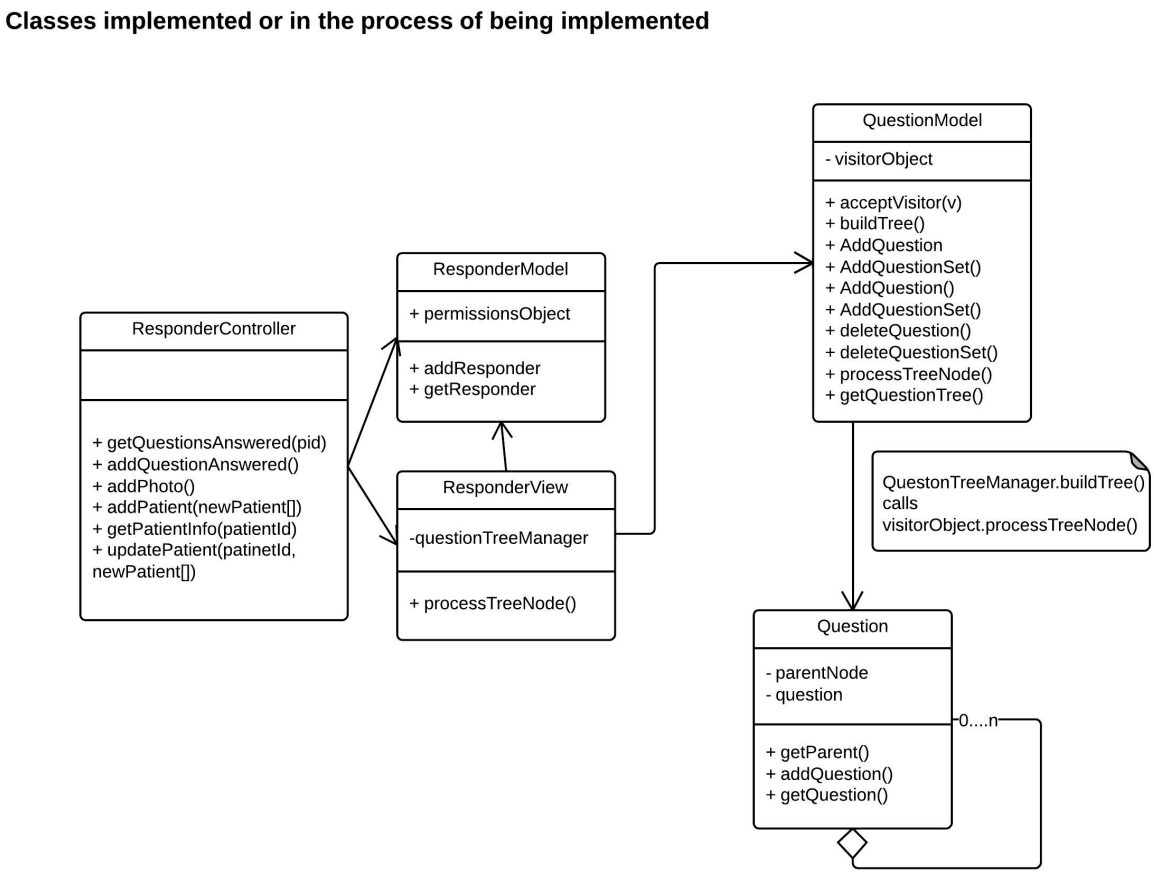
Additionally, we were able to implement a **facade** in the way that the members page of site controller was created as a user-friendly access all other controllers/modules. Lastly, the **observer pattern** became useful for the hospital controller. The default hospital controller listens to the user interaction with the CGridView subject, such that when a row is clicked, the default controller notifies its dependants and re-renders the CDetailView, image, and question tree.

3. *In addition, the report must discuss how the final system changed from the design you presented in Project Part 2. In particular, include the class diagram you submitted for Project Part 2 and use it to compare and contrast with the class diagram that represents the final state of the system.*

Class Diagram from part 2:

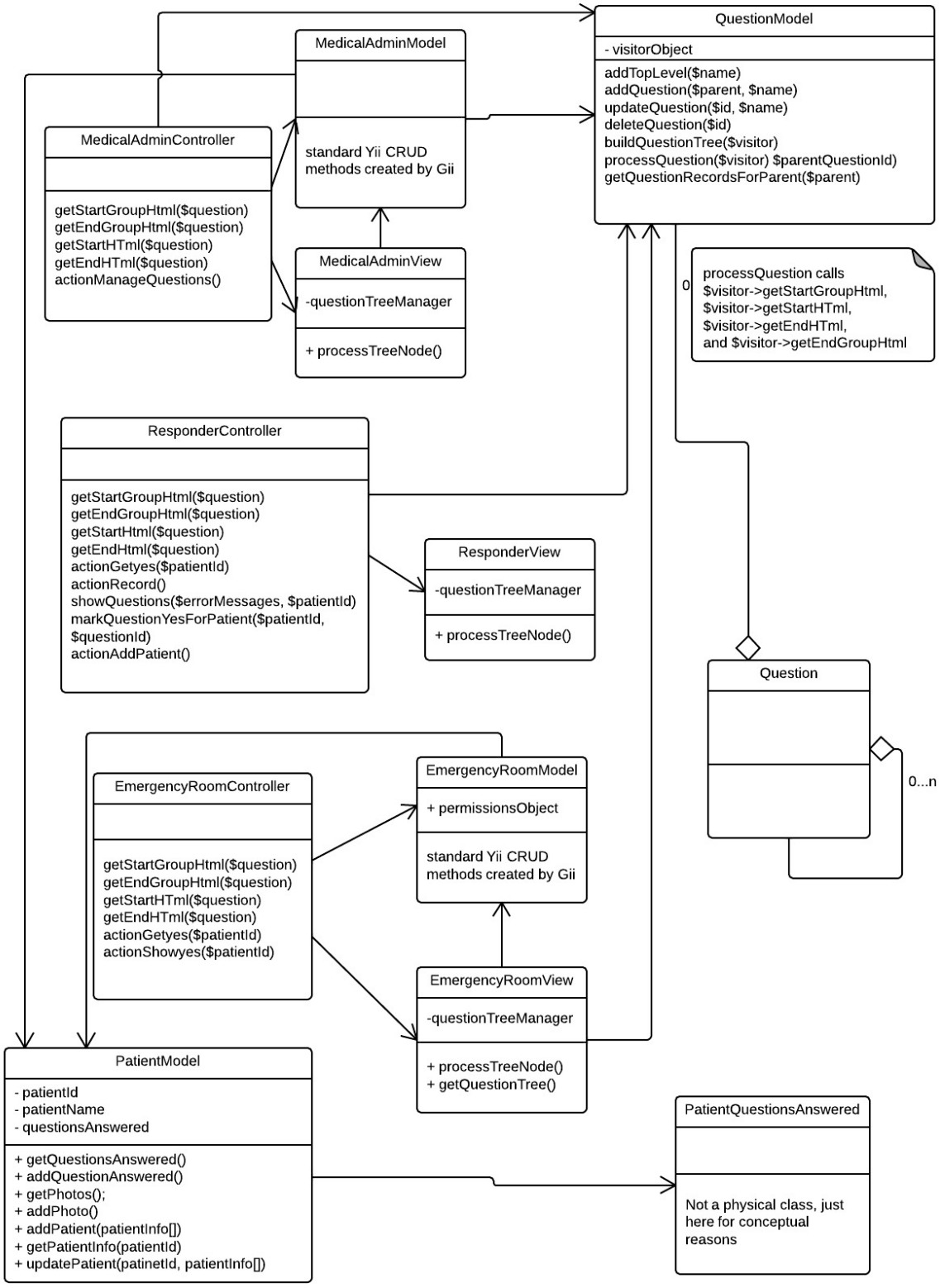


Class diagram from part 3:



Current Class Diagram:

A lot of changes happened to the class diagram. Moseley these were method name changes during coding. Also since views aren't really objects in Yii, the controllers had to be the visiting objects in the QuestionModel. We used Gii to automatically generate models and a bunch of standard methods were automatically generated for CRUD. Note that all the code to manage questions is now in QuestionModel. There is not actually a question class, physically, although it is still included on the class diagram conceptually because it helps to reinforce the idea of questions being added to questions.


QuestionModel
- visitorObject
addTopLevel($name)
addQuestion($parent, $name)
updateQuestion($id, $name)
deleteQuestion($id)
buildQuestionTree($visitor)
processQuestion($visitor) $parentQuestionId)
getQuestionRecordsForParent($parent)
MedicalAdminModel
standard Yii CRUD methods created by Gii
MedicalAdminController
getStartGroupHtml($question)
getEndGroupHtml($question)
getStartHTml($question)
getEndHTml($question)
actionManageQuestions()
MedicalAdminView
-questionTreeManager
+ processTreeNode()
0
processQuestion calls $visitor->getStartGroupHtml, $visitor->getStartHTml, $visitor->getEndHTml, and $visitor->getEndGroupHtml
ResponderController
getStartGroupHtml($question)
getEndGroupHtml($question)
getStartHtml($question)
getEndHtml($question)
actionGetyes($patientId)
actionRecord()
showQuestions($errorMessages, $patientId)
markQuestionYesForPatient($patientId, $questionId)
actionAddPatient()
ResponderView
-questionTreeManager
+ processTreeNode()
Question
0...n
EmergencyRoomController
getStartGroupHtml($question)
getEndGroupHtml($question)
getStartHTml($question)
getEndHTml($question)
actionGetyes($patientId)
actionShowyes($patientId)
EmergencyRoomModel
+ permissionsObject
standard Yii CRUD methods created by Gii
EmergencyRoomView
-questionTreeManager
+ processTreeNode()
+ getQuestionTree()
PatientModel
- patientId
- patientName
- questionsAnswered
+ getQuestionsAnswered()
+ addQuestionAnswered()
+ getPhotos();
+ addPhoto()
+ addPatient(patientInfo[])
+ getPatientInfo(patientId)
+ updatePatient(patinetId, patientInfo[])
PatientQuestionsAnswered
Not a physical class, just here for conceptual reasons

4. *What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?*

Like we were warned in class, learning frameworks has a steep learning curve. Yii is a very robust framework with a relatively small online community (especially of English-speaking users). That said, taking the extensive amount of time to understand the built-in widgets and gii code generation made the later stages of development much cleaner and faster.

We found that some of our stated requirements were unreasonable given a variety of factors; namely, the innate structure of Yii, restricted amount of time, and limited/varying skillset of the small development team. These variables did not prevent us from creating an aesthetic and useful product. However, we will be able to take these issues in consideration the next time we estimate the scale of changes that can be made within a given timeline.

The class diagrams and early designs that we created were very helpful. We were able to merge those ideas with the fundamental Yii structure for a scalable and well-encapsulated program. An example of this is our usage of Yii's module design. Using gii (Yii's automated code generator), we were able to create the main Admin, Hospital, and Responder views and controllers in a way that decoupled them from each other. This didn't exactly match our class diagrams, but followed the same principal and was highly supported by Yii. Modules are well-encapsulated, making it easy for team development, since we could work on different modules at a single time and not have to worry about code conflict.