SALUTE
WEB-BASED MEDICAL MANAGEMENT
MILESTONE 0

Musa Rayyan
Nada Hashem
Matteo Brucato
Ashwin Gopalakrishnan

February 6, 2011

Contents

Part I

Introduction

Bla bla bla...

Chapter 1

Overview

Bla bla bla..

Chapter 2

Tools and Technology

What tools and technology we used..

Part II

Requirements

Chapter 3

User Requirements

Chapter 4

System Requirements

Chapter 5

Current Status and Future Work

Part III

Design

Chapter 6

High Level View

Bla bla bla..

6.1 System Design

A Web application can be simplified as a collection of resources placed on a server which are accessible to a great number of users (clients). The access to these resources must be governed by policies and permissions. From a very general point of view, our application has two kinds of resources: *public* and *private* resources. For instance, the login page must be a public resource, whereas the setting page should be private.

You can think of a resource as either some data inside a database, a functionality that you ask to your application or even the result of some kind of operation. Based on this view, we decided to use a well-known design approach called MVC (Model-View-Controller) that works very well with this kind of settings. A controller is some executing code that performs exactly the action that the user asked to the server via an URL. In this way, a URL becomes the medium to ask for functionalities and a controller will be a merely executor. A model will be the medium to access data into a database and a view will be a result for the user, in a interface-depending fashion.

In this chapter we will describe our design from a high-level viewpoint, living the implementation for the next chapter.

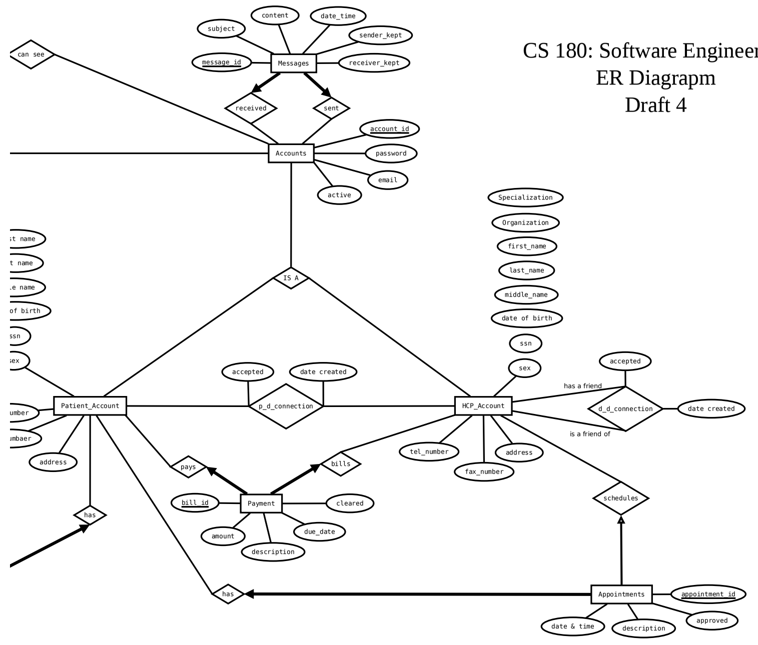
6.2 Database Design

Bla bla bla... general description ... why we chose to implement it like this... what problems we faced... etc...

6.2.1 Entity Relationship Diagram (ERD)

We provide a high-level description of the database using the famous ER diagram, for those who are familiar with it[1]. Then we will describe the database structure in more details.

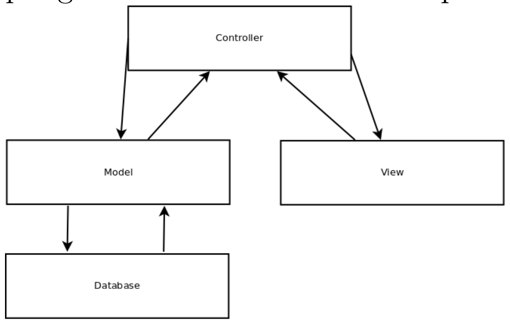
[1]Entities are represented in the ER diagram as rectangles. Each entity represents a table in the database that holds all of the information or attributes that represents that entity. In the ER diagram, each attribute is represented with a oval.

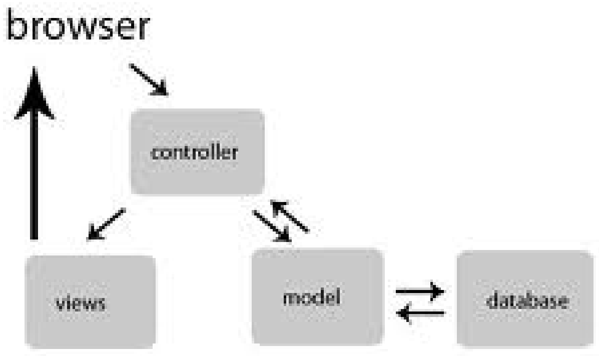


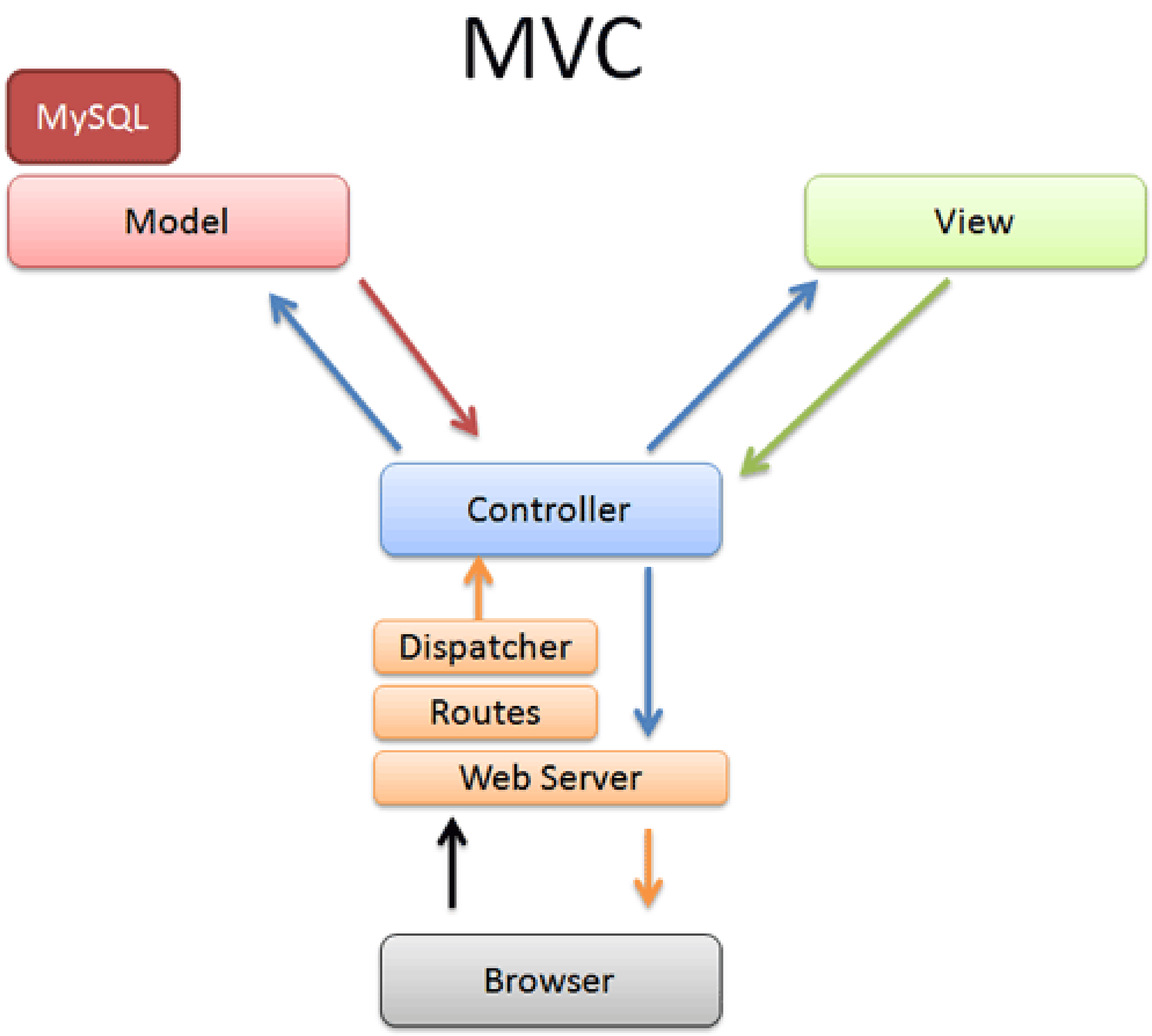
CS 180: Software Engineering
ER Diagrapm
Draft 4

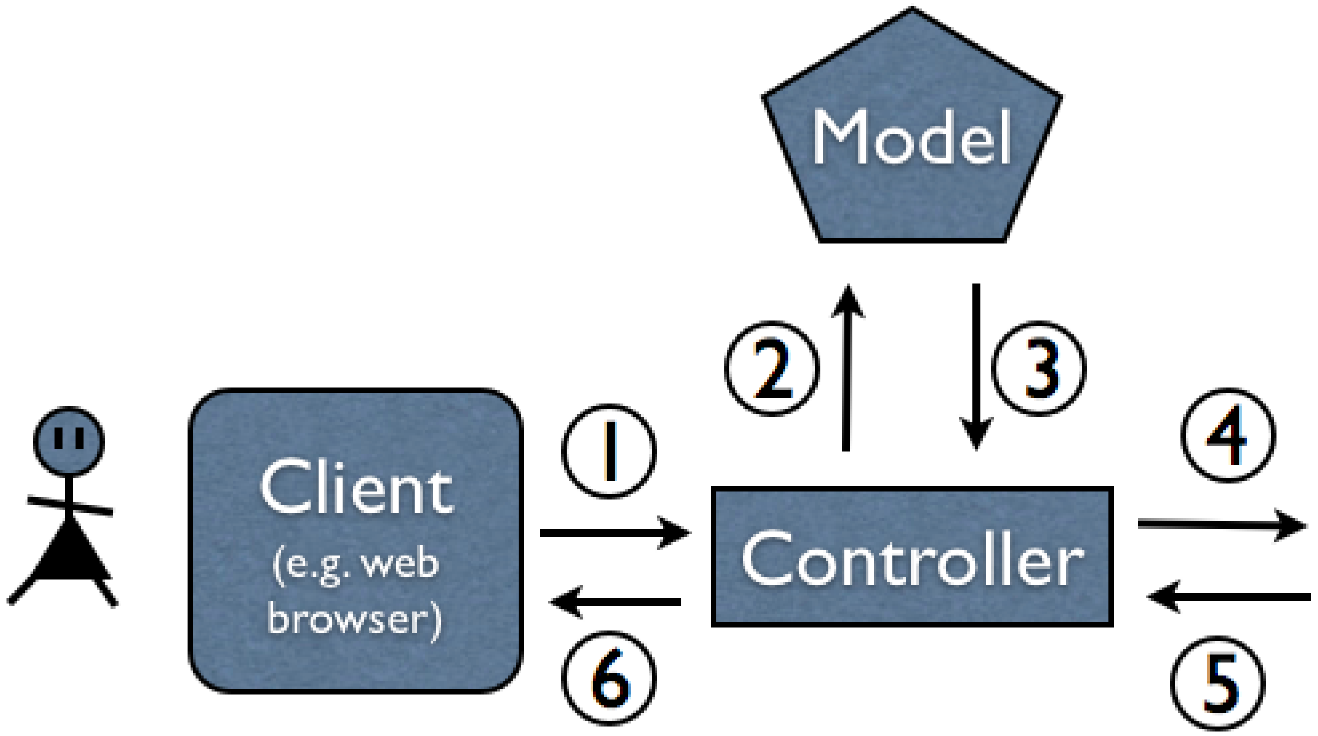
6.3 MVC Design

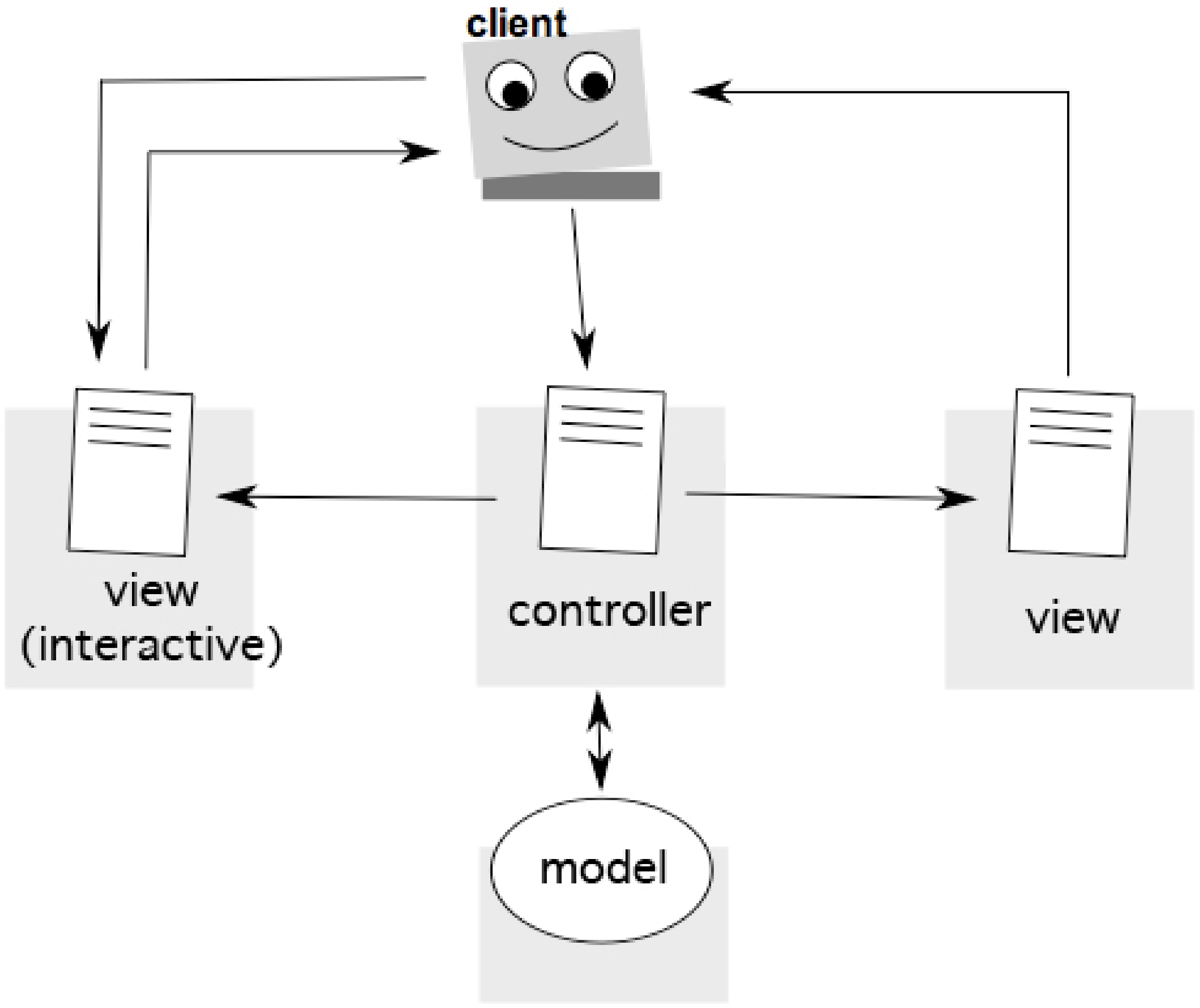
MVC stands for Model View Controller, and is a software architecture and an architectural pattern in software engineering. The purpose is to separate a system into parts,assigns responsibilities to each, and ensures that they can work together. This design method strives for high cohesion and low coupling which is essential in anticipating for future changes.






MVC
MySQL
Model
View
Controller
Dispatcher
Routes
Web Server
Browser


Model
Client
(e.g. web browser)
Controller
1
2
3
4
5
6



6.3.1 Models

The model is the tool used to access and modify the database. Everytime a user needs information, the models are queried to retrieve the information. If a user wants to add or modify, this too must be added into the database.

6.3.2 Views

The view is the the user interface, what the user sees and interacts with. In our case, its a medical management website. So, it is important to make sure that the website is user-friendly. If the layout and design of the view is difficult to navigate and/or inconsistent, the user will not desire to use the product.

6.3.3 Controllers

The controller is the middle man. It's in charge of loading the view(webpage) for the user, and calling the corresponding models to execute the functionality that the user has selected via the view.

6.4 Interface Design

Since this is a Web application, the interface is basically a set of HTML pages that a client gets from the server as it requests them. As for every Web page, the interface is a mixture between different technologies, like HTML, CSS, Javascript and so forth. It is also dependant in some degree by some server features. To keep everything simple, the interface design only defines a very high view of the user system. To allow for a better separation of concerns, we also divided the "content" of the Web page from the "presentation" of it.

6.4.1 UI design

The GUI is composed by:

- a header
- a navigation bar
- a footer
- two *dynamic* panels
    - a main panel to display the main content of the page
    - a side panel to display additional information about the main content

The header basically contains the branding. The navigation bar allows a user to navigate between all the *public* resources of the entire application and the footer has an extra set of public resources, not directly related to the use of the Web site.

The two panels are *dynamic* in the sense that their content depends on the current resource the user is accessing on a particular moment. They change during the application use, request after request.

6.4.2 Ajax

To follow a generic principle in Software Engineering, we decided to anticipate the change, creating our system capable to do every request using the famous technology Ajax. To be more precise, using requests to server called XMLHTTPRequest. Our interface is ready for that change, and the two dynamic panels are the ones that would change after a XHR request. Both the client and the server are aware of this and whenever the server receives an XHR request, it will answer with only the content for the two dynamic panels. Whereas, if the request is a normal HTTP request, it will provide the whole interface to the client. We will discuss later how we implemented it.

6.4.3 Layout

It's really hard to set a limit between content and presentation in an environment governed by HTML, but something can be done. We created all our views regardless of presentation aspects, focusing only on the most simple aspects of HTML. Our purpose is to separate this two concerns and leave to the *layout* the task to add colours, change positions of the elements and this kind of things. Doing so we also provided the possibility to change the layout in the future, or even dynamically.

6.5 Server Design

6.6 Server Design

Chapter 7

Implementation View

[doxygen here..]

7.1 Database implementation

7.1.1 Tables

We will list all the tables... bla bla bla... for each table we will describe all the attributes... bla bla bla...

Messages

Holds all of the information regarding messages sent from patient to hcp or vice versa. It has two total 1:N relationships with the Accounts entity.

serial *message_id*- ID to uniquely identify the message from other messages. serial datatype automatically creates the message_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

text *subject*- Subject of the message being sent. text datatype allows unlimited number of characters. Cannot be null.

text *content*- Where the sender can writte what they would like to send to the receiver. text datatype allows unlimited number of characters. Cannot be null.

serial *message_id*- ID to uniquely identify the message from other messages. serial datatype automatically creates the message_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

timestamp *date_time*- Date and time of when the message is sent. timestamp datatype format YY:MM:DD HH:MM:SS. Cannot be null.

boolean *sender_kept*- To determine if the sender would like to delete the message from their outbox. boolean value is either true or false. Cannot be null. By default it is true. Changing the status to false means it gets deleted.

boolean *receiver_kept*- To determine if the receiver would like to delete the message from their inbox. boolean value is either true or false. Cannot be null. By default it is true. Changing the status to false means it gets deleted.

Accounts

Holds all of the primary information every patient and hcp account needs to log into Salute. The entities Patient_Account and HCP_Account both inherit from Accounts using an IS A relationship. It has a partial N:1 relationship with the Permission and Medical_Records entities.

serial *account_id*- ID to uniquely identify the account from other accounts. serial datatype automatically creates the account_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

varchar(40) *email*- Email of the account holder. It is used to log into Salute along with the user password. varchar(40) datatype allows for a maximum of 40 characters. Cannot be null.

varchar(15) *password*- Password of the account holder. It is used to log into Salute along with the user email address. varchar(15) datatype allows for a maximum of 15 characters. Cannot be null.

boolean *active*- To determine wheather the account is active or not. boolean datatype value is either true or false. By default it is true. Changing the stauts to false means the account gets deactivated.

Patient_Account

Holds all of the personal information for every patient. It inherits from the Accounts entity with an IS A relationship. It has a partial N:1 relationship with the Medical_Records entity and a partial N:M relationship with the p_d_connection relationship.

serial *account_id*- ID to uniquely identify the account from other accounts. serial datatype automatically creates the account_id when a new tuple is inserted into the table. This ID is inherited from the Accounts entity. Primary key of the table. Cannot be null.

varchar(30) *first_name*- First name of the patient. varchar(30) datatype allows for a maximum of 30 characters. Cannot be null.

varchar(30) *last_name*- Last name of the patient. varchar(30) datatype allows for a maximum of 30 characters. Cannot be null.

varchar(30) *middle_name*- Middle name of the patient. varchar(30) datatype allows for a maximum of 30 characters.

numeric(9,0) *ssn*- Social Security Number of the patient. numeric(9,0) datatype allows exactly 9 numeric characters. Cannot be null.

date *dob*- Date of Birth of the patient. date datatype is of the format YY:MM:DD. Cannot be null.

char(6) *sex*- Sex of the patient. char(6) datatyep allows for a maximum of 6 characters. It has to be either ”male” or ”female”. Cannot be null.

varchar(11) *tel_number*- Primary telephone number of the patient. varchar(11) datatype allows a maximum of 11 characters.

varchar(11) *fax_number*- Fax number of the patient. varchar(11) datatype allows a maximum of 11 characteres.

text *address*- Primary address of the patient. text datatype allows unlimited number of characters.

HCP_Account

Holds all of the personal information for every hcp. It inherits from the Accounts entity with an IS A relationship. It has a partial N:1 relationship with the Appointments and Payment entities, as well as a partial N:M relationship with the p_d_connection and d_d_connection relationship.

serial *account_id*- ID to uniquely identify the account from other accounts. serial datatype automatically creates the account_id when a new tuple is inserted into the table. This ID is inherited from the Accounts entity. Primary key of the table. Cannot be null.

varchar(30) *first_name*- First name of the hcp. varchar(30) datatype allows for a maximum of 30 characters. Cannot be null.

varchar(30) *last_name*- Last name of the hcp. varchar(30) datatype allows for a maximum of 30 characters. Cannot be null.

varchar(30) *middle_name*- Middle name of the hcp. varchar(30) datatype allows for a maximum of 30 characters.

numeric(9,0) *ssn*- Social Security Number of the hcp. numeric(9,0) datatype allows exactly 9 numeric characters. Cannot be null.

date *dob*- Date of Birth of the hcp. date datatype is of the format YY:MM:DD. Cannot be null.

char(6) sex- Sex of the hcp. char(6) datatyep allows for a maximum of 6 characters. It has to be either "male" or "female". Cannot be null.

varchar(11) *tel_number*- Primary office telephone number of the hcp. varchar(11) datatype allows a maximum of 11 characters.

varchar(11) *fax_number*- Primary fax number of the hcp. varchar(11) datatype allows a maximum of 11 characteres.

text *specialization*- What the hcp specializes in. text datatype allows unlimited number of characters.

varchar(30) *org_name*- Name of the organization for which the hcp works for. varchar(30) datatype allows a maximum of 30 characteres.

text *address*- Primary address of the hcp place of business. text datatype allows unlimited number of characters.

Appointments

Holds all of the information for every appointment a patient makes with a hcp. It has a total 1:N relationship with the HCP_Account and Patient_Account entities.

serial *appointment_id*- ID to uniquely identify the appointment from other appointments. serial datatype automatically creates the appointment_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

serial *patient_id*- Unique account ID of the patient that requests the appointment. This is the foreign key to the Patient_Account entity. Cannot be null.

serial *hcp_id*- Unique account ID of the hcp that receives the appointment request. This is the foreign key to the HCP_Account entity. Cannot be null.

text *descryption*- Description of the appointment that the patient requests to the hcp. text datatype allows unlimited number of characters. Cannot be null.

timestamp *date_time*- Time and day of the appointment the patient requestes to the hcp. timestamp datatype of the form YY:MM:DD HH:MM:SS. Cannot be null.

boolean *approved*- Status of the appointment that the patient requests to the hcp. boolean datatype value is either true or false. By default it is false. HCP can accept the appointment and change the status to true.

Medical_Record

Holds all of the information for every medical record a patient has on Salute. It has a partial N:1 relationship with the Permission entity and a total 1:N relationship with the Accounts and Patient_Account entities.

serial *medical_rec_id*- ID to uniquely identify the medical record from other medical records. serial datatype automatically creates the medical_rec_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

serial *patient_id*- Unique account ID of the patient that owns the medical record. This is the foreign key to the Patient_Account entity. Cannot be null.

serial *account_id*- Unique account ID of the user(patient/hcp) that uploads the medical record. This is the foreign key to the Accounts entity. Cannot be null.

text *issue*- What the medical record deals with. text datatype allows unlimited number of characters. Cannot be null.

text *suplementary_info*- Any suplementary infomation that anybody (patient/hcp) would want to add to the medical record. text datatype allows unlimited number of characters.

text *file_path*- Path where the file can be found and downloaded from the server. text datatype allows unlimited number of characters. Cannot be null.

Payment

Holds all of the information for every bill that a patient receives and a hcp issues. It has a total 1:N relationship with the Patient_Account and HCP_Account entities.

serial *bill_id*- ID to uniquely identify the bill from other bills. serial datatype automatically creates the bill_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

serial *patient_id*- Unique account ID of the patient that received the bill. This is the foreign key to the Patient_Account entity. Cannot be null.

serial *hcp_id*- Unique account ID of the hcp that issued the bill. This is the foreign key to the HCP_Account entity. Cannot be null.

decimal(9,2) *amount*- The amount due to the hcp. decimal datatype allows charge to be up to 9 digits long, with 2 digits of percision. Cannot be null.

text *descryption*- Descryption of what the bill is being issued for. text datatype allows unlimited number of characters. Cannot be null.

timestamp *due_date*- Date by which the bill must be paid by. timestamp datatype of the form YY:MM:DD HH:MM:SS. Cannot be null.

boolean *cleared*- States wheather the bill was paid or not. boolean datatype value is either true or false. By default it is false. If patient pays the bill, its status is changed to true.

Permission

Holds information regarding which medical records a hcp that is connected with a patient can view. It has a total 1:N relationship with the Accounts and Medical_Records entities.

serial *permission_id*- ID to uniquely identify the permission from other permissions. serial datatype automatically creates the permission_id when a new tuple is inserted into the table. Primary key of the table. Cannot be null.

medical_rec_id- Unique ID of the medical record that a hcp can view. This is the foreign key to the Medical_Records entity. Cannot be null.

serial *account_id*- Unique ID of the hcp that can view the medical record. This is a foreign key to the Accounts entity. Cannot be null.

date *date_created*- Date in which the patient allowed the hcp to view the medical record. date datatype is of the form YY:MM:DD. Cannot be null.

p_d_connection

Holds all of the information for every connection between a patient and a hcp. This relationship has a patial N:M relationship with the HCP_Account and the Patient_Account entities.

serial *patient_id*- Unique account ID of the patient that establishes a connection with a hcp. The combination of patient_id and hcp_id is the primary key for this table. This is also the foreign key to the Patient_Account entity. Cannot be null.

serial *hcp_id*- Unique account ID of the hcp that accepts the connection request sent from the patient. The combination of hcp_id and patient_id is the primary key for this table. This is also the foreign key to the HCP_Account entity. Cannot be null.

boolean *accepted*- States wheather the request was accepted by the hcp. boolean datatype value is either true or false. By default it is false. If hcp accepts the request, its status is changed to true.

date *date_connected*- Date in which the request was sent by the patient to the hcp. date datatype is of the form YY:MM:DD. Canot be null.

d_d_connection

Holds all of the information for every connection between a hcp and another hcp. This relationship has two patial N:M relationships with the HCP_Account entity.

serial *requester_id*- Unique account ID of the hcp that establishes a connection with another hcp. The combination of requester_id and accepter_id is the primary key for this table. This is also a foreign key to the HCP_Account entity. Cannot be null.

serial *accepter_id*- Unique account ID of the hcp that accepts the connection request sent from hcp. The combination of accepter_id and requester_id is the primary key for this table. This is also a foreign key to the HCP_Account entity. Cannot be null.

boolean *accepted*- States wheather the request was accepted by the hcp. boolean datatype value is either true or false. By default it is false. If hcp accepts the request, its status is changed to true.

date *date_connected*- Date in which the request was sent by the hcp to the other hcp. date datatype is of the form YY:MM:DD. Canot be null.

Scripts to start the server and load the data

We used many scripts to create the database described above. First of all we have a bash script called start_everything.sh which as its name implies, starts everything. It starts the PostgreSQL server, creates a PostgreSQL database, and then it uses an sql file called create_tables.sql to create all of the tables described in the ER diagram. It then calls the file load_data.sql which loads all of the tables with test data. In case we want to drop all of the test data we have an sql file called drop.sql. In case we wanted to delete everything and start over, we have a bash script called delete_everything.sh.

7.2 MVC implementation

[doxygen here...]

7.2.1 Assumptions

- Doctors cannot delete medical records even if they uploaded them, only patients can
- Both doctors and patients can upload medical records
- When a doctor uploads a medical record, he has automatic viewing priveleges
- doctors cannot request appointments
- patients cannot accept an appointment
- doctors aren't allowed to reschedule an appointment
- both doctors and patients can cancel an appointment

7.3 Interface implementation

7.3.1 Layouts

Client-side and server-side... how the interface design is handled in the server and how XHTML and CSS are used to provide the interface. Talk about class "Layout" server-side.

7.3.2 JQuery

Give fancy effects... handle Ajax...

7.3.3 Ajax interaction between client and server

Talk about class=”ajaxlink” and JQuery triggers. Talk about class ”Ajax” server-side.

7.4 Tests

7.4.1 Controller Tests

There are three types of users: non-members, patients, and health care providers. Each type have been tested individually.

A non-member should only be able to view the default home page, or register. All other functions were tested to assure that a non-member could not access any other functionalities.

A patient is able to do the following:

- Login, Logout
- Requesting a connection with a healthcare provider
- Delete a connection with their healthcare provider
- Viewing all, pending, or connected healthcare providers
- Viewing their medical records
- Make an appointment with a connected doctor
- Cancel an appointment
- View all,upcoming, or past appointments
- Change their email, or password
- Retrieve their password if forgotten via email
- Edit their information
- Deactivate, reactivate their account
- View all, current, or past bills.

- Pay their bills (Note: This is not linked to any credit card/bank system)
- Add or delete a medical record
- View all their medical records
- Set each medical record to hidden or public to specific healthcare providers
- Send a message to doctors

A health care provider is able to do the following:

- Login, Logout
- Requesting a connection with a healthcare provider
- Accept a connection request from another health care provider or patient
- Reject a connection request from another health care provider or patient
- Delete a connection with their patient or collegue
- Viewing all healthcare providers
- Viewing pending incoming requests with other healthcare providers
- Viewing pending outgoing requests with other healthcare providers
- Viewing connected collegues, and patients
- Accept a requested appointment from their patients
- Cancel an existing appointment
- View all,upcoming, and past appointments
- Change their email or password
- Retrieve their password if forgotten via email

- Edit their information
- Deactivate, and reactivate their account
- View all, current, or past bills.
- Issue bills to connected patients
- Add a medical record to a specific patient
- Viewing their patient's medical records (the ones they are authorized to see)
- Set each medical record to hidden or public to specific healthcare providers
- Send a message their patients or collegues

7.4.2 Database Tests

In order to test the database schema and the Model View Controller design implementation we had to load the database with test data. The test data is formated exactly so that it fits each database table schema. There are ten different files with test data(approximately one for every table):

accounts.txt:

- Test login
- Test creation of an account (patient or healthcare provider)
- Test if account is active or not
- Test account updates
- Test creation of a medical record
- Test permissions to a medical record

appointments.txt

- Test creation of an appointment

- Test acception of an appointment
- Test cancelation of an appointment
- Test rescheduling of an appointment
- Test if an appointment belongs to a patient
- Test if an appointment belongs to a healthcare provider
- Test if we can view all, upcoming, and past appointments for a patien
- Test if we can view all, upcoming, and past appointments for a heath-care provier

d_d_connection.txt

- Test if we can see all of the colleagues of a doctor
- Test the incoming requests a heathcare provider gets from other health-care providers
- Test the outgoing requests a healthcare provider has to other healthcare providers
- Test if a healthcare provider can add, accept, and delete another health-care provider
- Test if a healthcare provider can delete pending requests
- Test if a healthcare provider is connected to another healthcare provider

payment.txt

- Test if a bill belongs to a patient
- Test if a bill was issued by a healthcare provider
- Test if we can view all, upcoming, and past bills for a patient
- Test if we can view all, upcoming, and past bills issued by a healthcare provider
- Test if we can view bills that have been paid as well as not paid

- Test if a healthcare provider can issue a bill to a patient
- Test if a patient can pay for a bill
- Test if only a healthcare provider can delete a bill

Part IV

Operating Manual

Chapter 8

How-to's

Chapter 9

Registration and login

Chapter 10

Viewing a user profile

Chapter 11

Connection management

Chapter 12

Screen-shots

Part V

Credits