Sanyuegongyi Charity System

Table of Contents

Definition, investigation and analysis

1. Definition
    - The problem I aim to solve is the database management of a charity, which helps poor students go to school.
    - Description of the organization: The Charity, called Sanyuegongyi, is a charity which takes money from the donator and distributes the money to schools in a poor area in China. What's special about this charity is that a donator can help a specific student, and get to know his or her information specifically. The charity would pay for the tuition, books, traffics, as well as lunch for students. Moreover, the charity is also responsible for showing the current status of students to the donators and the public. Obviously, such an organization requires a management system to manage cash flow and provide essential information to the public. The charity currently has around 5 staff members, 60 donators and 4 schools in coorperation.
    - Methods currently in use: Currently, the management is purely based on email, phone calls, text messages, and Excel tables: First, the donators call or email the manager of the charity, given a card number to transfer money and a list of students that he or she can help. Then, the manager will put this donator into a chatting group on the Internet (based on an app called WeChat, a chatting tool used by most Chinese people), through which the charity can share information of the process of charity or status of students. After that, the manager will contact each school in the poor area and transfer the money to the school, giving a list of students who will be helped. The the excel table, the manager has to record each transfer of money and the status of each students, updating them every week manually.
    - Origin of the data: The data comes from the email sent from the donators and schools. Both schools and donators send the information necessary to each other through the managers of the charity. The data are stored in folders in the manager's computer, ordered by school names and donator names.
    - Form the data takes: the manager has a few forms in excel to store data, about donators, schools, children and donation information. The forms are all organized with the title on the top, and the names of each record on the left. An example of the "donator" form is shown below:

Donators				
Name	Number of donations	Wechat account	E-mail	Phone number
Siyun Gong				
Juan Liu				
Peng Liu				
Riyong Zhang				
Fuzhou Lv				
Zhixiang Zhang				
Haitian Huang				
Jizhou He				
Kun Wang				
Hubo Zhang				
Jianjun Li				

2. Investigate the current system

As a core member in the charity, the initiator of the organization Mrs. L offered me sufficient information to specify the requirements, as is evidenced by the following Q&A part of the interview:

Q: Nice to meet you Mrs. L.

A: Nice to meet you too.

Q: How many types of users are involved in the charity?

A: I think there are three: the donators, managers, and school managers.

Q: Lets' start with the donator. What does the donator need to do to donate to a specific student?

A: First, the donator chooses a specific student from the database provided by the charity, and then transfers money to the account of our charity.

Q: What information is a donator supposed acquire after the donation?

A: Mainly, the current status of the student he or she donated, including photos, grades and health condition. The donator will probably have the chance to communicate with the student through letters.

Q: Through what way do the donators acquire the information of the donated students?

A: Through email or WeChat currently. In the new system, we want the donators to be informed automatically by the system, when the status of a student is changed.

Q: Can a donator donate for multiple students?

A: Of course. They can donate as long as they are willing to.

Q: Can a donator decide how much money to donate to a single student?

A: Yes, but currently, we have a minimum amount to support the basic school need of each student. If more money is donated, it would be given to the student for some other purposes.

Q: Let's talk about the managers. What jobs do the managers have to do?

A: The mangers are responsible for operating and recording cash flow. Also, they ask for information about students from the school and provided it to the donators. In the new system, we want to reduce the job of manager. The cash and information flow should be automatically proceeded on the Internet.

Q: In the new system, do we still need managers?

A: Yes. We still need the managers to have access to every piece of data in the system in case of emergency situation happens, or sudden change is needed to be made.

Q: Finally, what is the function of the school managers in the system?

A: School managers take money from the charity, and upload the current status of each students. They also have access to their own name list of students in need of donation.

Q: Do the school define the minimum donation amount?

A: No. The minimum donation amount is always equal in all schools.

Q: So basically, how is the current system operated?

A: The current system is operated manually. The managers have to do everything about information flow and cash flow between the donators and the schools. It always takes the managers extremely long time to do the repetitive jobs. The managers sometimes still make mistakes, which might cause bad conflicts.

Q: Currently, how many users are there?

A: We have 2 managers, 59 donators and 4 schools operating in the system. There are 1005 children overall. After the new system comes online, we might expect the number of donators and schools to double or even triple.

Q: How frequently do a school send a message about a child?

A: It depends. Averagely, I think it is twice a week.

Q: Thank you for your patience.

3. Analysis

Basically, according to my investigation, current system solves the problem mainly by the managers.

Donators have to:

- join a WeChat group, and transfer money to the manager with bank or online payment method.
- receive the news of the children he or she donates.

Schools have to:

- receive payments from the manager, and mark the child as donated.
- frequently send news about the children by WeChat, QQ or news.

Managers have to:

- provide options to donators. If any donator wants to donate specific amount of money, a manager has to send the donator a list in excel, which shows all the children who haven't been donated. The manager also has to introduce the children, and provides some necessary information to the donators.
- perform the transactions between donators and acceptor schools. Whenever a donator makes a donation, the manager first has to record the donation in the excel file. Then, the manager will send the money to the bank accounts of the schools.
- transfer necessary information between donators and schools. Currently, the manager first takes the huge amount of information, such as photos from the schools. Then, the information is transfered to donators associated with the children/school being donated. Usually, the information is sent by chatting tools like WeChat or QQ, or by email.
- finally, the manager is reponsible the maintainence of the system. Currently, the maintainence includes backup, logging, safety control and information control. First, the manger has to backup all the information occationally, because in case of some emergency situations, all necessary information might be lost, and we need to restore quickly. Besides, the manager should be able to get access to all the information about every child, cash transfer and new users, so that some most fake information or intentional abuse of the system can be avoided. The manager should be able to delete the posted information and temporarily disable some accounts.

According to the interview, the tables in excel can be concluded as two:

- For Children:

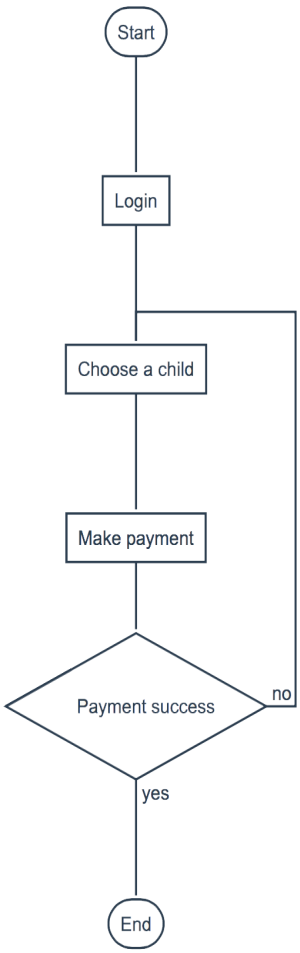
Children						
Name	**Age**	**School name**	**School e-mail**	**School phone number**	**School card number**	**Whether donated**
Zhiyuan Huang						
Zi Li						
Zhanghan Rong						
Tengyu Li						
Mingshu Sun						
Jinghe Fang						

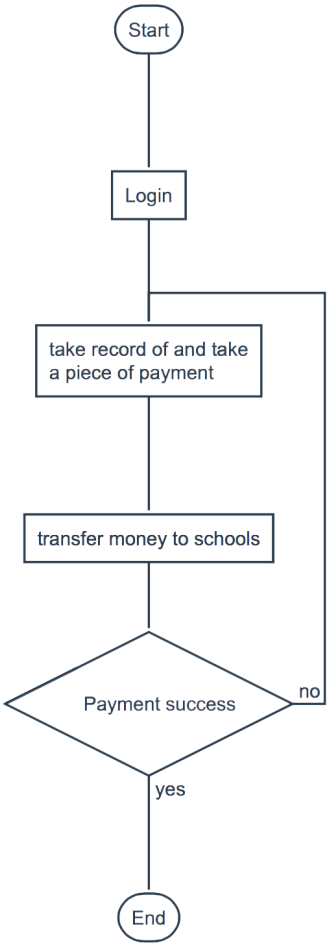
- For Dontors:

Donators				
Name	**Number of donations**	**Wechat account**	**E-mail**	**Phone number**
Siyun Gong				
Juan Liu				
Peng Liu				
Riyong Zhang				
Fuzhou Lv				
Zhixiang Zhang				
Haitian Huang				
Jizhou He				
Kun Wang				
Hubo Zhang				
Jianjun Li				

4. System flowchart

Donation process. Payment process of the manager





5. Data capture method

In the current system, data is captured through:

- The text, Wechat or email between the managers and donators, about each donation.
- The text, Wechat or email between the managers and schools, about each donation and the information about the students in each school.

As the current system is entirely manual, so inputs and outputs are only generated by the managers manually.

Inputs for schools:

- Name of schools
- Schools e-mail
- School phone numbers
- School card numbers
- Name of each child
- Age of each child

Inputs for donators:

- Name of donators
- E-mail of donators
- Phone number of donators
- Wechat account of donators

Inputs for donations:

- Amount donated
- The child donated
- Whether the money is transferred
- Whether the school receives the money

Outputs for donators:

- The information of each child donated.
- The information about the money transfer.

Outputs for schools:

- The child donated.
- The donator who donates the child.

Storage: all the data are stored in the computers of the two managers locally. They copy the data on their computers to each other if needed.

Processing: after managers receive the inputs by donators or schools, the data are put into excel tables. If the inputs are about donations, or the information/news sent by schools, the managers will forward the information to donators or schools if necessary.

6. Inefficiencies of current system

The current system does solve the problem, but it has some problems.

- It consumes unnecessary efforts of the managers. The managers work for a long time to make effort to manage all the transactions and information transfer. When currently the donators and schools make transactions or send information to the web, such interaction can be done fast by an automatic system.
- Such a manual system might go wrong if the managers make a mistake. Involving transactions of money, the whole charity might have serious troubles if such mistakes are made. Instead, with a computer system, the probability of making mistakes becomes much lower.
- The managers take the record of the current system in Excel, which means all the transactions and information are stored in flat files. Such method of storage is both inefficient and unsafe. If one is editing an entry which is involved in many different records, the editing will take a long time to modify each of these entries and the records. Also, the managers might forget easily to backup the existing data with manual storage of data in flat files locally. As a result, we need a system on web, that both store data within a dynamic database and backup and create logs of the existing data.
- There are a lot of redundancies in the excel flat file. For example, the information for a single school can be recorded for many times. It wastes space as well as efficiency of searching.
- It’s hard for users, including donators and schools, to have immediate access to information online. It takes time to send and receive information.
- Two managers have separate data storage on different computers, so they need to synthesize manually frequently. Moreoever, the discrepency in data for each managers can cause errors with users.

7. Alternative approaches

Alternative solutions were discussed with the client.

- Use an online system(e.g. Github) to synchronize the data in excel tables on both managers.
    - The problem of the discrepancy in data is partially solved with a synchronizing system. Still, the synchronization is not real-time, so before each updates, there might still be discrepancy in data.
    - The redundant efforts of the managers are not reduced.
    - The database is still not efficient in space.
    - The data is still stored in flat files, so it's slow to access the records.
    - The time for donators and schools to access to the data is still extremely long, as the information still has to be sent by the managers.
- Use Microsoft Access and Visual Basic developed software on a shared local server to replace Excel for database storage.
    - As the database is placed on a shared server, both managers have access to the database at the same time.
    - With proper development, the redundancies in database can be eliminated, and the effort of managers can be reduced.
    - In Microsoft Access, the efficiency of data access can be improved significantly.
    - The time for donators and schools to access to the data is still extremely long, as the information still has to be sent by the managers.
    - All operations in the system are still carried out by the two managers.

After the discussion of the previous possible alternatives, we found the major problem of the system is that donators and schools can not have access to the database by themselves. We came up with the final solution

- Develop a web based application with ruby and html. The application is stored in the server which is connected to the Internet. Different users have different access to the system by being identified with user name and password.
    - The system is highly secure with user names and passwords.

- The time for donators and schools to access to the data is a lot shorter, as they have their own access to the system online.
- The work of two managers is significantly reduced, as all users have their own access to the system.
- Operations like donations and enrollments can be done by donators and schools themselves, without the interaction of the managers.
- Rails, the web framework with ruby, is embedded with SQLite, a database system with efficient data access. As a result, the data can be accessed much more quickly.
- Web based application has no limitation on the device in use.
- Web based application with proper design is more user friendly.
- Development in ruby and html is more difficult than that in Visual Basic.

8. Requirement specification

(1)Database requirement

- A database must be established to store the following properties:
- Donators: donator information
- Children: child information, whether donated, donator_id
- Schools: information.

- Donator information includes: name, user_name, password, email, phone number;
- child information includes: name, age,school_id;
- school information includes: name, email, phone number, address, card_number.
- Further data types incude:
- News: child_id, title, content, date.

(2) Functions requirement

- The system must allow users to login to keep the system secure.
- The system must allow different users to do operations, and the system will update the database meanwhile.

(3) A GUI is needed for the system, for the users to view and edit data.

(4) The system must work on the Internet, with a URL that can be accessed anywhere.

(5) Software requirement

Software	Reason
A web browser	The program is developed on a web environment. Any browser can access it with its URL.
A server Cent OS/Mac Operating system	The program has to be operated on a server with a operating system.
Rails	Rails needs to be installed on the server to make sure the system can be operated.

(6)Hardware requirement for development: personal computer(mac) and a test server.

	Device	Reason
Input device	Keyboard	To enter necessary information in fields.
	Mouse	To move the pointer on the page.
Output device	Monitor	Allow users and managers to see the actions of the system.
Storage	User storage 100Gb	The database of all users information needs to be stored online.

(7)Software requirement:

- Language used: ruby, html, javascript, SQLite and css.
- Operating system: Mac OS X on mac, Linux Cent OS on the test server.
- Developement environment: Rubymine.

Design

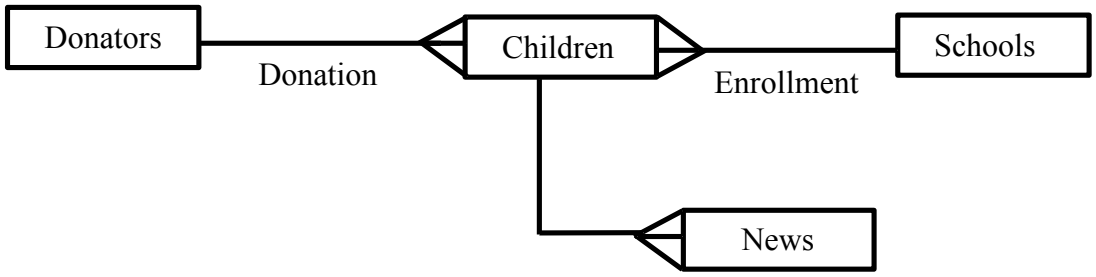
1. Objectives

After investigation and analysis, the final objectives of the system are listed as following:

- Create a database system, including the information of the donators, children and schools and the relations between different objects in the system.
- Create an graphical user interface, to allow the following:
    a. Allow donators and schools to sign up to the system with forms.
    b. Allow donators and schools to log in to the system and view the existed information available to them.
    c. Allow donators to view the news about the children they donated and the messages sent by children and schools in the system.
    d. Allow schools to upload new information about children. Schools should be able to add children to the database, and add information to each child's profil.
    e. Allow donators to donate children online.
    f. Allow schools to accept donations online.
    g. Allow donators to see a list of the children not donated, as well children donated.
    h. Allow schools to see a list of children belonging to the school.
    i. Allow the schools to edit and add children in its profile.
    j. Allow managers to have access to all the information, and to delete information or disable accounts if necessary.
    k. Allow each user in the system to edit his or her profile and password.
    l. Allow each user to click on a navigation bar on the top of the page.
- Create a security system that makes certain information only available to certain users. Users log in to the system with their account name and password.

2. Database design

Entity relationship diagram:



Donator

Field name	Type	Validation	Description	Example
id	int	presence and unique	The automatically generated ID number of each donator and school.	92
user_name	string	presence and unique	The user chooses a user_name that can be used in the file. Users login with user_name.	JohnMu
name	string	presence	The user's real name.	John
password	string	presence	The account's password.	123456
email	string	presence	The user's email address.	123456@123.com
phone_number	string	presence	The user's phone number.	123456789

School

Field name	Type	Validation	Description	Example
id	int	presence and unique	The automatically generated ID number of each donator and school.	92
user_name	string	presence and unique	The user chooses a user_name that can be used in the file. Users login with user_name.	JohnMu

name	string	presence	The school's name.	ABC High School
password	string	presence	The account's password.	123456
email	string	presence	The school's email address.	123456@123.com
phone_number	string	presence	The school's phone number.	123456789
card_number	string	presence	The school's card number.	1234567890000000
address	string	presence	The school's address	No. 20, ABC Street, Beijing, China

Children

Field name	Type	Validation	Description	Example
id	int	presence	The automatically generated ID number of each d.	92
name	string	presence	The child's real name	John
age	int	presence	The child's age	5
status	string		It is the donation status of the child. The statuses are stated below.	123456@123.com
phone_number	string	presence	The user's phone number.	123456789

There are three statuses for a child:

- "not_donated" means a child is not yet donated.
- "on_transfer" means a child is donated by a donator, but the cash has not been received by the school.
- "donated" means a child is donated by a donator and the cash is received by school

News

Field name	Type	Validation	Description	Example
id	int	presence and unique	The automatically generated ID number of each donator and school.	92
title	string	presence	The title of a news.	New news!!!
content	string	presence	The content of a news.	The child got 100

				in math exam!
child_id	int	presence	The child that the piece of news is related to.	12

Enrollment

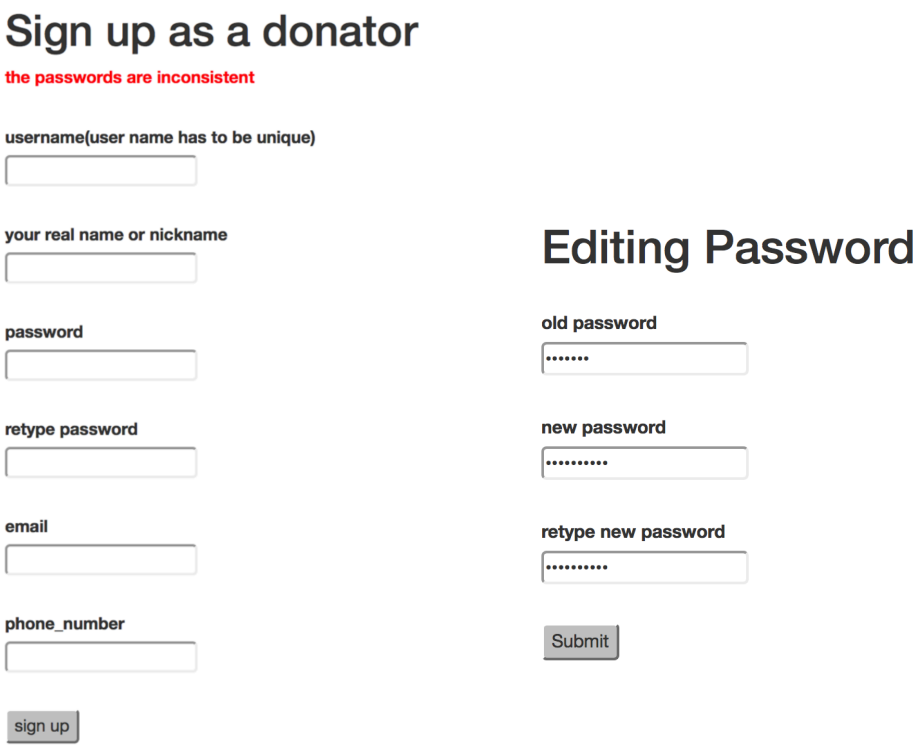
Field name	Type	Validation	Description	Example
id	int	presence and unique	The unique id of an enrollment.	12
school_id	int	presence	The school of an enrollment.	JohnMu
child_id	int	presence	The child of an enrollment.	John

Donation

Field name	Type	Validation	Description	Example
id	int	presence and unique	The unique id of a donation.	12
donator_id	int	presence	The donator of an enrollment.	JohnMu
child_id	int	presence	The school of an enrollment.	John

- There are a few things needed to be explained about the table design:
- For schools and donators, we use two primary fields, which are id and user_name. The reason for that is, in rails, any entity in a database is automatically given an id, so id should always be used to identify a donator or school. However, an id number is hard for users to remember, so we adopt another unique field called user_name, so users can edit their own user_name and make it memorable.
- For donator-children and school-children one-to-many relationship, we use two additional tables Donation and Enrollment to accelerate the search and improve the flexibility of the database.
- The database is implemented with sqlite, combined with rails.

3. Data input forms



The sign_up form serves as an example of the input form in the system. On the top, it has a bright title of the meaning of the page. Then, there is a red line containing the error message, to inform the user what he or she is doing wrong. After that, it has many text fields for the user to type in user_name, name, password, email and phone_number. For the password field, it hides its content for security reason, and requires the user to type the password for two times to avoid mistakes. Finally, we have the “sign up” button, for the user to finish the form. As the system is a web page, the user can click all “Back” on his or her browser, so the system has no “Back” button.

The next example is the editing password page. All the input pages follow the same style of design.

4. Data output design

Basically, there are two types of data output in our system.

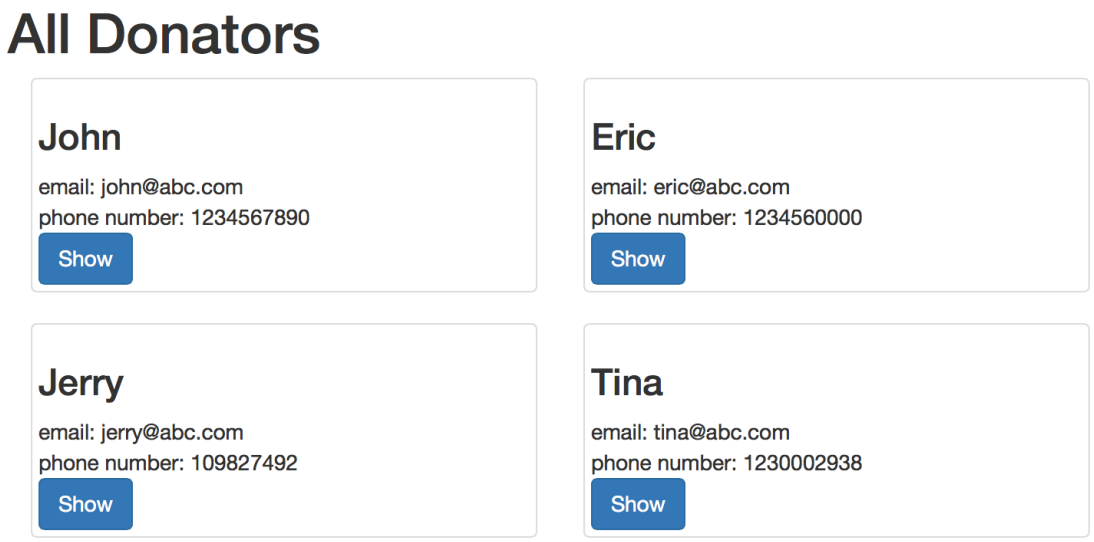
For single donator, child, school or news, we use the following design.

In the output view, the title shows the content of the page is about the donator. Then, each field of the donator is listed. Then, if the account on the page is the user using the page, he or she can see the edit buttons.

This screenshot is for a child, viewed by a school.

For a child, if the user is the school of the child, the school has the access to edit, destroy or add news to the child in the database.

The second type of data output is for multiple donators, children, schools or news. The design of a page showing all donators is as the following:



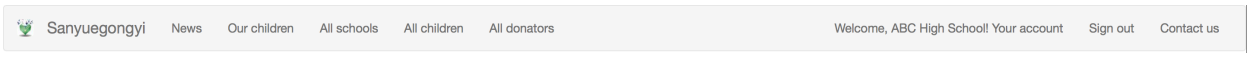
Each donator has all the information shown on this page. Also, there is a show button, to enter the single page, in which only one donator is shown.

5. Navigation bar

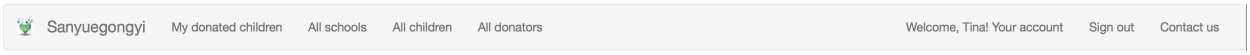
All navigation bar has the icon of “Sanyuegongyi” on the left. And then, the left part of the bar has different accesses to different parts in the database. Then, on the right part, on the links are about the personal account.

Different users will see different navigation bars.

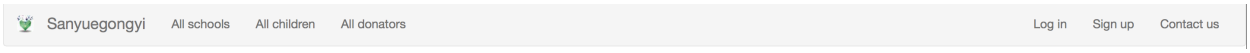
For schools:



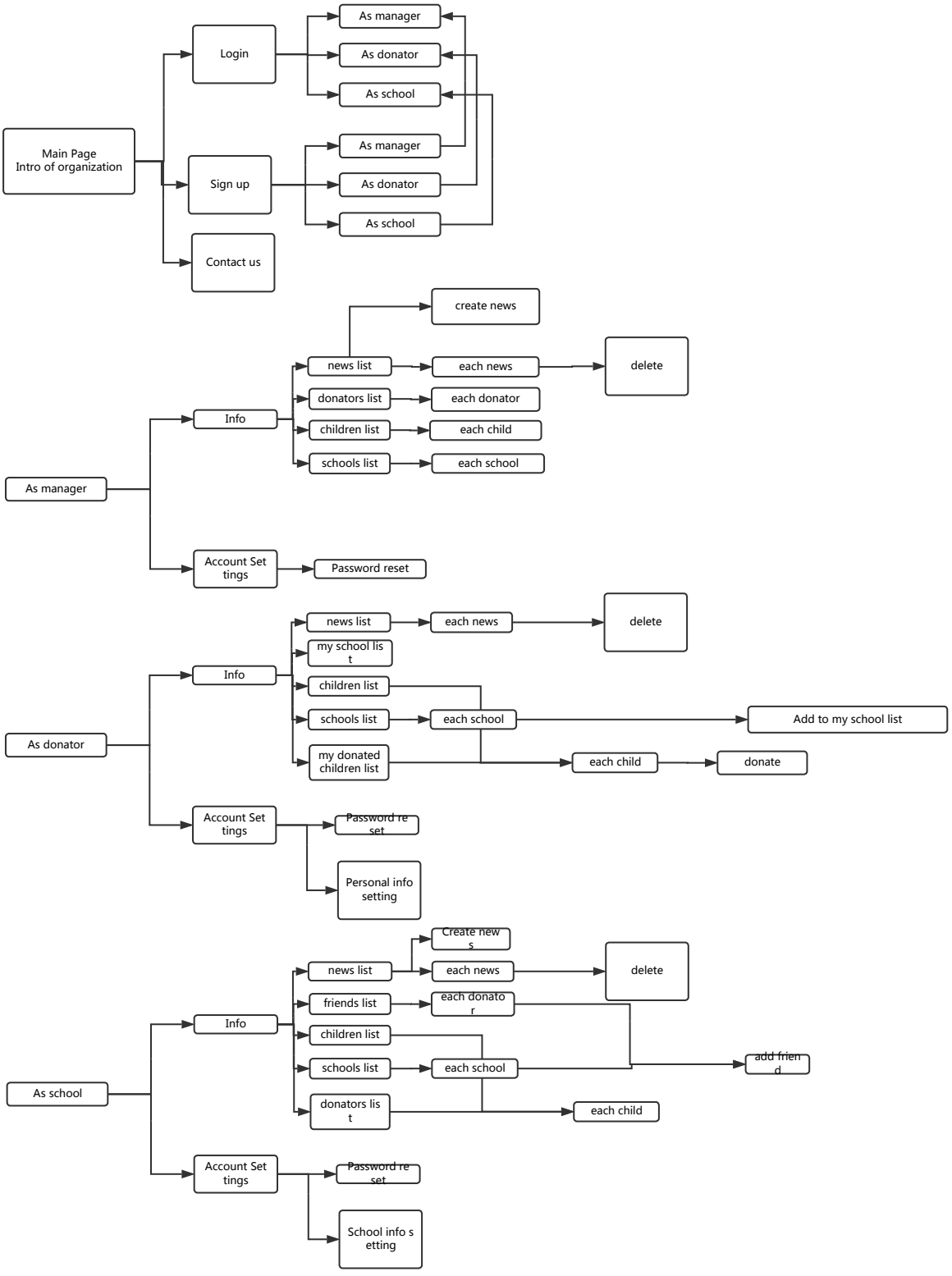
For donators:



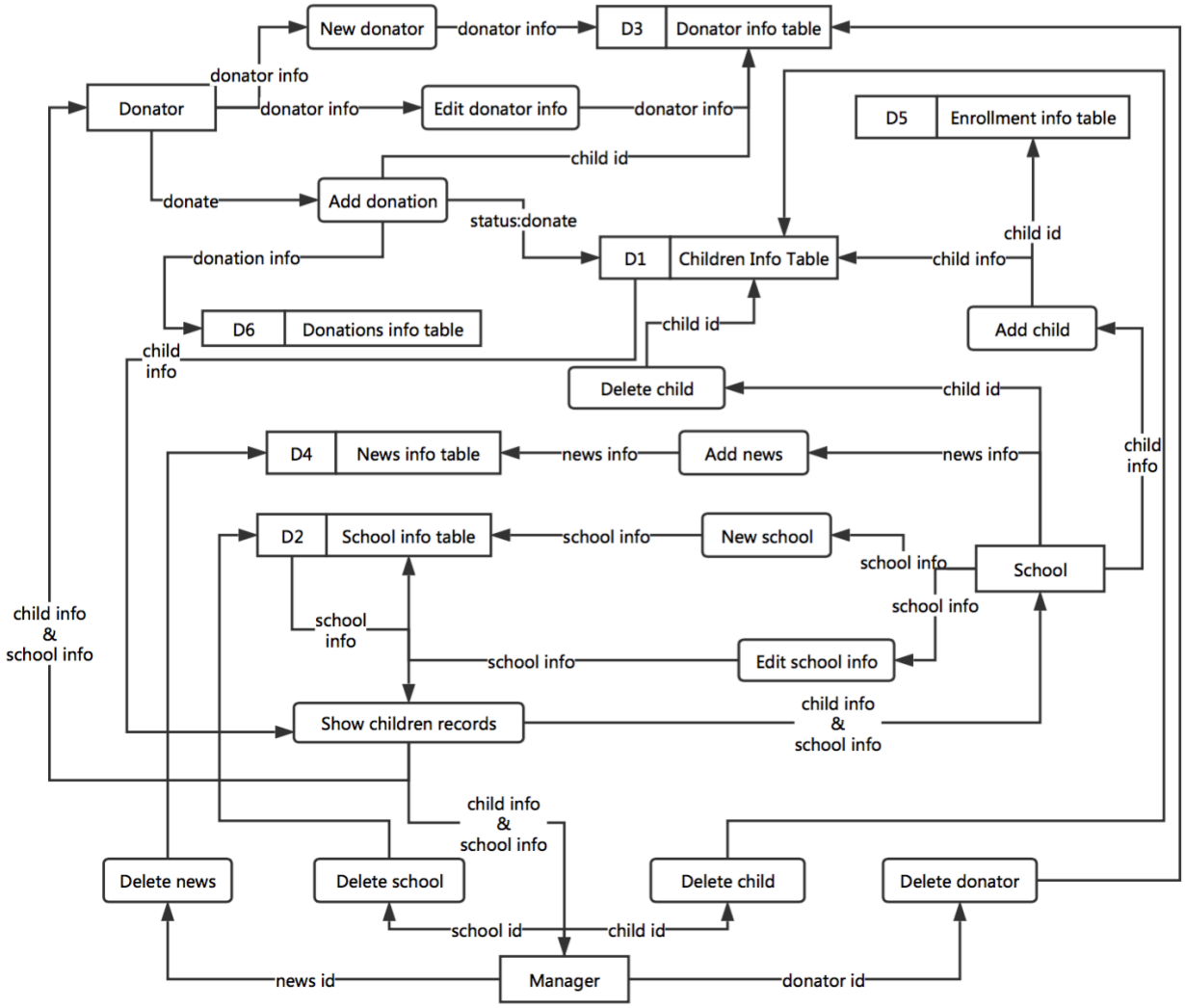
For users not logged in:



6. Navigation design



7. Data flow diagram



8. Estimation of storage size

The charity provides us with the following data from the interview:

Q: Currently, how many users are there?

A: We have 2 managers, 59 donators and 4 schools operating in the system. There are 1005 children overall. After the new system comes online, we might expect the number of donators and schools to double or even triple.

Q: How frequently do a school send a message about a child?

A: It depends. Averagely, I think it is twice a week.

So to estimate a upper bound of the storage size, we assume there are 200 donators, 30 schools and 6000 children overall. We assume the number of Donations and Enrollments are maximized(6000), and each child has n news.

Donator

Field name	Type	Example	Size(bytes)
id	int	92	4
user_name	string	JohnMu	40
name	string	John	40
password	string	123456	40
email	string	123456@123.com	40
phone_number	string	123456789	40

School

Field name	Type	Example	Size(bytes)
id	int	92	4
user_name	string	JohnMu	40
name	string	ABC High School	40
password	string	123456	40
email	string	123456@123.com	40
phone_number	string	123456789	40
card_number	string	1234567890000000	40

address	string	No. 20, ABC Street, Beijing, China	40

Children

Field name	Type	Example	Size(bytes)
id	int	92	4
name	string	John	40
age	int	5	40
status	string	123456@123.com	40
phone_number	string	123456789	40

News

Field name	Type	Example	Size(bytes)
id	int	92	4
title	string	New news!!!	40
content	string	The child got 100 in math exam!	40
child_id	int	12	4

Enrollment

Field name	Type	Example	Size(bytes)
id	int	12	4
school_id	int	JohnMu	4
child_id	int	John	4

Donation

Field name	Type	Example	Size(bytes)

id	int	12	4
donator_id	int	JohnMu	4
child_id	int	John	4

On average, we assume each string has average length 20. We use Unicode as the character set of our system, as Chinese characters are also allowed.

- Each donator: 1*integer + 7 * string(Unicode) = 284 bytes
- Each school: 1 * integer + 8 * string(Unicode) = 324 bytes
- Each child: 2 * integer + 2 * string(Unicode) = 88 bytes
- Each news: 2 * integer + 2 * string(Unicode) = 88 bytes
- Each enrollment： 3*interger = 12 bytes
- Each donation： 3*interger = 12 bytes
- Overall: 284*200 + 324*30 + 6000*88 + 6000*88*n + 6000*12 + 12*6000 = 738520 + 528000 * n = 739 + 528n Mb

For the system to operate 5 years, each child, on average, has 26*5 = 130 news

739 + 528*130 = 39379Mb = 70 Gb.

As a result, for the system to safely operate 5 years, we at least need the size of storage to be more than 70Gb.

9. Benefit of the new system

- The flat file implemented with excel is now replaced with relational database. So all database operations are faster, and the logics is better understood.
- Compared with excel, the graphical user interface makes information more visible and obvious. The structure is clear, and even if any of the user or donator makes mistake in any operation, there won't be any disaster.
- The login system substitutes and simplifies the original email or chatting process. It makes sure every member in the system has his or her own access to the system. Also, the security system keeps certain information only available to certain users.
- The system is established in web condition, so it can be accessed by any device with a web browser like smart phones and laptops.
- The navigation system builds a clear structure of all information to make sure each user only see the necessary information at a time.

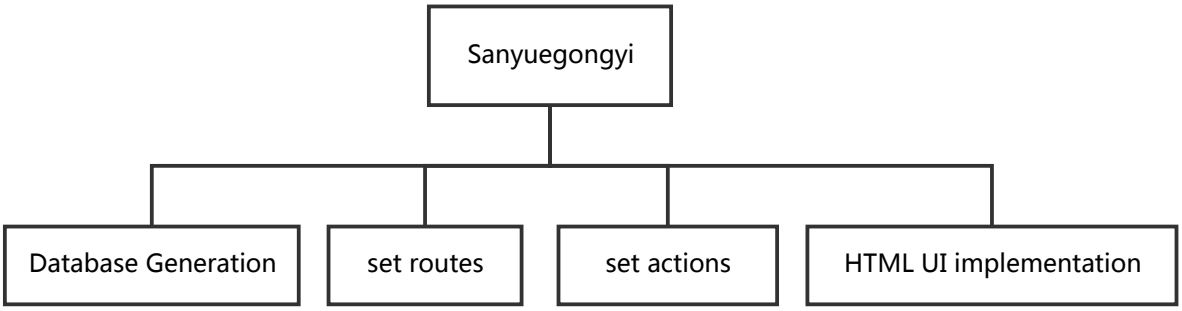
10. Limitation of the scope of the solution

The entire app is based on web, so it requires a server to be on all the time. So there are two limitations:

- If the server stops working or breaks, the system stops working.
- Backup is needed all the time. If any error happens before backup is made, the result would be disastrous.

Software development, programming, testing and installation

1. Top down Design



Basically, rails application uses MVC structural design. MVC means models, views and controllers. All the data are stored by the models part. Users access the pages with URL, which is translated by the route file into actions in ruby. The ruby actions will determine which html page to display and what to display in the html file. In rails application, ruby code can be combined with html code directly in an ".html.erb" file.

As a result, ruby, in the application, is responsible for all models, view, and controllers. HTML and CSS are only responsible for the view design.

2. Database Generation

To generate a database in rails, first we need to build models in the db:migrate folder.

```
class CreateDonators < ActiveRecord::Migration
 def change
  create_table :donators do |t|
    t.string :name
    t.string :user_name
    t.string :password
    t.string :email
    t.string :phone_number

    t.timestamps
  end
 end
end

class CreateSchools < ActiveRecord::Migration
 def change
  create_table :schools do |t|
    t.string :name
    t.string :user_name
    t.string :password
```

```
    t.string :email
    t.string :phone_number
    t.string :address
    t.string :card_number

    t.timestamps null: false
   end
  end
end

class CreateChildren < ActiveRecord::Migration
  def change
   create_table :children do |t|
    t.string :name
    t.integer :age
    t.string :status

    t.timestamps null: false
   end
  end
end

class CreateNews < ActiveRecord::Migration
  def change
   create_table :news do |t|
    t.string :title
    t.string :content
    t.integer :child_id
    t.timestamps
   end
  end
end

class CreateEnrollments < ActiveRecord::Migration
  def change
   create_table :enrollments do |t|
    t.integer :child_id
    t.integer :school_id

    t.timestamps null: false
   end
  end
end

class CreateDonations < ActiveRecord::Migration
  def change
   create_table :donations do |t|
    t.integer :donator_id
    t.integer :child_id
    t.string  :status
    t.timestamps null: false
   end
  end
end
```

After build the model, execute "*rake db:migrate*", and then a database is generated automatically.

3. Set routes

In a web application, routes are always a key part. Routes link a url with an action in the application. An action is a method of an application controller.

In the route file in a rails application:

```
Rails.application.routes.draw do
  #for test only
  get "/mytest" => "application#mytest"


  # Routes for the News resource:
  # CREATE
  get '/new_news' => 'news#new'
  get "/create_news/:id" => "news#create"
  get '/new_news/:id' => 'news#new'
  get '/create_news' => 'news#create'

  # READ
  get '/news' => 'news#index'
  get '/news/:id' => 'news#show'

  # DELETE
  get '/news/:id/destroy' => 'news#destroy'
  #------------------------------

  # Routes for the School resource:
  # CREATE
  get '/new_school' => 'schools#new'
  get '/create_school' => 'schools#create'

  # READ
  get '/schools' => 'schools#index'
  get '/schools/:id' => 'schools#show'

  # UPDATE
  get '/schools_edit' => 'schools#edit'
  get '/schools_update' => 'schools#update'

  # DELETE
  get '/schools/:id/destroy' => 'schools#destroy'
  #------------------------------

  get "/school_children" => "schools#show_related_children"
  get "/school_news" => "schools#show_related_news"
  get "/school_account" => "schools#account"
  get "/school_change_password" => "schools#change_password"
  get "/school_update_password" => "schools#update_password"


  # Routes for the Donator resource:
  # CREATE
  get '/new_donator' => 'donators#new'
  get '/create_donator' => 'donators#create'

  # READ
  get '/donators' => 'donators#index'
  get '/donators/:id' => 'donators#show'

  # UPDATE
```

```
  get '/donators_edit' => 'donators#edit'
  get '/donators_update' => 'donators#update'

  # DELETE
  get '/donators/:id/destroy' => 'donators#destroy'
  #------------------------------

  get "/donator_children" => 'donators#show_related_children'
  get "donator_schools" => "donators#show_related_schools"
  get "/donator_account" => "donators#account"

  get "/donator_change_password" => "donators#change_password"
  get "/donator_update_password" => "donators#update_password"



  # Routes for the Child resource:
  # CREATE
  get '/new_child' => 'children#new'
  get '/create_child' => 'children#create'

  # READ
  get '/children' => 'children#index'
  get '/children/:id' => 'children#show'

  # UPDATE
  get '/children/:id/edit' => 'children#edit'
  get '/children/:id/update' => 'children#update'

  # DELETE
  get '/children/:id/destroy' => 'children#destroy'
  get '/donateachild/:id' => 'children#donate'

  get '/change_status/:id' => 'children#change_status'


  get '/' => 'application#main'
  get '/login'=>'application#login'
  get '/signup'=>'application#signup'
  get '/sign_in' => 'application#sign_in'
  get '/sign_out' => "application#sign_out"
  get '/error_page'=> "application#error_page"
  get '/contact' => "application#contact"


end
```

The *get* '**xxx**' => "**yyy#zzz**" means when the url is xxx, execute the zzz action in the yyy controller.

Some of those routes are only available to certain users, so the access is limited in the controller.

4. Set actions

The following is the code in the application controller

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception
  def main
    #go to the main page
  end
  def login
    #go to the login page
  end

  def signup
    #go to the signup page

  end

  def sign_in
    #preset username and password for the manager
    if params["username"] == "manager" and params["password"] ==
"managerPassword97531"
      session["id"] = 0
      session["type"] = "manager"
      redirect_to "/"
    else
      #the user is not a manager
      ifDonator = FALSE
      ifSchool = FALSE
      kuser = Donator.find_by_user_name(params["username"])
      if kuser
        #the user is a donator
        thisUser = Donator.find_by_user_name(params["username"])
        ifDonator = TRUE
      else
        kuser = School.find_by_user_name(params["username"])
        if kuser
          #the user is a school
          thisUser = School.find_by_user_name(params["username"])
          ifSchool = TRUE
        else

        end
      end
      @test = thisUser

      if thisUser
        #the user exists
        if thisUser.password == params["password"]
          session["id"] = thisUser.id
          if ifDonator
            session["type"] = "donator"
          elsif ifSchool
            session["type"] = "school"
          end
          redirect_to '/'
        else
          #set error message
          session["errormessage"] = "login fail"
          redirect_to "/login"
```

```
        end
      else
        session["errormessage"] = "login fail"
        redirect_to '/login'
      end
    end


  end

  def sign_out
    #clean cookies
    session["id"] = nil
    session["type"] = nil
    session = nil
    redirect_to "/"
  end

  def error_page

  end
  def mytest

  end
  def contact

  end
end
```

The following is the code in the children controller:

```
class ChildrenController < ApplicationController

 def index
  @children = Child.all
 end

 def show
  @child = Child.find_by(id: params[:id])
  @enrollment = Enrollment.find_by_child_id(@child.id)
  @news = News.where(child_id: params[:id]).to_a
 end

 def new
 end

 def create
  if session["type"] == "school"
   @child = Child.new
   @child.name = params[:name]
   @child.age = params[:age]
   @child.status = "not donated"
   newEnrollment = Enrollment.new
   newEnrollment.school_id = session["id"]
   if @child.save
    newEnrollment.child_id = @child.id
    newEnrollment.save
    redirect_to "/children/#{ @child.id }"
   else
    render 'new'
   end
  else
   redirect_to "/error_page"
  end

 end

 def edit
  @child = Child.find_by(id: params[:id])
  @enrollment = Enrollment.find_by_child_id(params[:id])
  if session["type"] == "school" and session["id"] == @enrollment.school_id
   render "edit"
  else
   redirect_to "/error_page"
  end
 end

 def update
  @child = Child.find_by(id: params[:id])
  @enrollment = Enrollment.find_by_child_id(params[:id])
  if session["type"] == "school" and session["id"] == @enrollment.school_id
   @child.name = params[:name]
   @child.age = params[:age]

   if @child.save
    redirect_to "/children/#{ @child.id }"
   else
    render 'edit'
```

```
    end
  else
    redirect_to "/error_page"
  end

 end

 def destroy
  @child = Child.find_by(id: params[:id])
  @enrollment = Enrollment.find_by_child_id(@child.id)
  if session["type"] == "school" and session["id"] == @enrollment.school_id
    @child.destroy
    @enrollment.destroy
  else
    redirect_to "/error_page"
  end

  redirect_to "/children"
 end


 def change_status
  childID = eval(params["id"])
  a = Child.find_by(id:childID)
  @enrollment = Enrollment.find_by_child_id(childID)
  if session["type"] == "donator"
    a.status = "transfer"
    dd = Donation.new
    dd.child_id = childID
    dd.donator_id = session["id"]
    dd.status = "transfer"
    dd.save
    a.save
    redirect_to "/donator_children"    else
    redirect_to "/error_page"
  end

 end

 def donate
  childID = eval(params["id"])
  @enrollment = Enrollment.find_by_child_id(childID)
  if session["type"] == "school" and session["id"] == @enrollment.school_id
    thisD = Donation.find_by_child_id(childID)
    thisD.status = "complete"
    thisD.save
    thisC = Child.find_by_id(childID)
    thisC.status = "complete"
    thisC.save
    redirect_to "/"
  else
    redirect_to "/error_page"
  end
 end
end
```

The following is the code in the donators controller

```
class DonatorsController < ApplicationController

  def index
    #show all donators
    @donators = Donator.all
  end


  def show
    #show a specific donator
    @donator = Donator.find_by(id: params[:id])

  end

  def new
    #to the page to sign up as a donator
  end

  def create
    #the create action
    #check if the two password consistent
    if params[:password] == params[:repassword]
        if Donator.find_by_user_name(params[:username]) or
School.find_by_user_name(params[:username])
          session["errormessage"] = "the username already exists"
          redirect_to "/new_donator"
        else
          @donator = Donator.new
          #existence check and set the error message
          if params[:name] == ""
            session["errormessage"] = "please enter a name"
          end
          if params[:email] == ""
            session["errormessage"] = "please enter an email"
          end
          if params[:phone_number] == ""
            session["errormessage"] = "please enter a phone number"
          end
          if params[:username] == ""
            session["errormessage"] = "please enter a name"
          end
          if params[:password] == ""
            session["errormessage"] = "please enter a password"
          end
          @donator.name = params[:name]
          @donator.user_name = params[:username]
          @donator.password = params[:password]
          @donator.email = params[:email]
          @donator.phone_number = params[:phone_number]

          if !session["errormessage"]
            if @donator.save
              redirect_to "/"
              session["id"] = @donator.id
              session["type"] = "donator"
            else
              session["errormessage"] = "error"
              render 'new'
            end
```

```
        else
          redirect_to "/new_donator"

        end

      end

  else
    session["errormessage"] = "the passwords are inconsistent"
    redirect_to "/new_donator"
  end

end

def edit
  #to the account settings page for the user himself
  if session["id"] and session["type"] == "donator"
    @donator = Donator.find_by(id: session["id"])
    k = "nothing happens"
  else
    redirect_to "/error_page"
  end
end


def update
  #edit a donator's informaiton in the database

  if session["id"] and session["type"] == "donator"
    @donator = Donator.find_by(id: session["id"])
    #existence check and set the error message
    if params[:name] == ""
      session["errormessage"] = "please enter a name"
    end
    if params[:email] == ""
      session["errormessage"] = "please enter an email"
    end
    if params[:phone_number] == ""
      session["errormessage"] = "please enter a phone number"
    end
    if params[:password] == ""
      session["errormessage"] = "please enter a password"
    end
    @donator.name = params[:name]
    @donator.email = params[:email]
    @donator.phone_number = params[:phone_number]

    if !session["errormessage"]
      if @donator.save
        redirect_to "/donators/#{ @donator.id }"
      else
        render 'edit'
      end
    else
      redirect_to "/donators_edit"
    end

  else
    redirect_to "/error_page"
  end
```

```
end

def show_related_children
  #show the children donated by the donator
  if session["id"] and session["type"] == "donator"
    #find all donations of the donator
    donations = Donation.where(donator_id:session["id"]).to_a
    if donations
      @children = []
      if donations.class == [].class
        donations.each do |ddd|
          #find children with the donation relationship
          @children.append(Child.find_by_id(ddd.child_id))
        end
      else
        @children.append(Child.find_by_id(donations.child_id))
      end

    end
    if @children == nil
      @children = []
    end
  else
    redirect_to "/error_page"
  end

end


def account
  #show the account information
  if session["id"] and session["type"] == "donator"
    redirect_to "/donators/"+session["id"].to_s
  else
    redirect_to "/error_page"
  end
end

def inside(arrayy,item)
  #a bool function that indicate whether an item is in an array
  for i in arrayy
    if item == i
      return TRUE
      break
    end
  end
  return FALSE
end

def change_password
  # go to the change password page
  if session["id"] and session["type"] == "donator"
    @donator = Donator.find_by(id: session["id"])
    render "change_password"
  else
    redirect_to "/error_page"
  end
end

def update_password
  #reset the password in the database
```

```
    if session["id"] and session["type"] == "donator"
      @donator = Donator.find_by(id: session["id"])
      if params["old"] == @donator.password
        if params["new"] == params["renew"]
          @donator.password = params["new"]
          @donator.save
          redirect_to "/donator_account"
        else
          session["errormessage"] = "New passwords are not consistent"
          render "change_password"
        end

      else
        session["errormessage"] = "Incorrect old password"
        render "change_password"
      end
    else
      redirect_to "/error_page"
    end

  end

  def destroy
    #delete this donator
    if session["type"] == "manager"
      @donator = Donator.find_by(id: params[:id])
      @donator.destroy
      redirect_to "/donators"
    else
      redirect_to "/error_page"
    end

    #remove related donations
    found = TRUE
    while found
      thisDonation = Donation.find_by_donator_id(params[:id])
      if thisDonation
        thisDonation.destroy
      else
        found = FALSE
      end
    end
  end
end
```

The following is the code in the news controller :

```
class NewsController < ApplicationController

  def show
    #show the information of a piece of news
    @news = News.find_by(id: params[:id])
    @school_id = @news.school_id
    #determine the current user type
    if session["type"] == "manager"
    #check the suer type
    elsif session["type"] == "school"
      if session["id"] == @school_id

      else
        redirect_to "/error_page"
      end
    elsif session["type"] == "donator"
      #create new enrollment
      enrollments = Enrollment.where(school_id:@school_id).to_a
      donations = Donation.where(donator_id:session["id"] ).to_a
      childrenInSchool = []
      childrenInDonator = []
      enrollments.each do |i|
        childrenInSchool.append(Child.find_by_id(i.child_id))
      end
      #check if the news should be visible to the current user
      donations.each do |j|
        if j.status == "complete"
          childrenInDonator.append(Child.find_by_id(j.child_id))
        end
      end
      ifVisible = FALSE
      childrenInSchool.each do |k|
        childrenInDonator.each do |w|
          if k.id == w.id
            ifVisible = TRUE
            break
          end
        end
        if ifVisible
          break
        end
      end
      if not ifVisible
        #the news should not be visible to the current user
        redirect_to "/error_page"
      end

    end

  end
  def new
    #direct to the create news form
  end

  def create
    #create a news in the database
    if session["type"] == "school"
      #create a new object
      @news = News.new
```

```
    #presence check
    if params[:title] == ""
      session["errormessage"] = "please enter a title"
    end
    if params[:content] == ""
      session["errormessage"] = "please enter the content"
    end
    @news.title = params[:title]
    @news.content = params[:content]
    if params[:id]
      @news.child_id = eval(params[:id])
    else
      @news.child_id = 0
    end
    @news.school_id = session["id"]
    if !session["errormessage"]
      if @news.save
        redirect_to "/news/#{ @news.id }"
      else
        render 'new'
      end
    else
      if params[:id]
        redirect_to '/new_news/' + params[:id].to_s
      else
        redirect_to '/new_news'
      end
    end

  else
    redirect_to "error_page"
  end


end

def update
  #update the information in the database
  @news = News.find_by(id: params[:id])
  if session["type"] == "school" and session["id"] == @news.school_id
    #set the properties of the object
    @news.title = params[:title]
    @news.content = params[:content]
    @news.child_id = params[:child_id]
    @news.school_id = params[:school_id]

    #make sure the data is saved properly
    if @news.save
      redirect_to "/news/#{ @news.id }"
    else
      render 'edit'
    end
  else
    redirect_to "/error_page"
  end

end

def destroy
  #remove a news object from the database
  @news = News.find_by(id: params[:id])
```

```
    if (session["type"] == "school" and session["id"] == @news.school_id)or
session["type"] == "manager"
      if @news
        @news.destroy
      end
    else
      redirect_to "error_page"
    end
    redirect_to "/"
  end

  def index
    #show all news
    if session["type"] == "manager"
      @news = News.all
    else
      redirect_to "/error_page"
    end
  end
end
```

The following is the code in the schools controller:

```
class SchoolsController < ApplicationController

  def index
    #show all schools in the database
    @schools = School.all
  end

  def show
    #show a single school information in the database
    @school = School.find_by(id: params[:id])
  end

  def new
    #direct to the creation form of a school (sign up)
  end

  def create
    #create a school object in the database
    #check if passwords are consistent
    if params[:password] == params[:repassword]
      #check if username has been taken
      if Donator.find_by_user_name(params[:username]) or
School.find_by_user_name(params[:username])
        session["errormessage"] = "the username already exists"
        redirect_to "/new_school"
      else
        @school = School.new
        #presence check
        if params[:username] == ""
          session["errormessage"] = "please enter a username"
        end
        if params[:name] == ""
          session["errormessage"] = "please enter a name"
        end
        if params[:email] == ""
          session["errormessage"] = "please enter a email"
        end
        if params[:phone_number] == ""
          session["errormessage"] = "please enter a phone number"
        end
        if params[:address] == ""
          session["errormessage"] = "please enter a address"
        end
        if params[:card_number] == ""
          session["errormessage"] = "please enter a card number"
        end
        if params[:password] == ""
          session["errormessage"] = "please enter a card password"
        end
        #set the objects in the database
        @school.user_name=params[:username]
        @school.name = params[:name]
        @school.email = params[:email]
        @school.phone_number = params[:phone_number]
        @school.address = params[:address]
        @school.card_number = params[:card_number]
        @school.password = params[:password]
        #no error
        if !session["errormessage"]
          if @school.save
```

```
            redirect_to "/"
            session["id"] = @school.id
            session["type"] = "school"
          else
            render 'new'
          end
        else
          redirect_to "/new_school"
        end

      end
    else
      session["errormessage"] = "the passwords are inconsistent"
      redirect_to "/new_school"
    end


  end

  def edit
    #redirect to the edit page of the user account
    if session["id"] and session["type"] == "school"
      @school = School.find_by(id: session["id"])
    else
      redirect_to "/error_page"
    end
  end

  def update
    #update the user information
    if session["id"] and session["type"] == "school"
      @school = School.find_by(id: session["id"])
      #presence check
      if params[:name] == ""
        session["errormessage"] = "please enter a name"
      end
      if params[:email] == ""
        session["errormessage"] = "please enter a email"
      end
      if params[:phone_number] == ""
        session["errormessage"] = "please enter a phone number"
      end
      if params[:address] == ""
        session["errormessage"] = "please enter a address"
      end
      if params[:card_number] == ""
        session["errormessage"] = "please enter a card number"
      end
      @school.name = params[:name]
      @school.email = params[:email]
      @school.phone_number = params[:phone_number]
      @school.address = params[:address]
      @school.card_number = params[:card_number]


      #no error
      if !session["errormessage"]
        if @school.save
          redirect_to "/schools/#{ @school.id }"
        else
          render 'edit'
```

```
      end
    else
      redirect_to "schools_edit"
    end

  else
    redirect_to "/error_page"
  end

end


def destroy
  #remove a school from the database
  #only available to the manager
  if session["type"] == "manager"
    @school = School.find_by(id: params[:id])
    @school.destroy
    redirect_to "/schools"
  else
    redirect_to "/error_page"
  end

  found = TRUE
  while found
    thisNews = News.find_by_school_id(params[:id])
    if thisNews
      thisNews.destroy
    else
      found = FALSE
    end
  end

  found = TRUE
  while found
    thisEnrollment = Enrollment.find_by_school_id(params[:id])
    if thisEnrollment
      thisEnrollment.destroy
    else
      found = FALSE
    end
  end
end



def show_related_children
  #show the children in the schools
  if session["id"] and session["type"] == "school"
    #find related enrollments
    enrollments = Enrollment.where(school_id:session["id"]).to_a
    if enrollments
      @children = []
      enrollments.each do |eee|
        this = Child.find_by_id(eee.child_id)
        if this
          @children.append(this)
        end
      end

    else
```

```
        @children = []

      end

      if not @children
        @children = []
      end
    else
      redirect_to "/error_page"
    end

  end
  def show_related_news
    #show the news of the school/news related to the children in the school
    if session["id"] and session["type"] == "school"
      @news = News.where(:school_id => session["id"]).to_a

      if @news.class != [].class
        @news = [@news]
      end
    else
      redirect_to "/error_page"
    end
  end

  def account
    #show account information
    if session["id"]
      redirect_to "/schools/"+session["id"].to_s
    else
      redirect_to "/error_page"
    end
  end

  def change_password
    #direct to the change_password form of the account
    if session["id"] and session["type"] == "school"
      @school = School.find_by(id: session["id"])
    else
      redirect_to "/error_page"

    end
  end

  def update_password
    #edit the password in the database
    if session["id"] and session["type"] == "school"
      @school = School.find_by(id: session["id"])
      #check if old password correct
      if params["old"] == @school.password
        #check if two new passwords consistent
        if params["new"] == params["renew"]
          @school.password = params["new"]
          @school.save
          redirect_to "/school_account"
        else
          session["errormessage"] = "New passwords are not consistent"
          render "change_password"
        end

      else
```

```
        session["errormessage"] = "Incorrect old password"
        render "change_password"
      end
    else
      redirect_to "/error_page"
    end

  end
end
```

Some further interpretation:

a. “params” is the variables passed between html codes by the URL.

b. to implement log in, we use the default variable “session”. Session is a kind of cookie that is stored with the browser, and deleted when the browser is quitted. If the

c. the variable which begins with an “@” means the variable is global. The ”@” notation makes the variable still accessible in the html files.

5. HTML UI Implementation

For the HTML part, instead of using files with .html suffix, we use .html.erb, which means ruby code can be included in the file with the <% %> symbol. For example

```
<input id="email" name="email" value="<%= @school.email  %>" type="text" />
```

Now an HTML file can read values of variables.

The content of layout file is applied to every single page in the system.

In our layout file, we do two things:

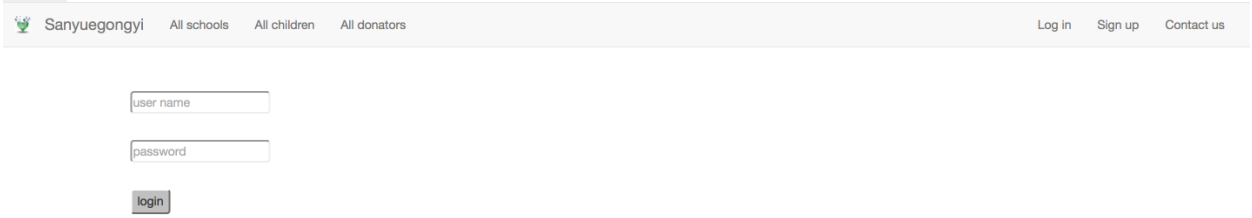
- set up bootstrap, a package of web UI design for the project.
- set the navigation bar, which appears on every page in the project.

```
<!DOCTYPE html>
<html>
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <!-- Brand and toggle get grouped for better mobile display -->
    <div class="navbar-header">
      <img class="navbar-brand"  style = "border-radius: 20px;" src="icon.jpg">
      <button type="button" class="navbar-toggle collapsed " data-toggle="collapse" data-target="#bs-example-navbar-
collapse-1" aria-expanded="false">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a class="navbar-brand" href="/">Sanyuegongyi</a>
    </div>

    <!-- Collect the nav links, forms, and other content for toggling -->
    <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
      <ul class="nav navbar-nav">
        <%if session["type"] == "donator"%>
          <li><a href="/donator_children">My donated children</a></li>
        <%elsif session["type"] == "school"%>
          <li><a href="/school_news">News</a></li>
          <li><a href="/school_children">Our children</a></li>
        <%end%>
        <li><a href="/schools">All schools</a></li>
        <li><a href="/children">All children</a></li>
        <li><a href="/donators">All donators</a></li>


      </ul>


      <ul class="nav navbar-nav navbar-right">
        <%if session["type"] == "donator"%>
          <li><a href="/donator_account">Welcome, <%= Donator.find_by_id(session["id"]).name%>! Your account</a></li>
          <li><a href="/sign_out">Sign out </a></li>
        <%elsif session["type"] == "school"%>
```

```
        <li><a href="/school_account">Welcome, <%= School.find_by_id(session["id"]).name%>! Your account</a></li>
        <li><a href="/sign_out">Sign out </a></li>
      <%elsif session["type"] == "manager"%>
        <li><a href="/">Welcome, manager! Your account</a></li>
        <li><a href="/sign_out">Sign out </a></li>
      <%else%>
        <li><a href="/login">Log in</a></li>
        <li><a href="/signup">Sign up</a></li>
      <%end%>
      <li><a href="/contact">Contact us</a></li>


    </ul>
   </div><!-- /.navbar-collapse -->
  </div><!-- /.container-fluid -->
</nav>
<head>
  <title>Sanyuegongyi</title>
  <%= stylesheet_link_tag    'application', media: 'all', 'data-turbolinks-track' => true %>
  <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
  <%= csrf_meta_tags %>
  <script src="https://code.jquery.com/jquery.min.js"></script>
  <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css" rel="stylesheet" type="text/css" />
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
  <meta charset="utf-8">

  <style>
    .form-group{
      margin-top: 30px;
      margin-bottom: 30px

    }
    input {
      border-radius: 4px;
    }
  </style>

</head>
<body>

<%= yield %>

</body>
</html>
```

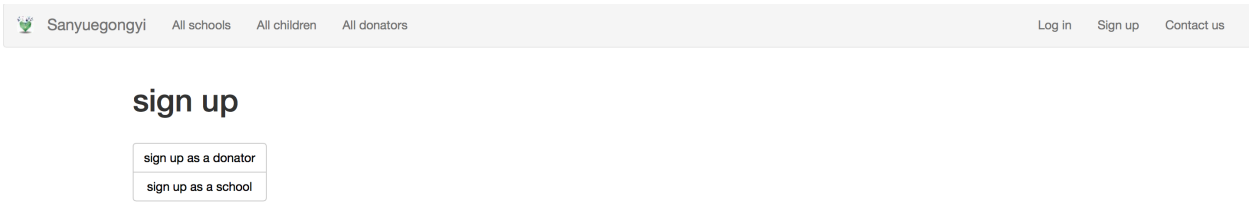
Login Page

```
<div class = "container">
<form action="/sign_in" role="form" class="form-horizontal">
 <div class = "form-group">
  <div class="col-sm-12">
   <input type = "text" placeholder="user name" name = "username">
   </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
  <input type = "password" placeholder="password" name = "password">
  </div>
  </div>
  <div class = "form-group">
   <div class="col-sm-12">
   <input type = "submit" value = "login">
    </div>

  </div>
</form>
</div>
```



Sign up page

```
<head>
 <style>
    .row{
       margin-top: 30px;
       margin-bottom: 30px;
    }
 </style>
</head>


<div class="container">
<h1 style="margin-bottom: 30px">sign up</h1>

<div class="btn-group-vertical" role="group" aria-label="...">
 <button type="button" class="btn btn-default" ><a href="/new_donator" style="color:black">sign up as a
donator</a></button>

 <button type="button" class="btn btn-default"><a href="/new_school" style="color:black">sign up as a
school</a></button>
</div>
</div>
```



Create information

Children

```
<div class = "container">
 <h1>New Child</h1>
 <b style="color:red"><%=session["errormessage"]%></b>
 <%session["errormessage"] = nil%>
 <form action="/create_child" role="form" class="form-horizontal">

  <div class = "form-group">
   <div class="col-sm-12">
    <label for="name">name</label><br />
    <input id="name" name="name" type="text" />
   </div>
  </div>

  <div class = "form-group">
   <div class="col-sm-12">
    <label for="age">age</label><br />
    <input id="age" name="age" type="text" />
   </div>
  </div>

  <div class = "form-group">
   <div class="col-sm-12">
    <input type="submit">
   </div>
  </div>
 </form>
</div>
```

Schools(Sign up)

```
<div class = "container">
 <h1>New School</h1>
 <b style="color:red"><%=session["errormessage"]%></b>
 <%session["errormessage"] = nil%>
 <form action="/create_school" role="form" class="form-horizontal">
  <div class = "form-group">
   <div class="col-sm-12">
    <label for="username">user_name</label><br />
    <input id="username" name="username" type="text" />    </div>
  </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="name">name</label><br />
   <input id="name" name="name" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="password">password</label><br/>
   <input id="password" name="password" type="password" />
  </div>
 </div>
```

```
 <div class = "form-group">
  <div class="col-sm-12">
   <label for="repassword">repassword</label><br />
   <input id="repassword" name="repassword" type="password" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="email">email</label><br />
   <input id="email" name="email" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="phone_number">phone_number</label><br />
   <input id="phone_number" name="phone_number" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="address">address</label><br />
   <input id="address" name="address" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">

   <label for="card_number">card_number</label><br />
   <input id="card_number" name="card_number" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <input type="submit">
  </div>
 </div>
 </form>
</div>
```

News

```
<%if params[:id]%>
  <div class = "container">
   <h1>New news</h1>
   <b style="color:red"><%=session["errormessage"]%></b>
   <%session["errormessage"] = nil%>
   <form action="/create_news/<%=params[:id]%>" role="form" class="form-horizontal">
    <div class = "form-group">
     <div class="col-sm-12">
      <label for="title">title</label><br />
      <input id="title" name="title" type="text" />
     </div>
    </div>
```

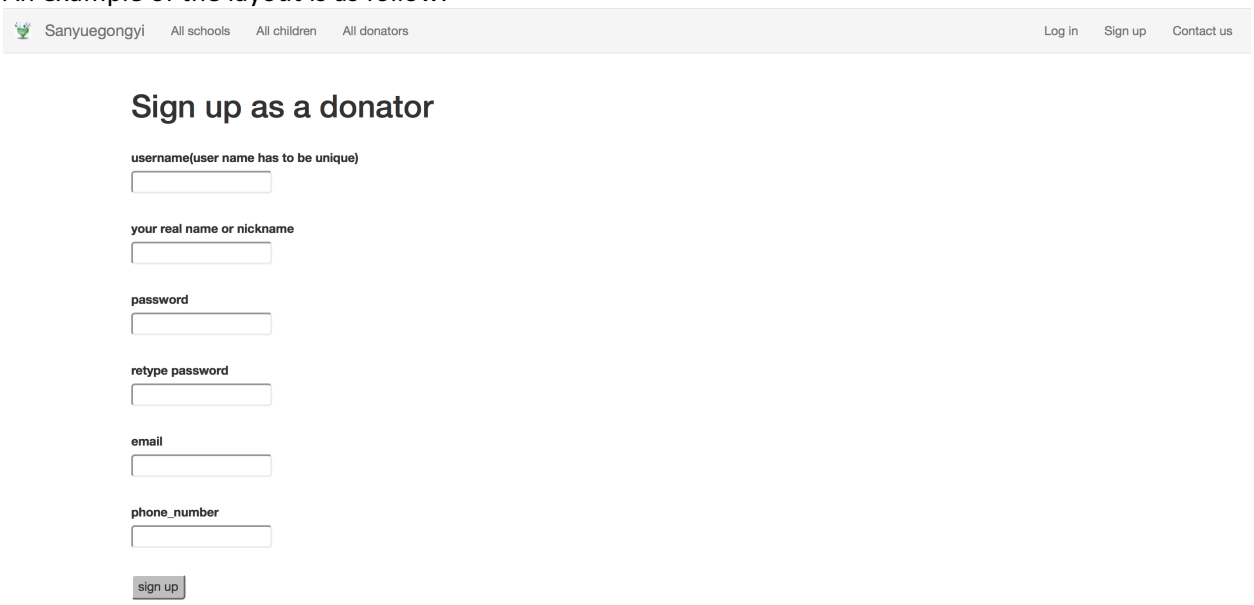
```
      <div class = "form-group">
       <div class="col-sm-12">
        <label for="content">content</label><br />
        <textarea  name = "content" placeholder="content" style = "width:100%; height: 300px;border-radius:
3px;"></textarea>
       </div>
      </div>

      <div class = "form-group">
       <div class="col-sm-12">
        <input type="submit">
       </div>
      </div>
    </form>
   </div>
<%else%>
   <div class = "container">
    <h1>New news</h1>
    <b style="color:red"><%=session["errormessage"]%></b>
    <%session["errormessage"] = nil%>
    <form action="/create_news" role="form" class="form-horizontal">
     <div class = "form-group">
      <div class="col-sm-12">
       <label for="title">title</label><br />
       <input id="title" name="title" type="text" />
      </div>
     </div>

      <div class = "form-group">
       <div class="col-sm-12">
        <label for="content">content</label><br />
        <textarea  name = "content" placeholder="content" style = "width:100%; height: 300px;border-radius:
3px;"></textarea>
       </div>
      </div>

      <div class = "form-group">
       <div class="col-sm-12">
        <input type="submit">
       </div>
      </div>
    </form>
   </div>
<%end%>
```

Donators(Sign up)

```
<div class = "container">
 <h1>Sign up as a donator</h1>
 <b style="color:red"><%=session["errormessage"]%></b>
  <%session["errormessage"] = nil%>
 <form action="/create_donator" role="form" class="form-horizontal">
 <div class = "form-group">
  <div class="col-sm-12">
   <label for="username">username(user name has to be unique)</label><br />
   <input id="username" name="username" type="text" />
  </div>
```

```
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="name">your real name or nickname</label><br />
   <input id="name" name="name" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="password">password</label><br />
   <input id="password" name="password" type="password" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <label for="repassword">retype password</label><br />
   <input id="repassword" name="repassword" type="password" />   </div>
 </div>


 <div class = "form-group">
  <div class="col-sm-12">
   <label for="email">email</label><br />
   <input id="email" name="email" type="text" />   </div>
 </div>


 <div class = "form-group">
  <div class="col-sm-12">
   <label for="phone_number">phone_number</label><br />
   <input id="phone_number" name="phone_number" type="text" />
  </div>
 </div>

 <div class = "form-group">
  <div class="col-sm-12">
   <input type="submit" value = "sign up">
  </div>
 </div>

</form>

</div>
```

An example of the layout is as follow.



Edit information

Children

```
<div class = "container">
 <h1>Editing Child #<%= @child.name %></h1>
 <b style="color:red"><%=session["errormessage"]%></b>
 <%session["errormessage"] = nil%>
 <form action="/children/<%= @child.id %>/update" role="form" class="form-horizontal">

  <div class = "form-group">
   <div class="col-sm-12">
    <label for="name">name</label><br />
    <input id="name" name="name" value="<%= @child.name  %>" type="text" />
   </div>
  </div>

  <div class = "form-group">
   <div class="col-sm-12">
    <label for="age">age</label><br />
    <input id="age" name="age" value="<%= @child.age  %>" type="text" />
   </div>
  </div>

  <div class = "form-group">
   <div class="col-sm-12">
    <input type="submit">
   </div>
  </div>
 </form>
</div>
```

School

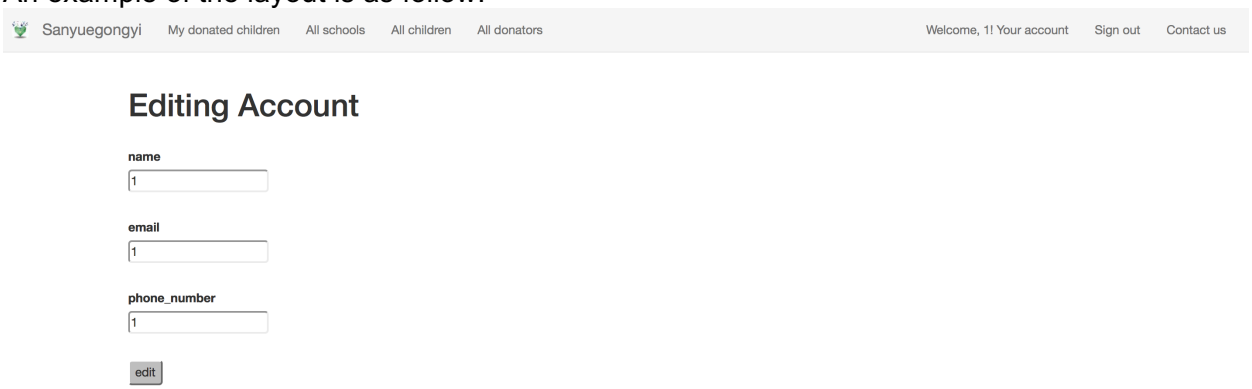
```
<div class = "container">
 <h1>Editing School #<%= @school.name %></h1>
 <b style="color:red"><%=session["errormessage"]%></b>
 <%session["errormessage"] = nil%>
 <form action="/schools_update" role="form" class="form-horizontal">
  <div class = "form-group">
   <div class="col-sm-12">
    <label for="name">name</label><br />
    <input id="name" name="name" value="<%= @school.name  %>" type="text" />
   </div>
  </div>

  <div class = "form-group">
   <div class="col-sm-12">
    <label for="email">email</label><br />
    <input id="email" name="email" value="<%= @school.email  %>" type="text" />
   </div>
  </div>

  <div class = "form-group">
   <div class="col-sm-12">
    <label for="phone_number">phone_number</label><br />
    <input id="phone_number" name="phone_number" value="<%= @school.phone_number  %>" type="text" />
   </div>
  </div>

  <div class = "form-group">
```

```
    <div class="col-sm-12">
      <label for="address">address</label><br />
      <input id="address" name="address" value="<%= @school.address  %>" type="text" />
    </div>
   </div>

   <div class = "form-group">
    <div class="col-sm-12">
      <label for="card_number">card_number</label><br />
      <input id="card_number" name="card_number" value="<%= @school.card_number  %>" type="text" />
    </div>
   </div>

   <div class = "form-group">
    <div class="col-sm-12">
      <input type="submit">
    </div>
   </div>
  </form>
</div>
```

Donators

```
<div class = "container">
  <h1>Editing Account</h1>
  <b style="color:red"><%=session["errormessage"]%></b>
  <%session["errormessage"] = nil%>
  <form action="/donators_update" role="form" class="form-horizontal">
   <div class = "form-group">
    <div class="col-sm-12">
      <label for="name">name</label><br />
      <input id="name" name="name" value="<%= @donator.name  %>" type="text" />
    </div>
   </div>

   <div class = "form-group">
    <div class="col-sm-12">
      <label for="email">email</label><br />
      <input id="email" name="email" value="<%= @donator.email  %>" type="text" />
    </div>
   </div>

   <div class = "form-group">
    <div class="col-sm-12">
      <label for="phone_number">phone_number</label><br />
      <input id="phone_number" name="phone_number" value="<%= @donator.phone_number  %>" type="text" />
    </div>
   </div>
   <div class = "form-group">
    <div class="col-sm-12">
      <input type="submit" value = "edit">
    </div>
   </div>
  </form>
</div>
```

An example of the layout is as follow.



Show information of individual Children

```
<div class="container">
 <h1>Child <%= @child.name %></h1>
 <div class="list-group">
  <p class="list-group-item active">
   <h4 class="list-group-item-heading">Name</h4>
   <p class="list-group-item-text"><%= @child.name %></p>
  </p>

  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Age</h4>
  <p class="list-group-item-text"><%= @child.age %></p>
  </p>

  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Status</h4>
  <p class="list-group-item-text"><%= @child.status %></p>
  </p>


 </div>

<%if session["type"] == "donator"%>
<div>
 <%if @child.status == "not donated"%>
   <a class="btn btn-default" role="button" href= <%="/change_status/"+@child.id.to_s %> >
    Donate
   </a>
 <%end%>

</div>
<%end%>

<%if session["type"] == "school" and session["id"] == @enrollment.school_id%>
<a class="btn btn-default" role="button" href="/children/<%= @child.id %>/edit">Edit</a>
<a class="btn btn-default" role="button" href="/children/<%= @child.id %>/destroy">Destroy</a>
  <a class="btn btn-default" role="button" href="/new_news/<%= @child.id %>">Add news</a>
  <%if @child.status == "transfer"%>
    <a class="btn btn-default" role="button" href="/donateachild/<%= @child.id %>">Accept donation</a>

  <%end%>
<%end%>

<%ifVisible = FALSE%>
<%if session["type"] == "manager" %>
  <% ifVisible = TRUE %>
<% elsif session["type"] == "school"%>
  <% enrollment = Enrollment.find_by_child_id(@child.id)%>
  <% if enrollment.school_id == session["id"]%>
    <% ifVisible = TRUE%>
  <%end%>
<%elsif session["type"] == "donator"%>
  <% donation = Donation.where(id:@child.id).to_a%>
  <% donation.each do |bla|%>
    <%if bla.donator_id == session["id"]%>
      <%ifVisible = TRUE%>
```

```
      <%break%>
    <%end %>
  <%end %>
<%end%>

</div>
<%if ifVisible%>
  <div class="container">
   <h1>News</h1>
   <div class="row">
    <div class="col-xs-6 col-md-3">

     <%if session["type"] == "school" or session["type"] == "manager" %>
       <p><a href="/new_news/<%=@child.id%>" class="btn btn-default" role="button">Add a New news</a></p>
     <%end%>
    </div>
   </div>
   <% @news.each do |news| %>
     <div class="col-xs-6 col-md-3">
      <div class="thumbnail">
       <div class="caption">
        <h3><%= news.title %></h3>
        content:<%= news.content %><br/>
        <%if Child.find_by(news.child_id)%>
          child:<%= Child.find_by(news.child_id).name %><br/>
        <%end%>
        school:<%= School.find_by(news.school_id).name  %></br>
        <a href="/news/<%= news.id %>" class="btn btn-primary" role="button">Show</a>
        <%if session["type"] == "manager" %>
          <a href="/news/<%= news.id %>/destroy" class="btn btn-default" role="button">Destroy</a>
        <%end%>
       </div>
      </div>
     </div>
  <%end%>
 </div>
<%end%>
```

Schools

```
<div class="container">
 <h1>School #<%= @school.name %></h1>
 <div class="list-group">
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Name</h4>
  <p class="list-group-item-text"><%= @school.name %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Email</h4>
  <p class="list-group-item-text"><%= @school.email %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Phone number</h4>
  <p class="list-group-item-text"><%= @school.phone_number %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Address</h4>
  <p class="list-group-item-text"><%= @school.address %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Card Number</h4>
  <p class="list-group-item-text"><%= @school.card_number %></p>
  </p>
  <%if @school.id == session["id"] and session["type"] == "school"%>
     <a class="btn btn-default" role="button" href="/schools_edit">edit account</a>
     <a class="btn btn-default" role="button" href="/school_change_password">change password</a>
  <%end%>

 </div>
</div>
```

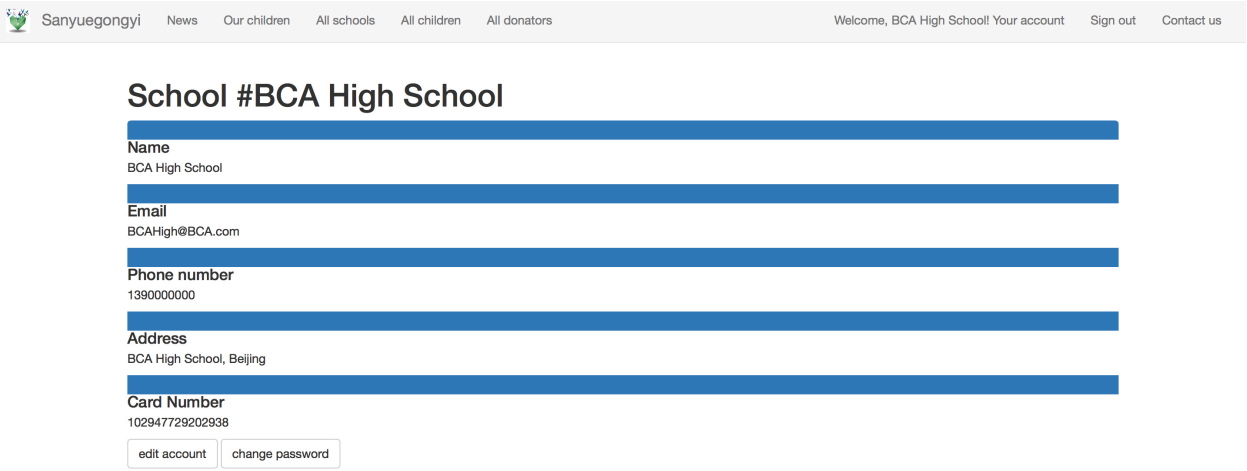
News

```
<div class="container">
 <h1>News</h1>
 <div class="list-group">
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Title</h4>
  <p class="list-group-item-text"><%= @news.title %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Content</h4>
  <p class="list-group-item-text"><%= @news.content %></p>
  </p>

  <%if Child.find_by(@news.child_id)%>
     <p class="list-group-item active">
        <h4 class="list-group-item-heading">Child</h4>
     <p class="list-group-item-text"><%=  Child.find_by(@news.child_id).name %></p>
     </p>
  <%end%>
  <div>
   <p class="list-group-item active">
   <h4 class="list-group-item-heading">School</h4>
   <p class="list-group-item-text"><%= School.find_by(@news.school_id).name %></p>
   </p>
  </div>
```

```
  <%if session["id"] == @news.school_id and session["type"] == "school"%>
    <a class="btn btn-default" role="button" href="/news/<%= @news.id %>/destroy">Destroy</a>
  <%end%>
 </div>
</div>
```

Donators

```
<div class="container">
 <h1>Donator <%= @donator.name %></h1>
 <div class="list-group">
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Name</h4>
  <p class="list-group-item-text"><%= @donator.name %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Email</h4>
  <p class="list-group-item-text"><%= @donator.email %></p>
  </p>
  <p class="list-group-item active">
  <h4 class="list-group-item-heading">Phone number</h4>
  <p class="list-group-item-text"><%= @donator.phone_number %></p>
  </p>
  <%if @donator.id == session["id"] and session["type"] == "donator"%>
    <a class="btn btn-default" role="button" href="/donators_edit">edit account</a>
    <a class="btn btn-default" role="button" href="/donator_change_password">change password</a>
  <%end%>

 </div>
</div>
```



Show information of all Children

```
<div class="container">

<h1>Children</h1>
 <div class="row">
  <div class="col-xs-6 col-md-3">

  <%if session["type"] == "school" or session["type"] == "manager" %>
    <p><a href="/new_child" class="btn btn-default" role="button">Add a New Child</a></p>
  <%end%>
 </div>
  </div>
  <% @children.each do |child| %>
    <div class="col-xs-6 col-md-3">
     <div class="thumbnail">
      <div class="caption">
       <h3> <td><%= child.name %></td></h3>
       age: <td><%= child.age %></td><br>
       status  <td><%= child.status %></td>
       <p><a href="/children/<%= child.id %>" class="btn btn-primary" role="button">Show</a>
        <%if session["type"] == "manager" %>
         <div><a href="/children/<%= child.id %>/edit" class="btn btn-default" role="button">Edit</a></div><br>
         <a href="/children/<%= child.id %>/destroy" class="btn btn-default" role="button">Destroy</a>
       <%end%>
        </p>
      </div>
     </div>
    </div>
 <%end%>
</div>
```

Schools

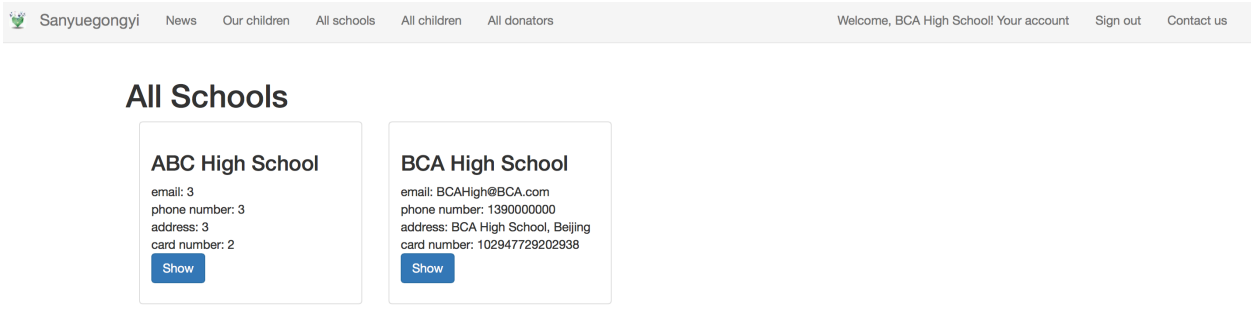
```
<div class="container">

 <h1>All Schools</h1>
 <% @schools.each do |school| %>
   <div class="col-xs-6 col-md-3">
    <div class="thumbnail">
     <div class="caption">
      <h3><%= school.name %></h3>
       email: <%= school.email %><br/>
       phone number: <%= school.phone_number %><br/>
       address: <%= school.address %><br/>
       card number: <%= school.card_number %><br/>
      <p><a href="/schools/<%= school.id %>" class="btn btn-primary" role="button">Show</a>
       <%if session["type"] == "manager" %>
        <a href=href="/schools/<%= school.id %>/destroy"> class="btn btn-default" role="button">Destroy</a>
      <%end%>
       </p>
     </div>
    </div>
   </div>
 <%end%>
</div>
```

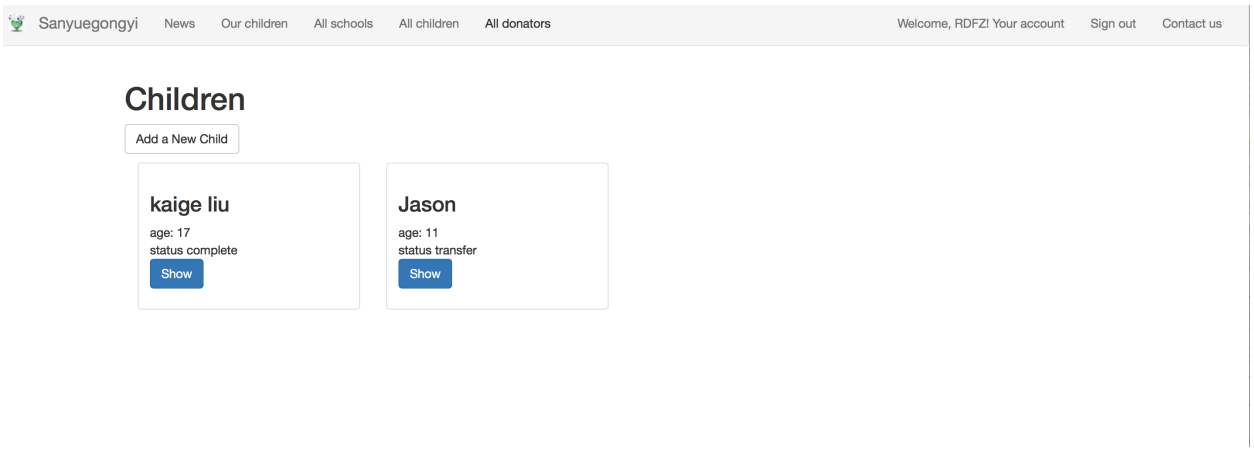
News

```
<div class="container">
 <h1>News</h1>
 <div class="row">
  <div class="col-xs-6 col-md-3">

   <%if session["type"] == "school" or session["type"] == "manager" %>
     <p><a href="/new_news" class="btn btn-default" role="button">Add a New news</a></p>
   <%end%>
  </div>
 </div>
 <% @news.each do |news| %>
   <div class="col-xs-6 col-md-3">
    <div class="thumbnail">
     <div class="caption">

      <h3><%= news.title %></h3>
      content:<%= news.content %><br/>
      <%if Child.find_by(news.child_id)%>
        child:<%= Child.find_by(news.child_id).name %><br/>
      <%end%>
      school:<%= School.find_by(news.school_id).name  %></br>
      <a href="/news/<%= news.id %>" class="btn btn-primary" role="button">Show</a>

       <%if session["type"] == "manager" %>
        <a href="/news/<%= news.id %>/destroy" class="btn btn-default" role="button">Destroy</a>
       <%end%>
     </div>
    </div>
   </div>
 <%end%>
</div>
```

Donators

```
<div class="container">

 <h1>All Donators</h1>
 <% @donators.each do |donator| %>
   <div class="col-xs-6 col-md-3">
    <div class="thumbnail">
      <h3><%= donator.name %></h3>
      email: <%= donator.email %><br/>
      phone number: <%= donator.phone_number %><br/>
     <a href="/donators/<%= donator.id %>" class="btn btn-primary" role="button">Show</a>
       <%if session["type"] == "manager" %>
       <a href="/donators/<%= donator.id %>/destroy" class="btn btn-default" role="button">Destroy</a>
       <%end%>

    </div>
   </div>
 <%end%>
</div>
```


Sanyuegongyi
News
Our children
All schools
All children
All donators
Welcome, BCA High School! Your account
Sign out
Contact us
All Schools
ABC High School
email: 3
phone number: 3
address: 3
card number: 2
Show
BCA High School
email: BCAHigh@BCA.com
phone number: 1390000000
address: BCA High School, Beijing
card number: 102947729202938
Show


Sanyuegongyi
News
Our children
All schools
All children
All donators
Welcome, RDFZ! Your account
Sign out
Contact us
Children
Add a New Child
kaige liu
age: 17
status complete
Show
Jason
age: 11
status transfer
Show

6. Testing

Test Plan:

Test type	Test No.	Test objective	Test method	Test data	Expected outcome	Test outcome
Login process	1	Test whether the login is allowed when input information is empty.	Leave the user name or password field empty.	N/A	The system will warn the user the login cannot success.	Unsuccessful but resolved
	2	Test whether the login is allowed when the password is wrong.	Input the wrong password and login.	123	The system will warn the user the login cannot success.	Success
Sign up process	3	Test whether the sign up process allow the two passwords inconsistent.	Input different values in two password fields	123 and 321	The system will warn the user the sign up cannot success with inconsistent passwords.	Success
	4	Test whether the sign up process allows any necessary field to be blank.	Leave the email field empty	N/A	The system will warn the user the sign up cannot success without the email address.	Unsuccessful but resolved
Sign out process	5	Test whether the sign out process cleans the name on the navigation bar.	Login, and then click sign out. Check the navigation bar.	N/A	The system will remove the name from the navigation bar.	Unsuccessful but resolved.
Create	6	Test whether the data is created	Login as a school,	N/A	The system will create a new child	Success

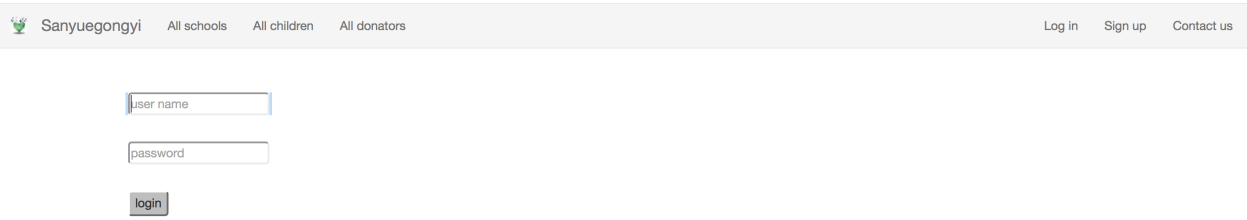
		and saved when a new child is created by a school in the system.	create a child and save.		in the database.	
	7	Test whether the data is created and saved when a new piece of news is created by a school in the system.	Login as a school, create a news and save.	N/A	The system will create a new news in the database.	Success
Edit	8	Test whether the data is edited and saved when a new child is created by a school in the system.	Login as a school, edit a child and save.	N/A	The system will edit the child in the database.	Success
Delete	9	Test whether the data is created and saved when a new child is created by a school in the system.	Login as a school and delete a child.	N/A	The system the remove the child from the database.	Success
Donation	10	Test whether the status of a child is changed to “on transfer” and the “donate” button disappears when a donation is	Login as a donator and donate a child	N/A	The status of a child is changed to “transfer” and the “donate” button disappears.	Success

		clicked by a donator.				
Access to news	11	Test whether the news of a child is available only to his or her donator.	Login as a new donator and see the news of a child.	N/A	The news of the child should not be available.	Unsuccessful but resolved

Test 1

First I sign up as a donator, with user name “abc” and password “111”.

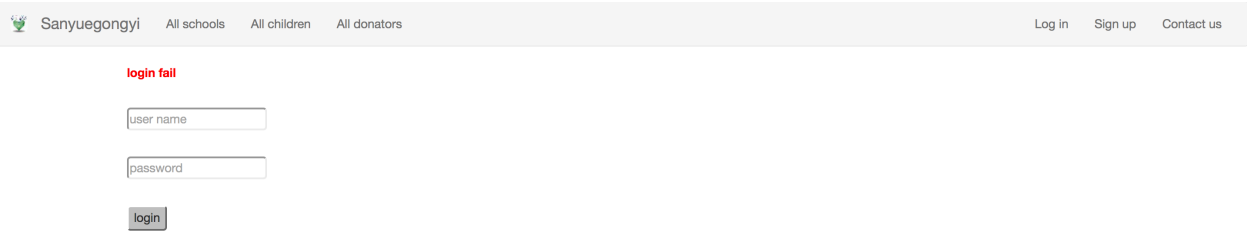
After clicking “login” button with the fields empty, nothing shows up. The expected result should be an error message appears.



The reason for this error is I forgot to add the error message to the view.

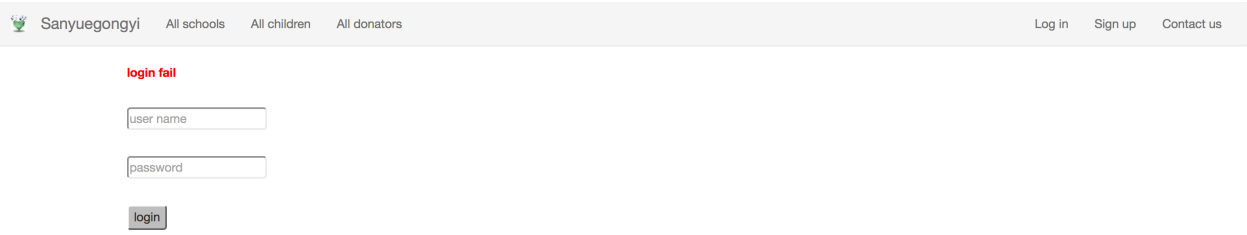
```
<%=session["errormessage"]%></b>
<%session["errormessage"] = nil%>
```

The problem is resolved by adding the two lines of code.



Test 2

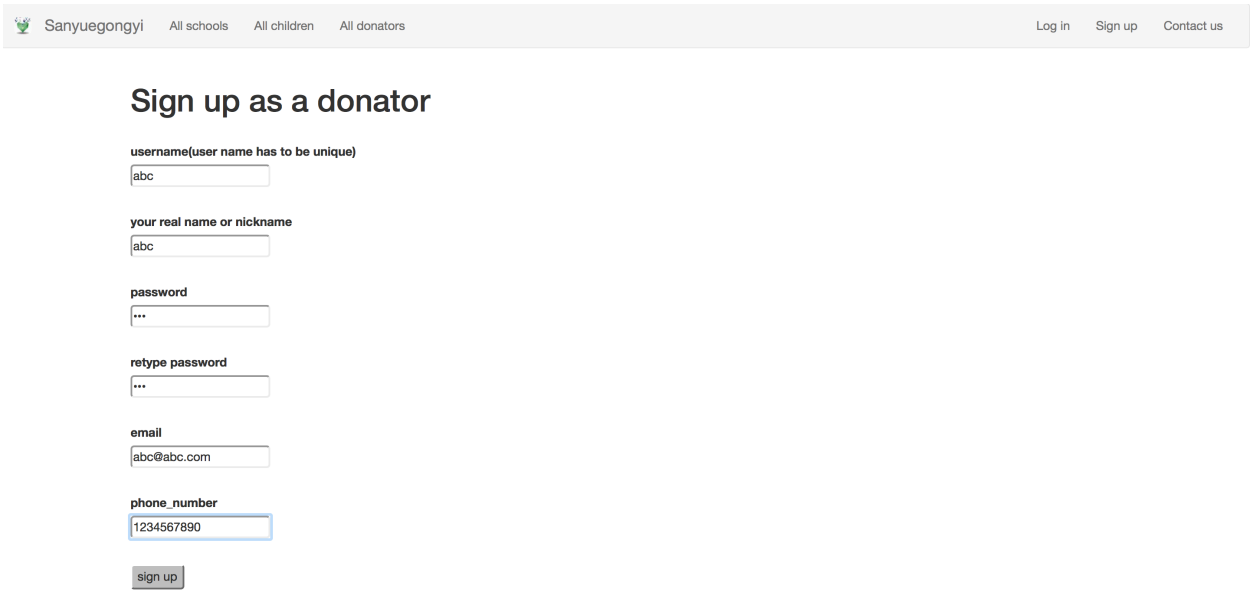
First I sign up as a donator, with user name “abc” and password “111”. Then, I login with user name “abc” and password “123”.

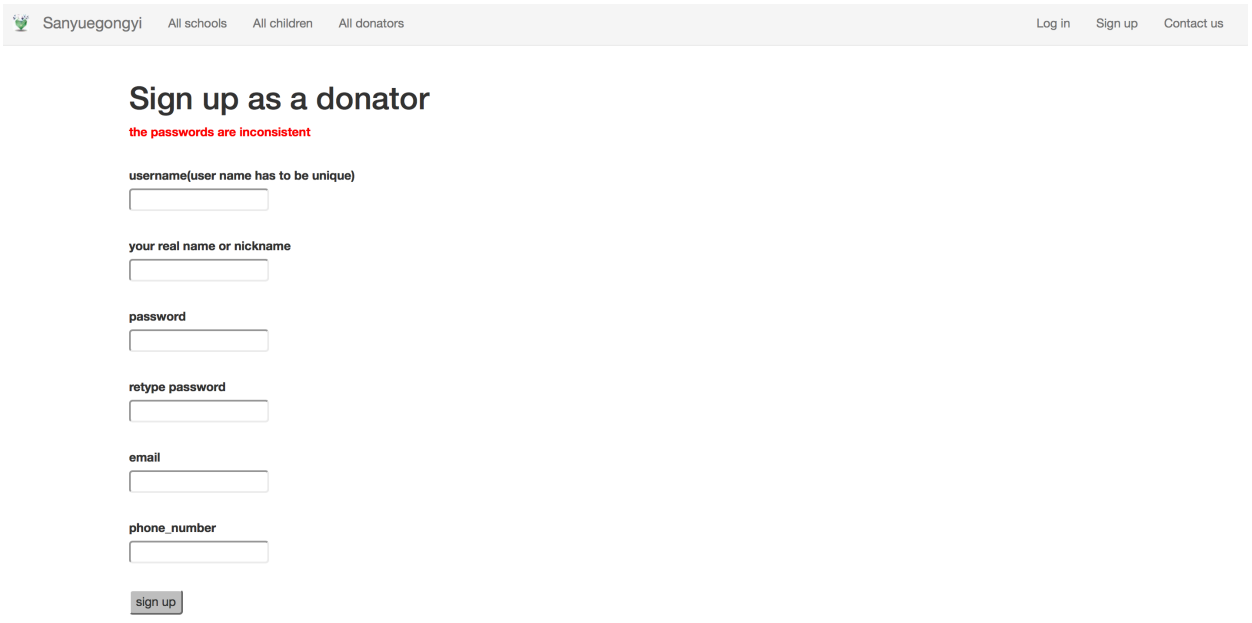


As expected, a warning appears, saying “login fail”.

Test 3

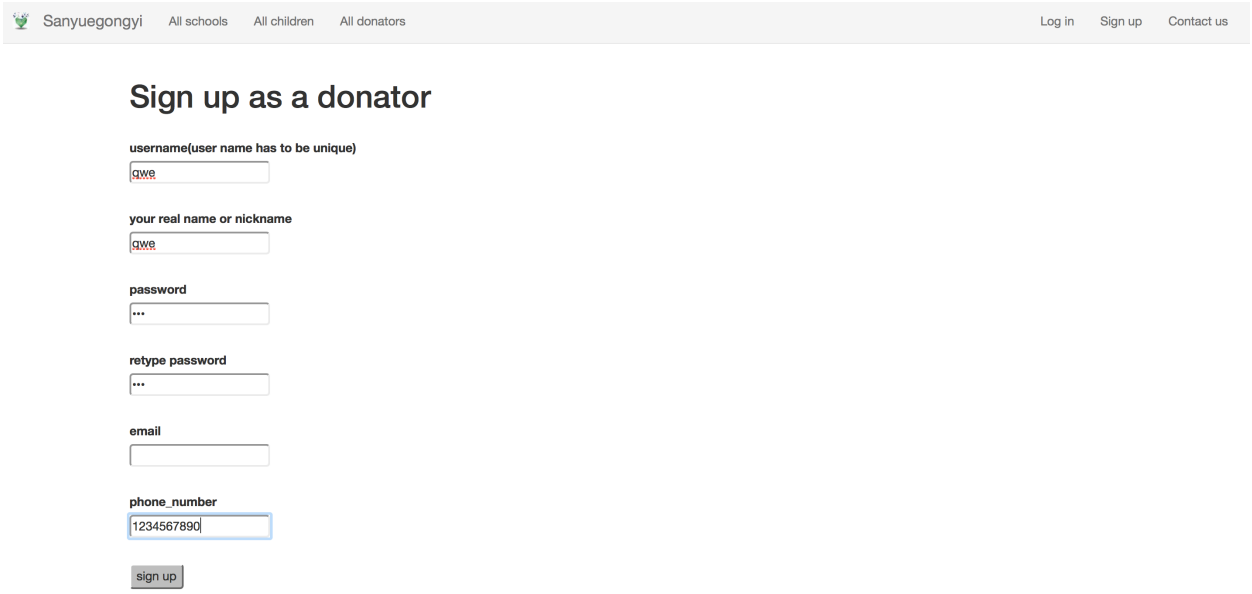
By entering “123” and “321” into the fields, an error message is produced .





Test 4

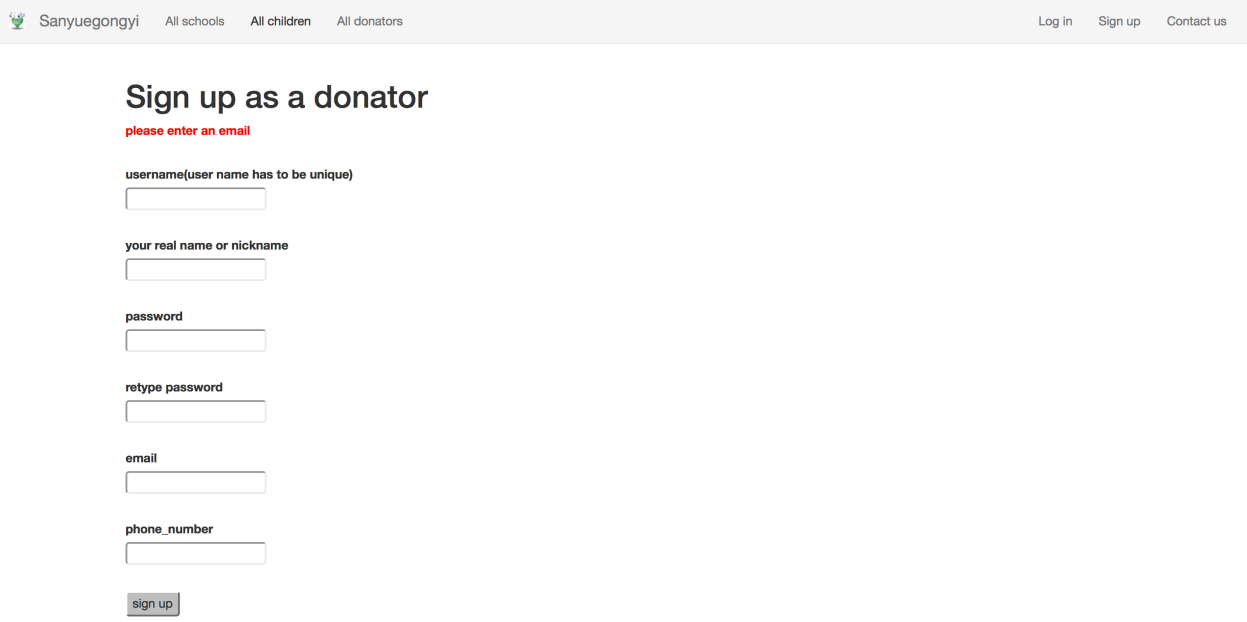
By leaving the email field blank and clicking “sign up”, no error message is produced. After checking the code, I forgot to add the error check to the “new” method in “donator” class.




```
if params[:name] == ""
  session["errormessage"] = "please enter a name"
end
if params[:email] == ""
  session["errormessage"] = "please enter an email"
end
if params[:phone_number] == ""
  session["errormessage"] = "please enter a phone number"
end
if params[:username] == ""
  session["errormessage"] = "please enter a name"
end
if params[:password] == ""
  session["errormessage"] = "please enter a password"
end
```

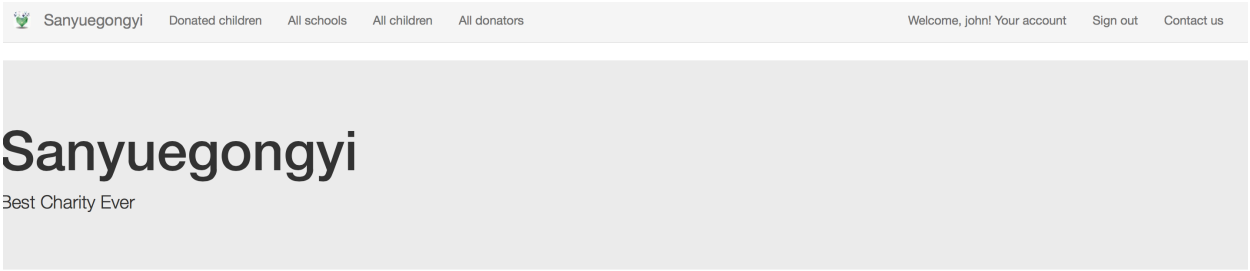
By leaving the email field blank and clicking “sign up”, no error message is produced. After checking the code, I forgot to add the error check to the “new” method in “donator” class.

After the error check is added, the correct error response is produced.



Test 5

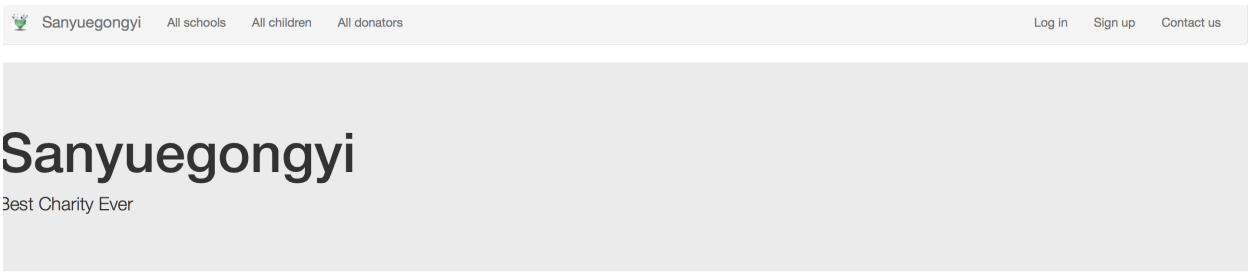
After I sign in as a donator, I click the “sign up” button. The navigation bar doesn’t change as expected.



After checking the code, I find I didn’t clean the “session”, which stores and encryptes the user name currently in use, when the user clicks the “sign out” button.

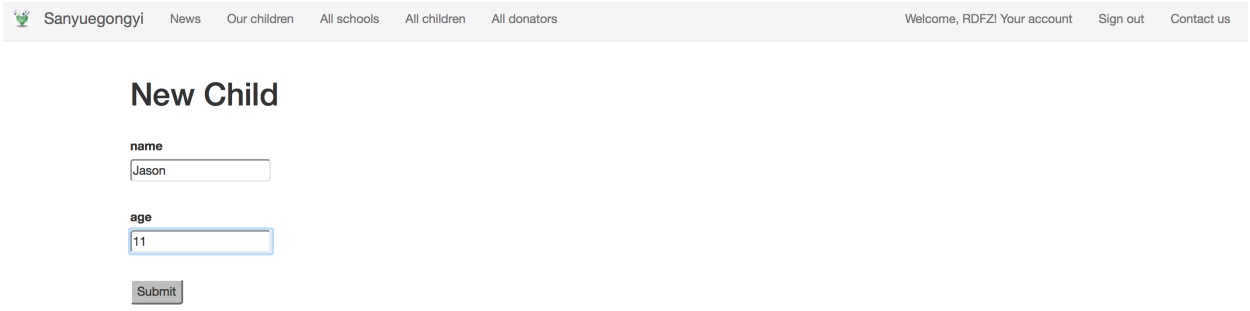
```
session["id"] = nil
session["type"] = nil
```

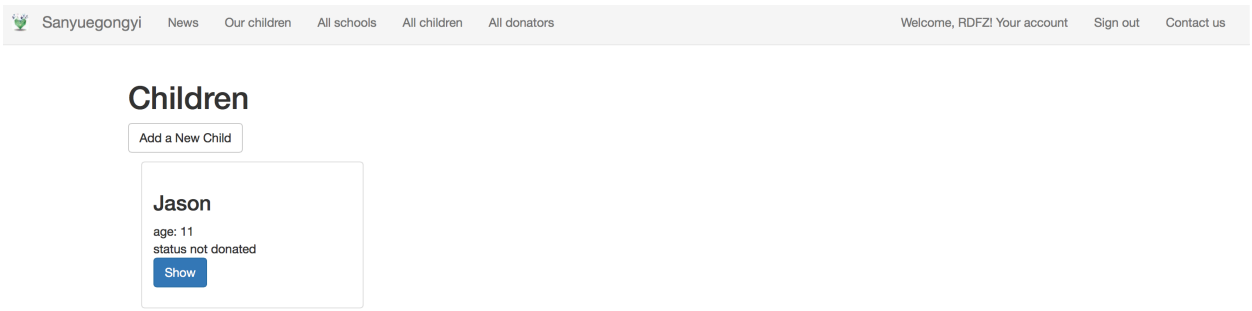
After cleaning the session, the sign out process cleans the navigation bar as expected.



Test 6

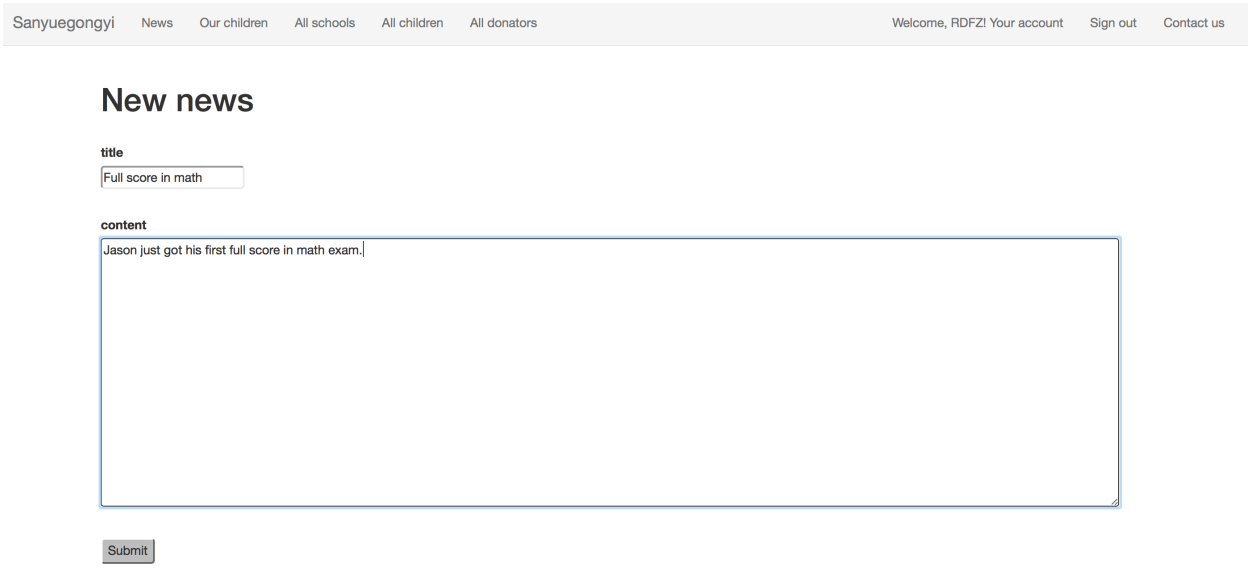
I create a child called Jason. After clicking submit, a child is successfully created in the database.

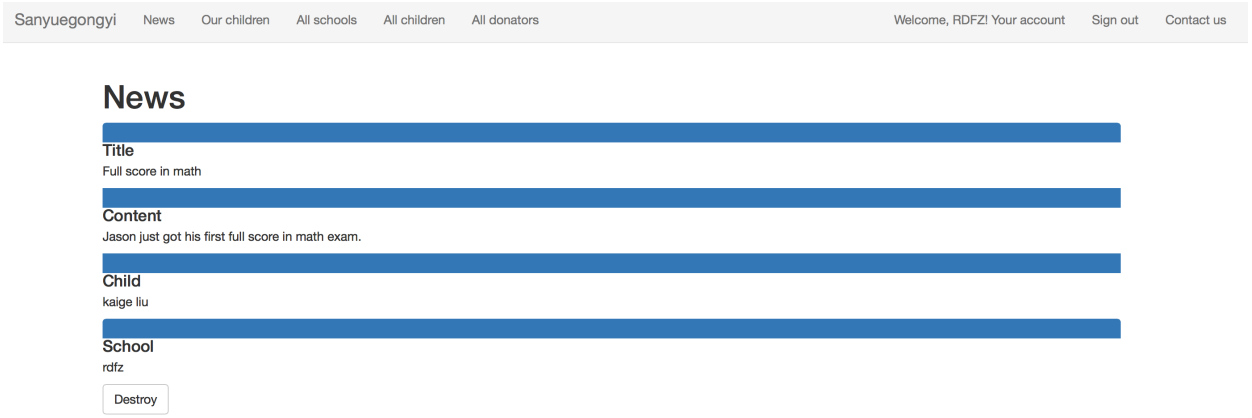


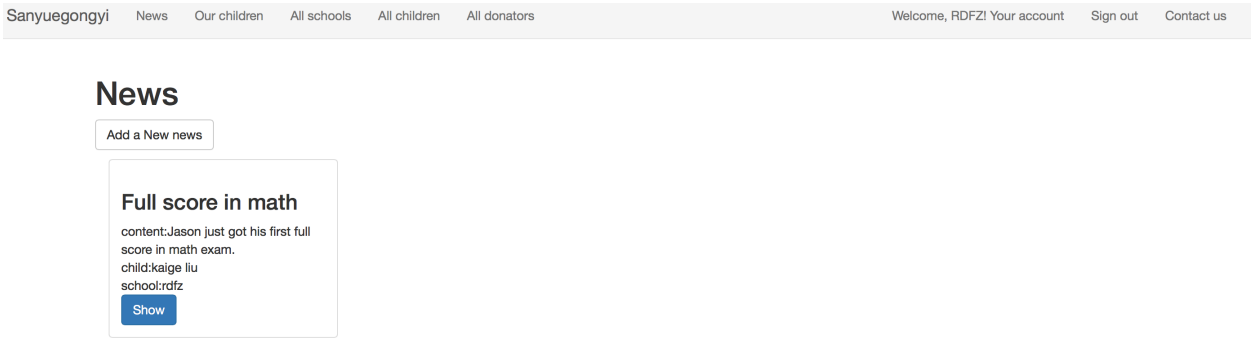


Test 7

I use similar operation as in Test 6. The new news is correctly stored.

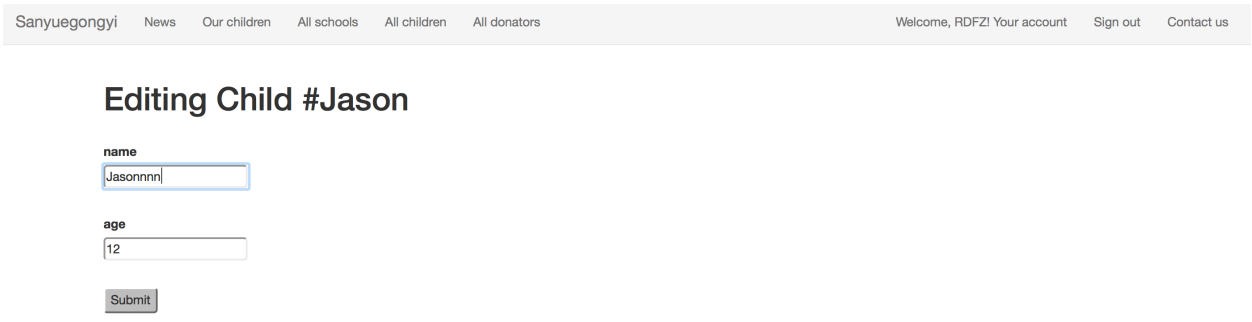


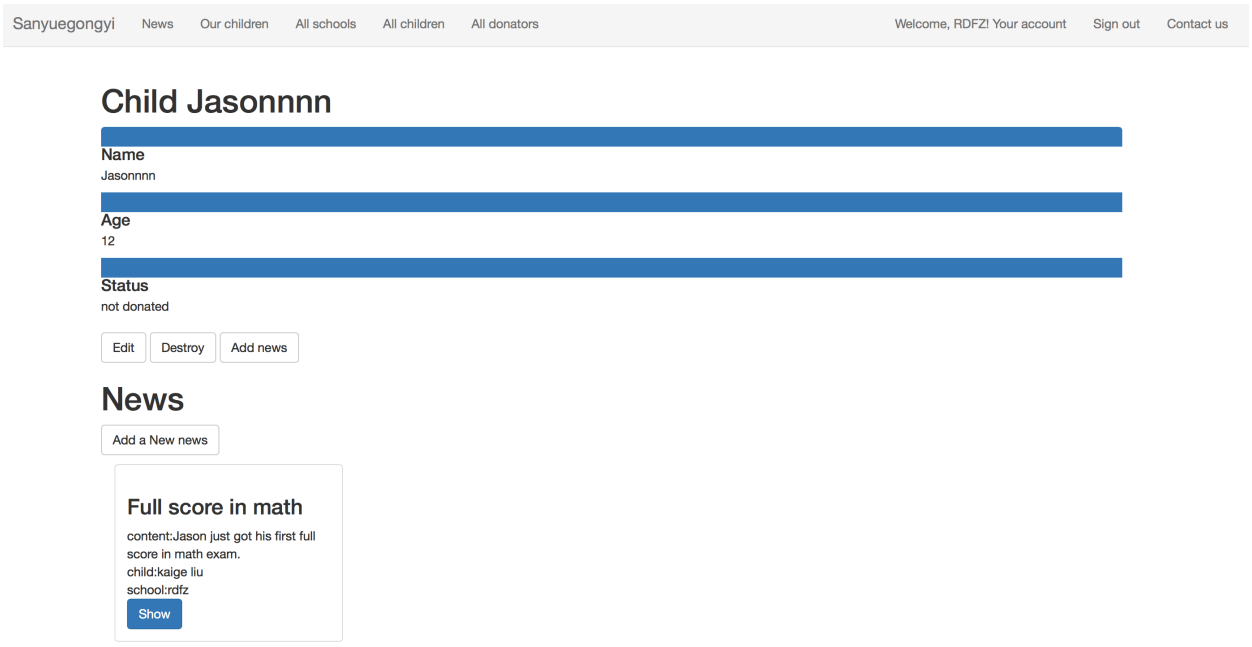




Test 8

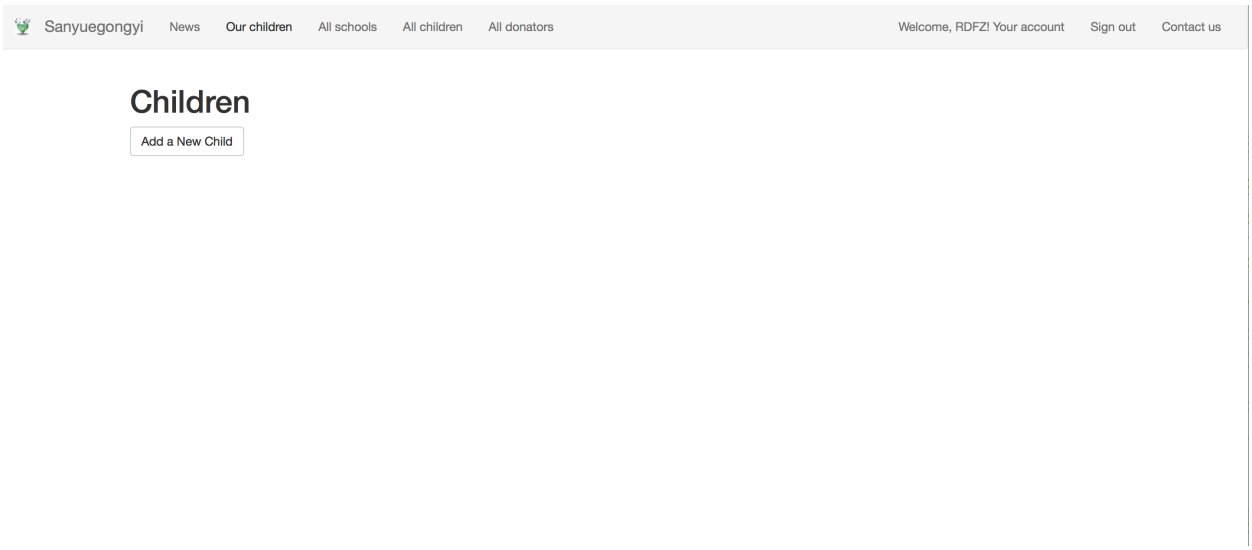
I edit the name of Jason to Jasonnn, and age to 12. After submitted, the system stores the new data successfully.





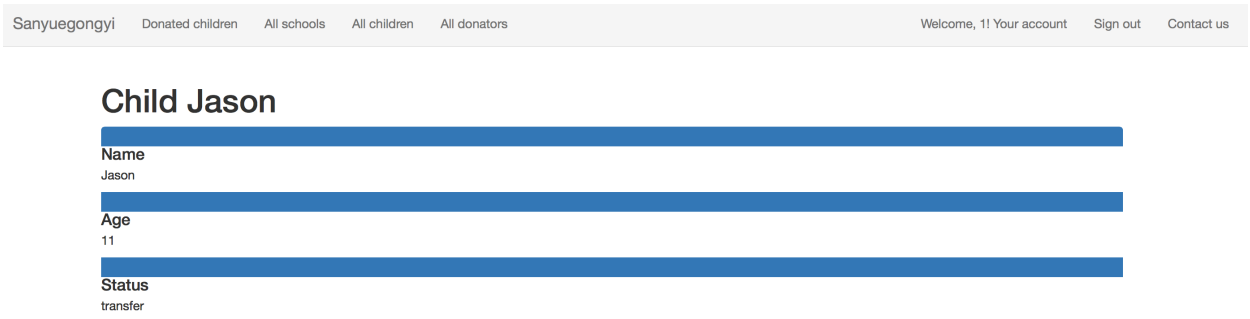
Test 9

After clicking destroy on the page of Jason, the child is removed from the database as shown.



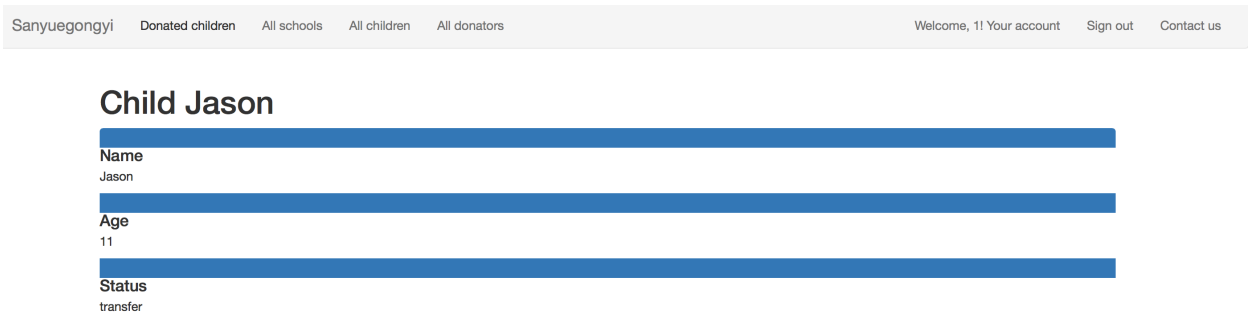
Test 10

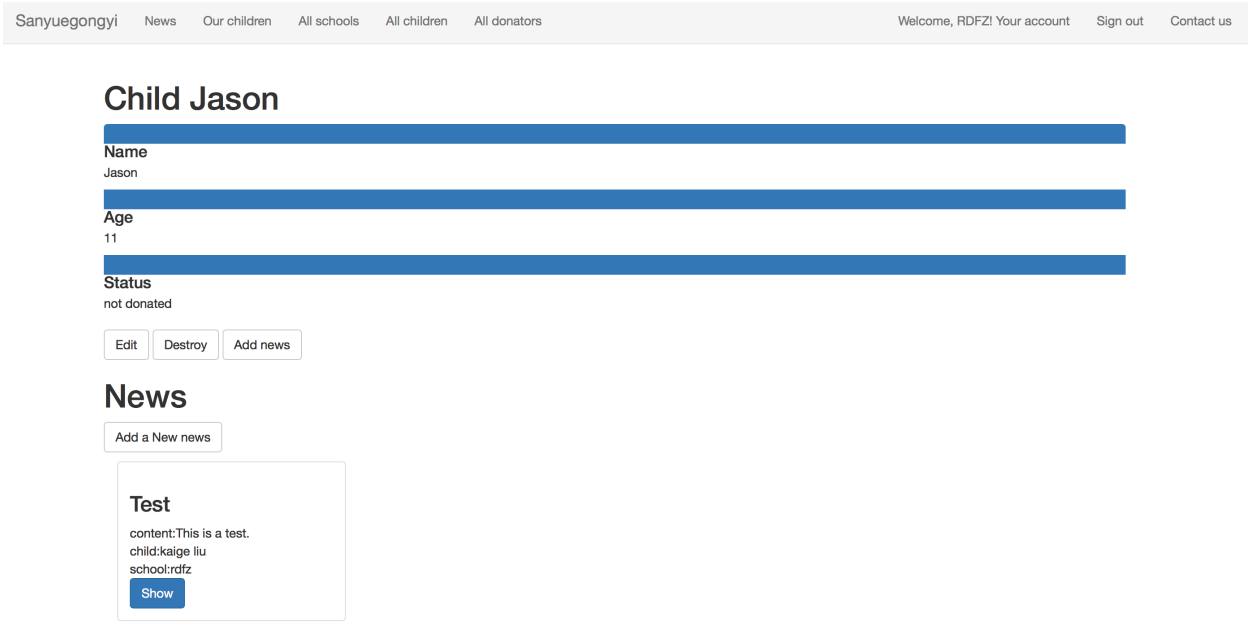
After I sign up as a donator, I enter the page of Jason, and click donate. Then, the status is changed to transfer, and the donate button disappears as expected.



Test 11

As the school of Jason, I create a news for Jason. Then, I sign in as a new donator. The donator has no access to the news.




Sanyuegongyi
News
Our children
All schools
All children
All donators
Welcome, RDFZ! Your account
Sign out
Contact us
Child Jason
Name
Jason
Age
11
Status
not donated
Edit
Destroy
Add news
News
Add a New news
Test
content:This is a test.
child:kaige liu
school:rdfz
Show

7. Installation

(1) Implementation plan

- Plan: Direct implementation
- Date: The system will be put into used in August 2016.
- Reason: As the system is well tested before the installation and is much more efficient than the original system, the system should be used as soon as possible.

(2) Staff training plan

- Date: 1st of May 2016 – 3rd of May 2016
- Reason: The 3-day Chinese vacation in May. Three days are sufficient for proper staff training.
- Method: Teach each staff about the allowed operations in the system. The documents are read carefully in the 3-day training.

(3) Evidence of User Testing

The following letter provided after the final testing of the application. It is also included in the "Evaluation" section.

Dear Du…,

After testing the system for three weeks, reading the documents and having our last meeting before final installation, I am glad to tell you that your system solves our problems, and meet the original requirements perfectly. All the errors are solved after our meetings. Not only the system meets all the original requirements, it is also extremely user friendly out of our expectation.

Thanks a lot for the design of the elegant user interface and clear web structure. Also, the database can be accessed fast and operated accurately. The security system is also well constructed.

Overall, the project is done well. The resultant system is exactly what we want. I am sure the system will be put into use, and the charity will benefit a lot from from your effort.

If a reference to this letter is needed, we would like to provide it gladly.

Sincerely,

H..., L...

The hand-written form in the following is also attached as the evidence of testing.

Objective	Whether satisfied
Create a database system, including the information of the donators, children and schools and the relations between different objects in the system.	✓
Allow donators and schools to sign up to the system with forms.	✓
Allow donators and schools to login to the system and view the existed information available to them.	✓
Allow donators to view the news about the children they donated and the messages sent by children and schools in the system.	✓
Allow schools to upload new information about children. Schools should be able to add children to the database, and add information to each child's profile, including text and photos.	✓
Allow donators to donate children online.	✓
Allow schools to accept donations online.	✓
Allow donators to see a list of the children not donated, as well children donated.	✓
Allow schools to see a list of children belonging to the school.	✓
Allow managers to have access to all the information, and to delete information or disable accounts if necessary.	✓
Allow each user in the system to edit his or her profile and password.	✓
Allow each user to click on a navigation bar on the top of the page.	✓
Create a security system that makes certain information only available to certain users. Users login to the system with their account name and password.	✓

Documentation

1. System Maintenance Documentation

(1) Forms and objects

	Object name	Type
URL: /new_child	name	Text Box
	age	Text Box
URL: /children/:id/edit	name	Text Box
	age	Text Box
URL: /new_news	title	Text Box
	content	Text Box
URL: /new_school	username	Text Box
	name	Text Box
	password	Text Box
	repassword	Text Box
	email	Text Box
	phone_number	Text Box
	address	Text Box
	card_number	Text Box
URL: /schools_edit	name	Text Box
	email	Text Box
	phone_number	Text Box
	address	Text Box
	card_number	Text Box
URL: /school_change_password	old	Text Box
	new	Text Box
	renew	Text Box
URL: /new_donator	name	Text Box
	password	Text Box
	repassword	Text Box
	email	Text Box
	phone_number	Text Box
URL: /donators_edit	name	Text Box

	email	Text Box
	phone_number	Text Box
URL: /donator_change_password	old	Text Box
	new	Text Box
	renew	Text Box

(2)Tables in database

Donator

Field name	Type	Validation	Description	Example
id	int	presence and unique	The automatically generated ID number of each donator and school.	92
user_name	string	presence and unique	The user chooses a user_name that can be used in the file. Users login with user_name.	JohnMu
name	string	presence	The user's real name.	John
password	string	presence	The account's password.	123456
email	string	presence	The user's email address.	123456@123.com
phone_number	string	presence	The user's phone number.	123456789

School

Field name	Type	Validation	Description	Example
id	int	presence and unique	The automatically generated ID number of each donator and school.	92
user_name	string	presence and unique	The user chooses a user_name that can be used in the file. Users login with user_name.	JohnMu
name	string	presence	The school's name.	ABC High School

password	string	presence	The account's password.	123456
email	string	presence	The school's email address.	123456@123.com
phone_number	string	presence	The school's phone number.	123456789
card_number	string	presence	The school's card number.	1234567890000000
address	string	presence	The school's address	No. 20, ABC Street, Beijing, China

Children

Field name	Type	Validation	Description	Example
id	int	presence	The automatically generated ID number of each d.	92
name	string	presence	The child's real name	John
age	int	presence	The child's age	5
status	string		It is the donation status of the child. The statuses are stated below.	not_donated
phone_number	string	presence	The user's phone number.	123456789

There are three statuses for a child:

- "not_donated" means a child is not yet donated.
- "on_transfer" means a child is donated by a donator, but the cash has not been received by the school.
- "donated" means a child is donated by a donator and the cash is received by school

News

Field name	Type	Validation	Description	Example
id	int	presence and unique	The automatically generated ID number of each donator and school.	92
title	string	presence	The title of a news.	New news!!!

content	string	presence	The content of a news.	The child got 100 in math exam!
child_id	int	presence	The child that the piece of news is related to.	12

Enrollment

Field name	Type	Validation	Description	Example
id	int	presence and unique	The unique id of an enrollment.	12
school_id	int	presence	The school of an enrollment.	JohnMu
child_id	int	presence	The child of an enrollment.	John

Donation

Field name	Type	Validation	Description	Example
id	int	presence and unique	The unique id of a donation.	12
donator_id	int	presence	The donator of an enrollment.	JohnMu
child_id	int	presence	The school of an enrollment.	John

(3) Key variables

Type	Variable name	Use
Dictionary	Session	It stores the user information and some background encrypted data. Session[user_name] and session[password] are examples of the use of session.
	Params	It stores the information that is passed between the controller and view. For example, when the form is filled with information in each field, all the information in the fields are passed through "params".
Child	@child	Pass a Child variable from the controller to the view.
School	@school	Pass a School variable from the controller to the view.
News	@news	Pass a News variable from the controller to the view.
Donator	@donator	Pass a Donator variable from the controller to the view.

(4) Key functions

Method name	use
Index	Show all the objects in the database with an html view.
Show	Show an object in the database with an html view.
New	Show an html form to create a new object in the database.
Create	Store the information in the "new" form.
Edit	Show an html form to edit an object in the database.
Update	Store the information in the "edit" form.
Destroy	Delete an object from the database.
Show_related_x	Show all x class objects related to the current object.
Account	Show the information of the sign in account.
Change_password	Change the password of the current account.

Update_password	Store the password in the “Change_password” view
Change_status	Change the status of the child, from not donated to on transfer.
Donate	Change the status of the child, from on transfer to donated.

(5) Possible adaptive maintenance

- Produce a local automatic backup of the database.
- Allow users to upload photos.
- Improve the user interface, by replacing the icons and button images.
- Add a search engine to the website.

(6) Backup and storage devices

The server should contain a hard drive with more than 100 Gb space to save database. To back-up the system, we need another hard drive with more than 100Gb space. Each week, the database is copied to the back-up disk.

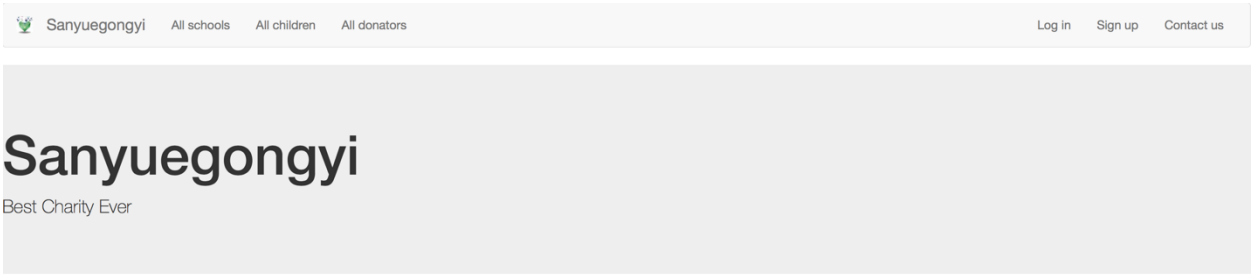
(7) Section reference

Most of the sections of the maintenance document are contained in this document previously.

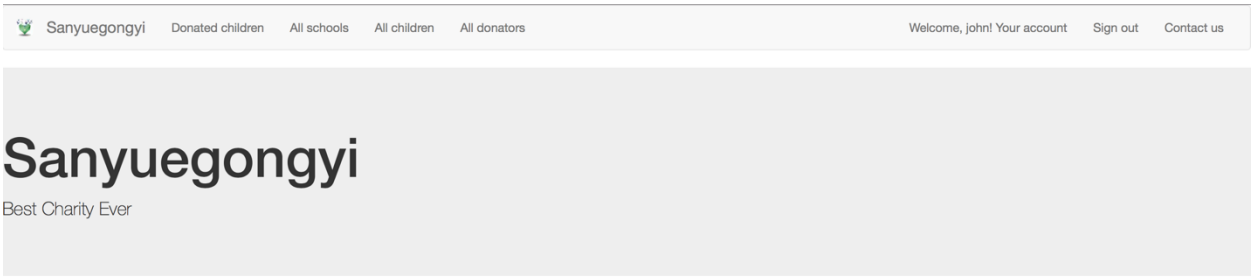
Section	Page number
Database design	15
Navigation design	21
Data flow diagram	22
Estimation of storage size	23
Top down design	27
Code	27

2. User Documentation

(1) Account operations



- Sign up for a new account on the top right corner of the home page.
- Remember your user name and password.
- Later, login to your account on the top right corner of the home page.



- After finishing your operation in your account or determining to switch to another account, click the sign out on the top right corner.

(2) Donation process

- First, a donator has to click the “donate” button of a child. The status of the child should be changed to “transfer”.
- Then, the donator transfer the cash through bank. Make sure the donator writes the name of both the donator and the child with the transfer.
- After the school receives the cash, the school should click the “accept donation” button to finish the donation. The status of the child should be changed to “complete”.

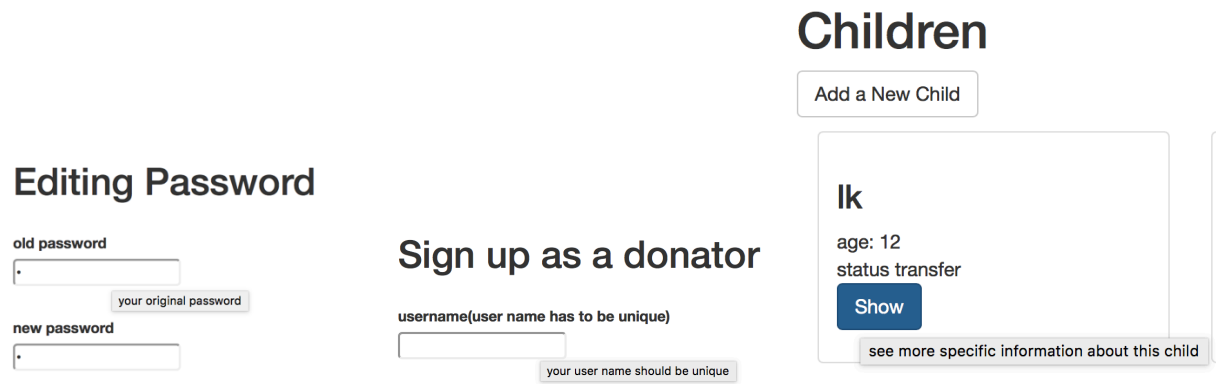
(3) Form filling

To create/edit a new news or child in the database, a form needs to be filled.

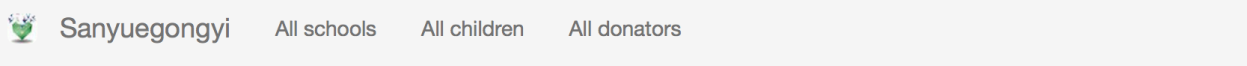
All the inputs in the website are text fields or text boxes. To fill in each field, click on it and type the text. Submit the form when the all the fields are filled. Before submission, check there is no mistake.

(4) On-screen help

If the user has any question on how to fill in a field, or how a button works, the user can get on-screen help by placing the cursor on the field or button for 3 seconds. A message will show on the screen to give terse instruction on the operation. Examples are listed below.



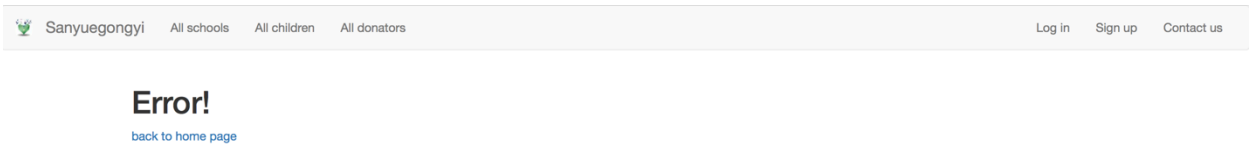
(4) Error guide



Mostly, there is a red error message with each error occurred in the system. The followings are examples of the error messages, their causes and solutions.

Error message	Cause	Solution
“Login fail.”	The user name does not exist or the password is incorrect.	Try again, or contact the manager if the password/username is forgotten.
“Please enter ____”	Certain necessary information is missing in the form submitted.	Fill in the necessary fields in the form.
“New passwords are not consistent.”	The two passwords typed are not consistent.	Type two consistent passwords.
“Incorrect old password.”	When the password is reset, the original password is incorrect.	Try again for a correct password, or contact the manager if the password/username is forgotten.
The username already exists.	The user name has been used.	Try with a new user name.
“Error.”	There might be error within the database.	Please contact us when it happens.

Users might also be directed to an error page.



If the the user is directed to the error page, the reasons might be:

- The user enter a URL not accessible to the user. In this case, follow the links on the website and prevent typing URL in the website.
- The problem with the input type. Please check if the inputs are all in the correct format.
- There are some errors with the server. In this case, contact the manager for help.

(5) Glossary

- Sign up: create a new account in the system.
- Login: enter the system as an existed account.
- Server: the machine on which the system is operated.
- URL: the link of the web page.
- Form: an interface form user to input information with graphical objects on screen.
- Text field: a graphical element that allows users to input text.

(6) Index

Evaluation

1. Evaluation with s objectives

Objective	Page of evidence	Description of evidence
Create a database system, including the information of the donators, children and schools and the relations between different objects in the system.	27-28	A database is generated with ruby code.
Allow donators and schools to sign up to the system with forms.	50	There is a sign up page in html.
Allow donators and schools to login to the system and view the existed information available to them.	49	There is a login page in html.
Allow donators to view the news about the children they donated and the messages sent by children and schools in the system.	35-37	There is a function called show related children in the donator controller. The news variable(news related to each child) is passed on to the view.
Allow schools to upload new information about children. Schools should be able to add children to the database, and add information to each child's profile, including text and photos.	39-41	The create action of the news is available to each school.
Allow donators to donate children online.	33-34	The children controller has a donate action, available to each donator
Allow schools to accept donations online.	33-34	The children controller has a change_status action, available to each school, to accept the donation to each

		specific child.
Allow donators to see a list of the children not donated, as well children donated.	33-34	The children controller can show all children in a list, with their status shown.
Allow schools to see a list of children belonging to the school.	39-41	The school controller can show all the children belonging to the school.
Allow managers to have access to all the information, and to delete information or disable accounts if necessary.	31,38,39,41…	On those pages, there is a selection to determine whether the user is a manager. If so, access to special actions like delete news will be shown.
Allow each user in the system to edit his or her profile and password.	35-41	Both schools and donators have the change_password action that allow users to change their passwords.
Allow each user to click on a navigation bar on the top of the page.	47-48	The navigation bar in the layout file is shown in each page.
Create a security system that makes certain information only available to certain users. Users login to the system with their account name and password.	35-41	To login to the system, passwords and user_name will be checked.

2. Evaluation with client response

(1) Original client response:

5 Mars 2016

Dear Du…,

After testing the system for three weeks, reading the documents and having our last meeting before final installation, I am glad to tell you that your system solves our problems, and meet the original requirements perfectly. All the errors are solved after our meetings. Not only the system meets all the original requirements, it is also extremely user friendly out of our expectation.

Thanks a lot for the design of the elegant user interface and clear web structure. Also, the database can be accessed fast and operated accurately. The security system is also well constructed.

Overall, the project is done well. The resultant system is exactly what we want. I am sure the system will be put into use, and the charity will benefit a lot from from your effort.

If a reference to this letter is needed, we would like to provide it gladly.

Sincerely,
H…, L…

(2) Response Interpretation

The response is made three weeks after the the testing started, so it is certain that the system is fully tested by sophisticated trained users of the system in the charity. In the email response, the client indicates that:

- The system is user friendly with good-looking GUI: "*Thanks a lot for the design of the beautiful user interface and clear web structure*".
- The system is efficient and accurate in database operations: "*the database can be accessed fast and operated accurately*".
- The web structure is clear and logical: "*Thanks a lot for the design of the beautiful user interface and clear web structure*".
- The security system is efficient and safe: "*The security system is also well constructed*".
- No further error is showing up in the system: "*All the errors are solved after our meetings*".

- All original requirements are met: "*I am glad to tell you that your system solves our problems, and meet the original requirements perfectly*".
- The system is going to be put into used for the charity: "*I am sure the system will be put into use, and the charity will benefit a lot from from your effort*".

Appendix

All of the following are scan of the content agreed and signed by the client.

5

A: The mangers are responsible for operating and recording cash flow. Also, they ask for information about students from the school and provided it to the donators. In the new system, we want to reduce the job of manager. The cash and information flow should be automatically proceeded on the Internet.

Q: In the new system, do we still need managers?

A: Yes. We still need the managers to have access to every piece of data in the system in case of emergency situation happens, or sudden change is needed to be made.

Q: Finally, what is the function of the school managers in the system?

A: School managers take money from the charity, and upload the current status of each students. They also have access to their own name list of students in need of donation.

Q: Do the school define the minimum donation amount?

A: No. The minimum donation amount is always equal in all schools.

Q: So basically, how is the current system operated?

A: The current system is operated manually. The managers have to do everything about information flow and cash flow between the donators and the schools. It always takes the managers extremely long time to do the repetitive jobs. The managers sometimes still make mistakes, which might cause bad conflicts.

Q: Currently, how many users are there?

A: We have 2 managers, 59 donators and 4 schools operating in the system. There are 1005 children overall. After the new system comes online, we might expect the number of donators and schools to double or even triple.

Q: How frequently do a school send a message about a child?

A: It depends. Averagely, I think it is twice a week.

Q: Thank you for your patience.

7. Alternative approaches

Alternative solutions were discussed with the client.

- Use an online system(e.g. Github) to synchronize the data in excel tables on both managers.
    - The problem of the discrepancy in data is partially solved with a synchronizing system. Still, the synchronization is not real-time, so before each updates, there might still be discrepancy in data.
    - The redundant efforts of the managers are not reduced.
    - The database is still not efficient in space.
    - The data is still stored in flat files, so it's slow to access the records.
    - The time for donators and schools to access to the data is still extremely long, as the information still has to be sent by the managers.
- Use Microsoft Access and Visual Basic developed software on a shared local server to replace Excel for database storage.
    - As the database is placed on a shared server, both managers have access to the database at the same time.
    - With proper development, the redundancies in database can be eliminated, and the effort of managers can be reduced.
    - In Microsoft Access, the efficiency of data access can be improved significantly.
    - The time for donators and schools to access to the data is still extremely long, as the information still has to be sent by the managers.
    - All operations in the system are still carried out by the two managers.

After the discussion of the previous possible alternatives, we found the major problem of the system is that donators and schools can not have access to the database by themselves. We came up with the final solution

- Develop a web based application with ruby and html. The application is stored in the server which is connected to the Internet. Different users have different access to the system by being identified with user name and password.
    - The system is highly secure with user names and passwords.

- The time for donators and schools to access to the data is a lot shorter, as they have their own access to the system online.
- The work of two managers is significantly reduced, as all users have their own access to the system.
- Operations like donations and enrollments can be done by donators and schools themselves, without the interaction of the managers.
- Rails, the web framework with ruby, is embedded with SQLite, a database system with efficient data access. As a result, the data can be accessed much more quickly.
- Web based application has no limitation on the device in use.
- Web based application with proper design is more user friendly.
- Development in ruby and html is more difficult than that in Visual Basic.

8. Requirement specification

(1)Database requirement

A database must be established to store the following properties:

a. Donators: donator information

b. Children: child information, whether donated, donator_id

c. Schools: information.

Donator information includes: name, user_name, password, email, phone number;

child information includes: name, age,school_id;

school information includes: name, email, phone number, address, card_number.

Further data types incude:

d. News: child_id, title, content, date.

(2) Functions requirement

The system must allow users to login to keep the system secure.

The system must allow different users to do operations, and the system will update the database meanwhile.

(3) A GUI is needed for the system, for the users to view and edit data.

(4) The system must work on the Internet, with a URL that can be accessed anywhere.

(5) Software requirement

Software	Reason
A web browser	The program is developed on a web environment. Any browser can access it with its URL.
A server Cent OS/Mac Operating system	The program has to be operated on a server with a operating system.
Rails	Rails needs to be installed on the server to make sure the system can be operated.

(6)Hardware requirement for development: personal computer(mac) and a test server.

	Device	Reason
Input device	Keyboard	To enter necessary information in fields.
	Mouse	To move the pointer on the page.
Output device	Monitor	Allow users and managers to see the actions of the system.
Storage	User storage 100Gb	The database of all users information needs to be stored online.

Software requirement:

Language used: ruby, html, javascript, SQLite and css.

Operating system: Mac OS X on mac, Linux Cent OS on the test server.

Developement environment: Rubymine.

Design

1. Objectives

After investigation and analysis, the final objectives of the system are listed as following:

- Create a database system, including the information of the donators, children and schools and the relations between different objects in the system.
- Create an graphical user interface, to allow the following:
    a. Allow donators and schools to sign up to the system with forms.
    b. Allow donators and schools to log in to the system and view the existed information available to them.
    c. Allow donators to view the news about the children they donated and the messages sent by children and schools in the system.
    d. Allow schools to upload new information about children. Schools should be able to add children to the database, and add information to each child's profil.
    e. Allow donators to donate children online.
    f. Allow schools to accept donations online.
    g. Allow donators to see a list of the children not donated, as well children donated.
    h. Allow schools to see a list of children belonging to the school.
    i. Allow the schools to edit and add children in its profile.
    j. Allow managers to have access to all the information, and to delete information or disable accounts if necessary.
    k. Allow each user in the system to edit his or her profile and password.
    l. Allow each user to click on a navigation bar on the top of the page.
- Create a security system that makes certain information only available to certain users. Users log in to the system with their account name and password.

Evaluation

1. Evaluation with s objectives

Objective	Page of evidence	Description of evidence
Create a database system, including the information of the donators, children and schools and the relations between different objects in the system.	27-28	A database is generated with ruby code.
Allow donators and schools to sign up to the system with forms.	50	There is a sign up page in html.
Allow donators and schools to login to the system and view the existed information available to them.	49	There is a login page in html.
Allow donators to view the news about the children they donated and the messages sent by children and schools in the system.	35-37	There is a function called show related children in the donator controller. The news variable(news related to each child) is passed on to the view.
Allow schools to upload new information about children. Schools should be able to add children to the database, and add information to each child's profile, including text and photos.	39-41	The create action of the news is available to each school.
Allow donators to donate children online.	33-34	The children controller has a donate action, available to each donator
Allow schools to accept donations online.	33-34	The children controller has a change_status action, available to each school, to accept the donation to each

		specific child.
Allow donators to see a list of the children not donated, as well children donated.	33-34	The children controller can show all children in a list, with their status shown.
Allow schools to see a list of children belonging to the school.	39-41	The school controller can show all the children belonging to the school.
Allow managers to have access to all the information, and to delete information or disable accounts if necessary.	31,38,39,41...	On those pages, there is a selection to determine whether the user is a manager. If so, access to special actions like delete news will be shown.
Allow each user in the system to edit his or her profile and password.	35-41	Both schools and donators have the change_password action that allow users to change their passwords.
Allow each user to click on a navigation bar on the top of the page.	47-48	The navigation bar in the layout file is shown in each page.
Create a security system that makes certain information only available to certain users. Users login to the system with their account name and password.	35-41	To login to the system, passwords and user_name will be checked.

2. Evaluation with client response

(1) Original client response:

5 Mars 2016

Dear Du...,

After testing the system for three weeks, reading the documents and having our last meeting before final installation, I am glad to tell you that your system solves our problems, and meet the original requirements perfectly. All the errors are solved after our meetings. Not only the system meets all the original requirements, it is also extremely user friendly out of our expectation.

Thanks a lot for the design of the elegant user interface and clear web structure. Also, the database can be accessed fast and operated accurately. The security system is also well constructed.

Overall, the project is done well. The resultant system is exactly what we want. I am sure the system will be put into use, and the charity will benefit a lot from from your effort.

If a reference to this letter is needed, we would like to provide it gladly.

Sincerely,

H..., L...

(2) Response Interpretation

The response is made three weeks after the the testing started, so it is certain that the system is fully tested by sophisticated trained users of the system in the charity. In the email response, the client indicates that:

- The system is user friendly with good-looking GUI: "*Thanks a lot for the design of the beautiful user interface and clear web structure*".
- The system is efficient and accurate in database operations: "*the database can be accessed fast and operated accurately*".
- The web structure is clear and logical: "*Thanks a lot for the design of the beautiful user interface and clear web structure*".
- The security system is efficient and safe: "*The security system is also well constructed*".
- No further error is showing up in the system: "*All the errors are solved after our meetings*".

- All original requirements are met: "*I am glad to tell you that your system solves our problems, and meet the original requirements perfectly*".
- The system is going to be put into used for the charity: "*I am sure the system will be put into use, and the charity will benefit a lot from from your effort*".

Objective	Whether satisfied
Create a database system, including the information of the donators, children and schools and the relations between different objects in the system.	✓
Allow donators and schools to sign up to the system with forms.	✓
Allow donators and schools to login to the system and view the existed information available to them.	✓
Allow donators to view the news about the children they donated and the messages sent by children and schools in the system.	✓
Allow schools to upload new information about children. Schools should be able to add children to the database, and add information to each child's profile, including text and photos.	✓
Allow donators to donate children online.	✓
Allow schools to accept donations online.	✓
Allow donators to see a list of the children not donated, as well children donated.	✓
Allow schools to see a list of children belonging to the school.	✓
Allow managers to have access to all the information, and to delete information or disable accounts if necessary.	✓
Allow each user in the system to edit his or her profile and password.	✓
Allow each user to click on a navigation bar on the top of the page.	✓
Create a security system that makes certain information only available to certain users. Users login to the system with their account name and password.	✓

12 12 2