project part2

Siliang Liu sl7109

Content

- Introduction to runtime environment setup and code deployment methods
- Website control flow diagram and the core module implementation method overview
- To improve the deficiencies of project part1
- Front-end interface display
- Introduction to website code directory and core page implementation methods
- Introduction to measures to ensure website security

Develop the website according to the requirements in project part2. First, introduce the runtime environment setup and code deployment methods:

Operating System:

- ubuntu16.04

Website running Software:

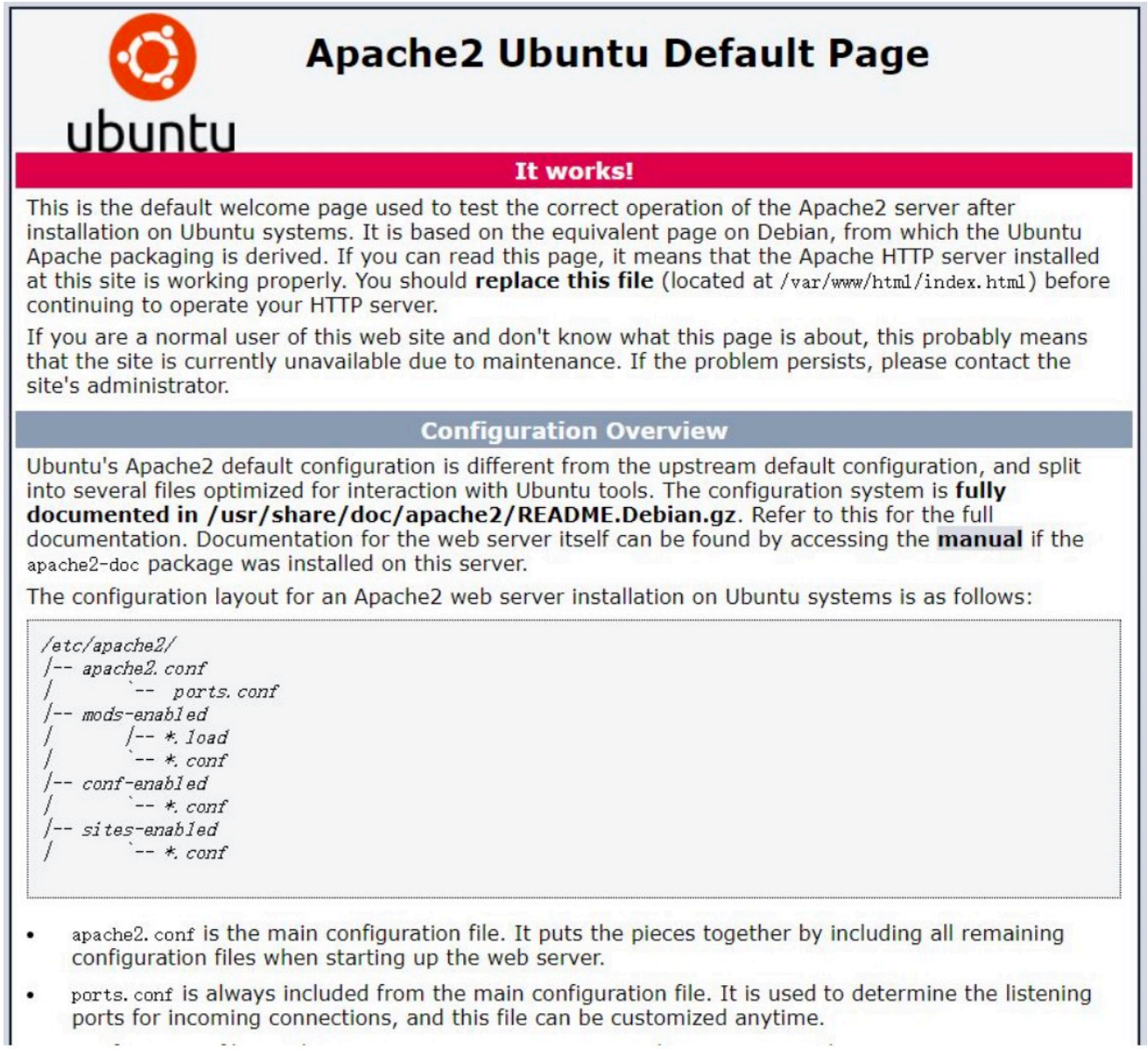
- mysql Ver 14.14 Distrib 5.7.28
- PHP 7.2.26-1
- Apache/2.4.18

Use mysql+php+bootstrap4 to develop the website, the specific installation statements are as follows:

The statements for installing apache2 are as follows:

```
sudo apt-get update
sudo apt-get install apache2
```

After the installation of apache2, visited the website. The screenshot shows that the installation was successful:



Install mysql using the following statements:

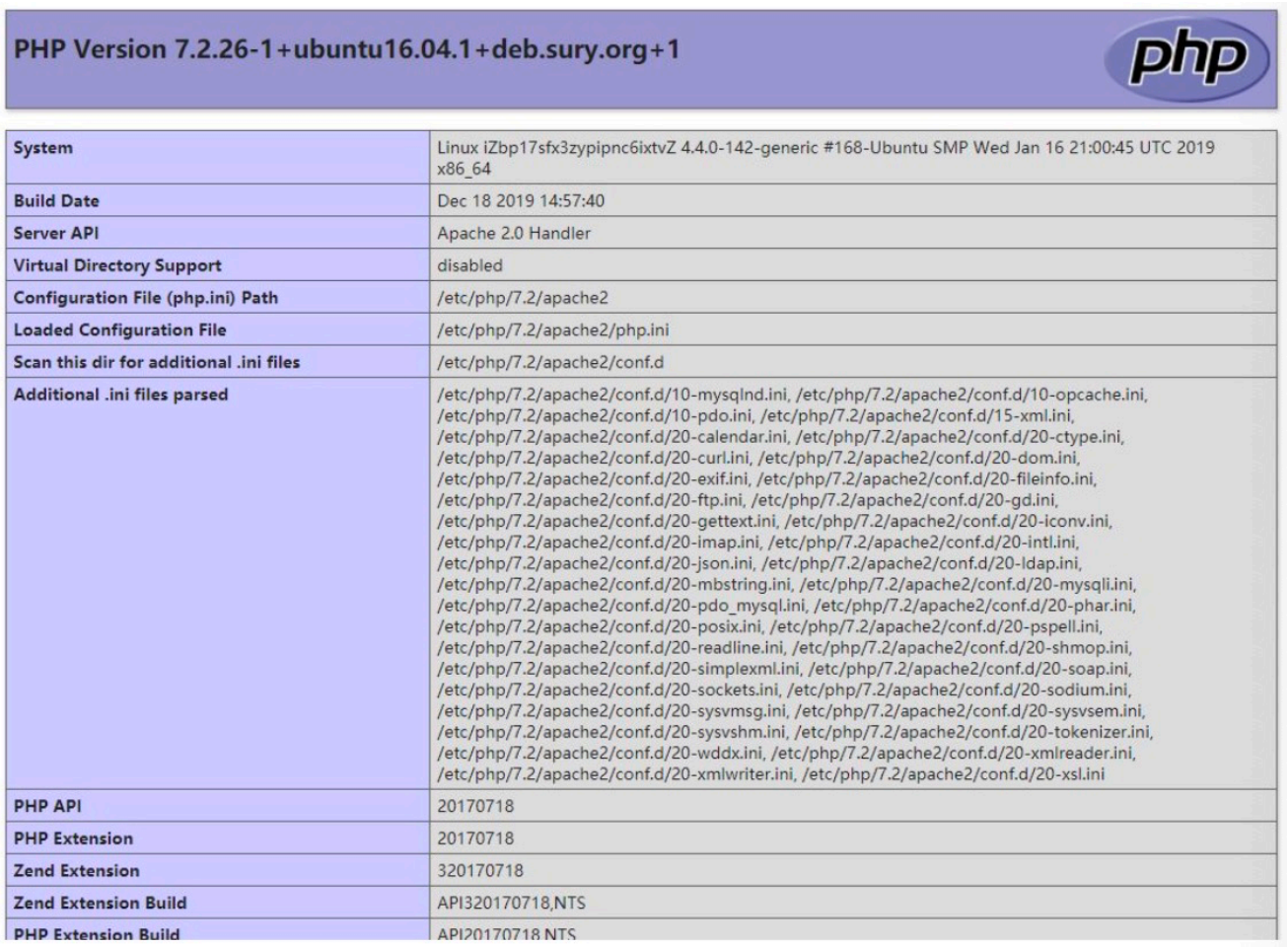
```
sudo apt-get install mysql-server
```

```
tmp@iZbp17sfx3zyp1pnc61xtvZ:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1557
Server version: 5.7.28-0ubuntu0.16.04.2 (Ubuntu)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Use the following statements to install PHP and related dependencies:

```
sudo apt-get install php libapache2-mod-php php-mcrypt php-mysql
```

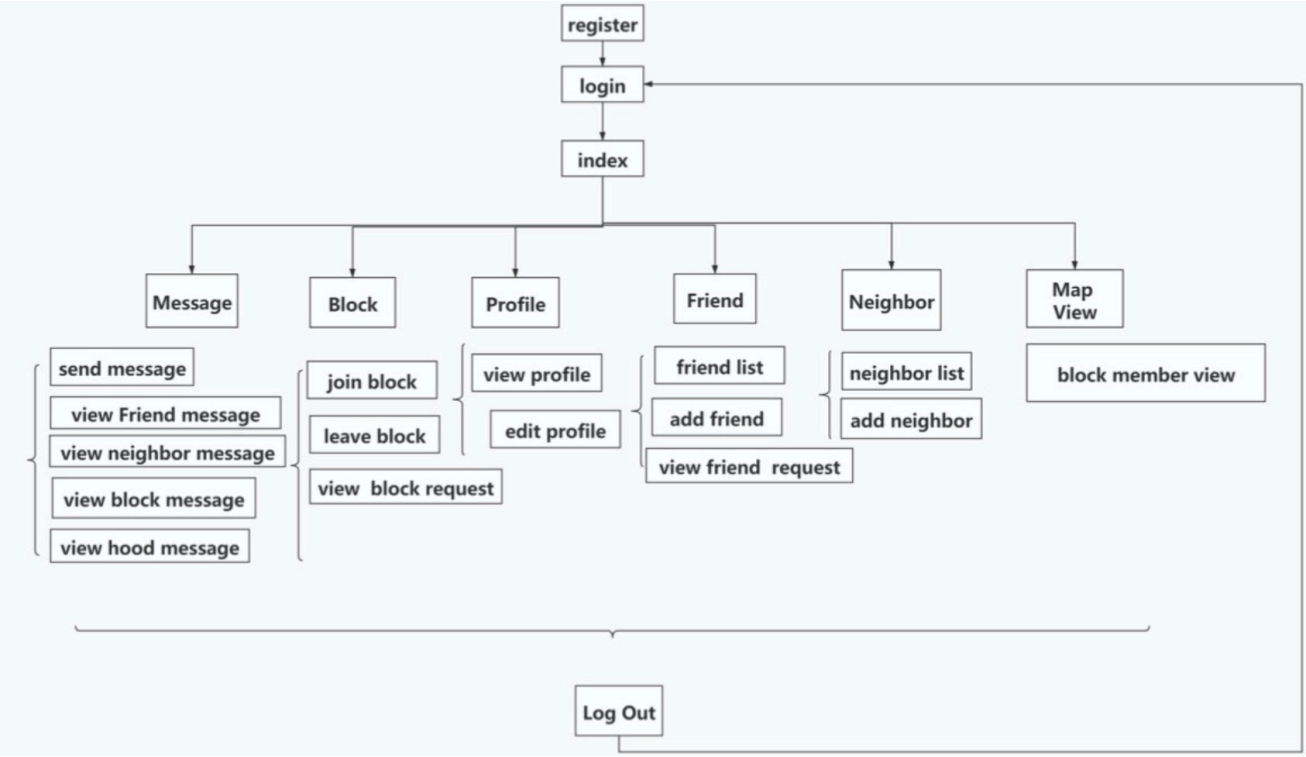
After the installation is complete, use the phpinfo() function to test if the PHP installation is successful, as shown below.



PHP Version 7.2.26-1+ubuntu16.04.1+deb.sury.org+1

System	Linux iZbp17sfx3zypipnc6ixtvZ 4.4.0-142-generic #168-Ubuntu SMP Wed Jan 16 21:00:45 UTC 2019 x86_64
Build Date	Dec 18 2019 14:57:40
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/apache2
Loaded Configuration File	/etc/php/7.2/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/apache2/conf.d
Additional .ini files parsed	/etc/php/7.2/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.2/apache2/conf.d/10-opcache.ini, /etc/php/7.2/apache2/conf.d/10-pdo.ini, /etc/php/7.2/apache2/conf.d/15-xml.ini, /etc/php/7.2/apache2/conf.d/20-calendar.ini, /etc/php/7.2/apache2/conf.d/20-ctype.ini, /etc/php/7.2/apache2/conf.d/20-curl.ini, /etc/php/7.2/apache2/conf.d/20-dom.ini, /etc/php/7.2/apache2/conf.d/20-exif.ini, /etc/php/7.2/apache2/conf.d/20-fileinfo.ini, /etc/php/7.2/apache2/conf.d/20-ftp.ini, /etc/php/7.2/apache2/conf.d/20-gd.ini, /etc/php/7.2/apache2/conf.d/20-gettext.ini, /etc/php/7.2/apache2/conf.d/20-iconv.ini, /etc/php/7.2/apache2/conf.d/20-imap.ini, /etc/php/7.2/apache2/conf.d/20-intl.ini, /etc/php/7.2/apache2/conf.d/20-json.ini, /etc/php/7.2/apache2/conf.d/20-ldap.ini, /etc/php/7.2/apache2/conf.d/20-mbstring.ini, /etc/php/7.2/apache2/conf.d/20-mysqli.ini, /etc/php/7.2/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.2/apache2/conf.d/20-phar.ini, /etc/php/7.2/apache2/conf.d/20-posix.ini, /etc/php/7.2/apache2/conf.d/20-pspell.ini, /etc/php/7.2/apache2/conf.d/20-readline.ini, /etc/php/7.2/apache2/conf.d/20-shmop.ini, /etc/php/7.2/apache2/conf.d/20-simplexml.ini, /etc/php/7.2/apache2/conf.d/20-soap.ini, /etc/php/7.2/apache2/conf.d/20-sockets.ini, /etc/php/7.2/apache2/conf.d/20-sodium.ini, /etc/php/7.2/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.2/apache2/conf.d/20-sysvsem.ini, /etc/php/7.2/apache2/conf.d/20-sysvshm.ini, /etc/php/7.2/apache2/conf.d/20-tokenizer.ini, /etc/php/7.2/apache2/conf.d/20-wddx.ini, /etc/php/7.2/apache2/conf.d/20-xmlreader.ini, /etc/php/7.2/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.2/apache2/conf.d/20-xsl.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS

According to the function description of the website in project part2, the function flow diagram of the website is drawn as follows:



According to the website function description diagram, the core functions of the website are as follows:

- **User registration, login and logout module**
- **Message processing module**

  send message：Users can send messages
  view Friend message： Users view messages from their friends
  view neighbor message：Users view messages from their neighbors
  view block message： Users view messages of their block
  view hood message：Users view messages of their hood
- **Profile processing module**

  view profile：Users view their profiles
  edit profile：Users edit their profiles
- **Friend processing module**

  friend list：Users view their friends list

  add friend：Users view other users in the same hood and add another user as a friend

  view friend request：Users view other users' friend application requests and decide to agree or reject.
- **Neighbor processing module**

neighbor list： Users view the list of all the neighbors they added
add neighbor： Users view the members of the entire block and add other members as neighbors.

- **Block processing module**

  join block： User applies to join a block
  leave block： User leaves the block he/she has joined
  view block request： User view requests from other users to join the block

- **Map View module**

  block member view： Users view the location information of the members in the same block on a Google map.

The implementation ideas of the core functions of the website are as follows.

- **User registration, login and logout module**

  The information of users was recorded in the `User` table when users sign up. The hash value of the password of users was also recorded together with other information in the database. When a user signs up, there are two levels of address information which can be specified: city and street.

  When a user logs in, the user fill in the user name and password which are sent to the backend by post request. The backend check whether the user name and the hash value of the password the user types are the same with the value recorded in the database. If so, the user is allowed to log in.

- **Message processing module**

  send message：

  This submodule is responsible for processing the user's request to send information. The user can fill in the title and the content of a message and select a kind of receiver of the message which could be the whole hood, the whole block, all the friends of the user, all the neighbors of the user, or just a specific person in the friends or neighbors of the user.

  At the same time, the user can decide whether or not to expose the location information of the message. The default location information of the message is consistent with the user's registration address information, which can be further improved.

  view Friend message：

  This is where the user views all unread messages from his/her friends. When the user selects this option, a thread is started to query the `message` table and retrieve all unread messages based on user id and request information classification.

  view neighbor message：

  This is where the user views all unread messages from his/her neighbors. When the user selects this option, a thread is started to query the `message` table and retrieve all unread messages based on user id and request information classification.

  view block message：

The user can view the block shared by the whole block here. When the user selects this option, a thread is started to query the `message` table and retrieve all unread messages based on user id and request information classification.

view hood message：

The user can view the message shared by the whole hood here. When the user selects this option, a thread is started to query the `message` table and retrieve all unread messages based on user id and request information classification.

- **Profile processing module**

  view profile：

  Users can view their profile here.
  edit profile：

  Users can edit their profile here. Users can choose whether to upload pictures, edit self and family introduction.

  The front end submits the information through a post request to the back end, which modifies the `Profile` data table based on the information provided by the user to save the user's request.For image uploads, the image is stored in a directory and the relative path of the image is stored in the database.

- **Friend processing module**

  friend list：

  Users view their friends list here.

  add friend：

  This is where the user views the users in the same hood and adds another user as a friend.

  The backend first queries the hood id where the user is, then queries all users that belong to the hood, then queries all friends ids of the user and user ids that have sent friend requests to. All user ids in the hood exclude the user's existing friend ids and ids have sent a friend request, and the rest are the ids that the user can apply to as a friend.

  The front-end uses PHP to output HTML, dynamically generates the front-end page according to the number of friends, and displays the users that can be added as friends.

  view friend request：

  Here the user views the friend application request sent by other users and decides to agree or reject a request.

- **Neighbor processing module**

  neighbor list：

  The user view the entire block's members and adds other members as neighbor.

  add neighbor：

The backend first queries the user's block id, then queries all the users that belong to that block, and then queries all the neighbors ids for that user. The first part users exclude the second part, leaving only the users ids that the user can add as neighbors.

The front-end uses PHP to output HTML, dynamically generates the front-end page according to the number of database queries, and display the users that can be added as neighbors.

- **Block processing module**

  join block：

  Users here to apply to join a block. Before the user can send a join request, first check to see if the user has already been in a block. If so, the user has to exit the existing block before joining a new one. The user can view a list of block, and select which block to send a join request. If the requested block has no members, the user can directly become the first member of this block after sending a join application. If members already exist in the block, then need to wait for the reply of the existing members in the block.
  leave block：

  This is where the user leaves the block. After the user leaves, all of the neighbors and friends that were added previously become invalid. However, the user is allowed to see the message generated in the previous block, but nor the message generated later.
  view block request：

  If the user views the application sent by another user to join the block, the user can choose to accept. If the user views the block join application of another user, but does not choose agree, the back-end system of the website will automatically assume that the user has rejected the application.

- **Map View module**

  block member view：

  The display of the location of the same block member with the user is realized by calling the Google map API. The pages are dynamically generated and sent to the front end for display by using JS code calling Google Map API generated by PHP loop.

For the website features designed, a small part of the part1 of the project(backend design) was modified to meet unexpected requirements as follows:

- A `block_request_list table` was added to record the status of the process in which the user applied to join a block.

```
CREATE TABLE IF NOT EXISTS `block_request_list`(
   `uid` INT NOT NULL,
   `bid` INT NOT NULL,
   `status` INT NOT NULL
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- The Street data was improved, and the latitude and longitude mark data was added to the Street table. After the improvement, the relative mysql sentences of Street are as follows:

```
#Street(sid,sname)
CREATE TABLE IF NOT EXISTS `Street`(
   `sid` INT UNSIGNED AUTO_INCREMENT,
    `sname` VARCHAR(40) NOT NULL,
    `lat` double,
    `lon` double,
   PRIMARY KEY ( `sid` )
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Street's data is refined as follows (the exact data obtained from the Google map API):

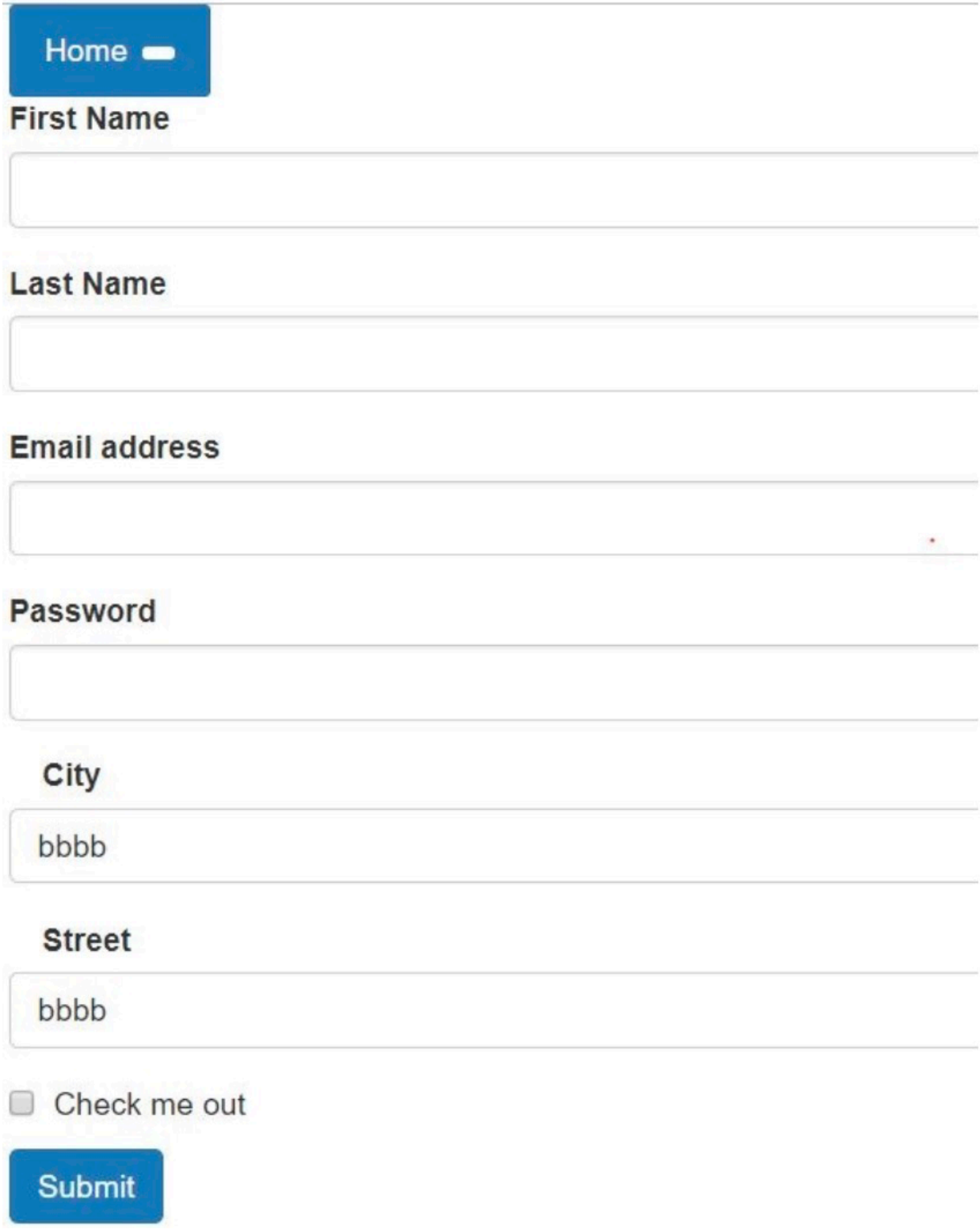
streetid	streetname	lat	lon
1	Washington Blvd	33.992599	-118.443713
2	Glencoe Ave	33.993201	-118.444785
3	Beach Ave	33.991924	-118.443003

Street test data inset statement was modified as follows:

```
INSERT INTO Street (sname,lat,lon) VALUES  ( "Washington
Blvd",33.992599,-118.443713);
INSERT INTO Street (sname,lat,lon) VALUES  ( "Glencoe
Ave",33.993201,-118.444785);
INSERT INTO Street (sname,lat,lon) VALUES  ( "Beach Ave",33.991924,-118.443003);
```

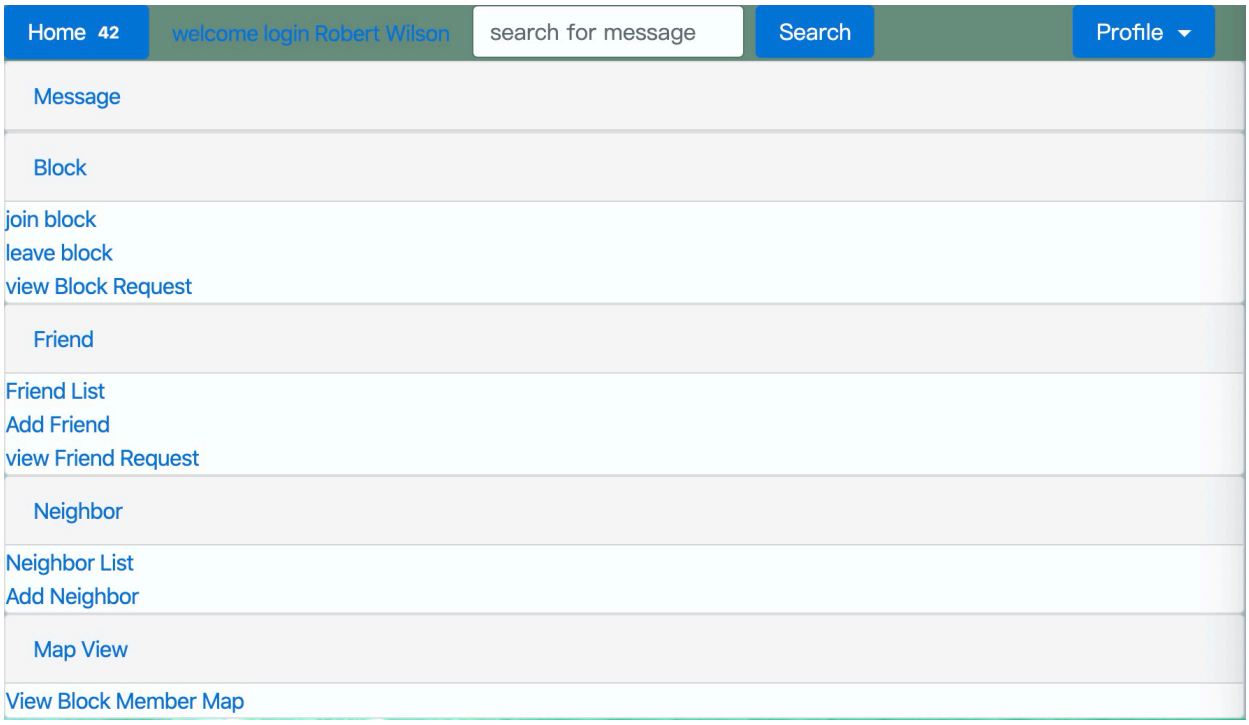
Here are some examples of how websites implement core interfaces:

- User registration interface

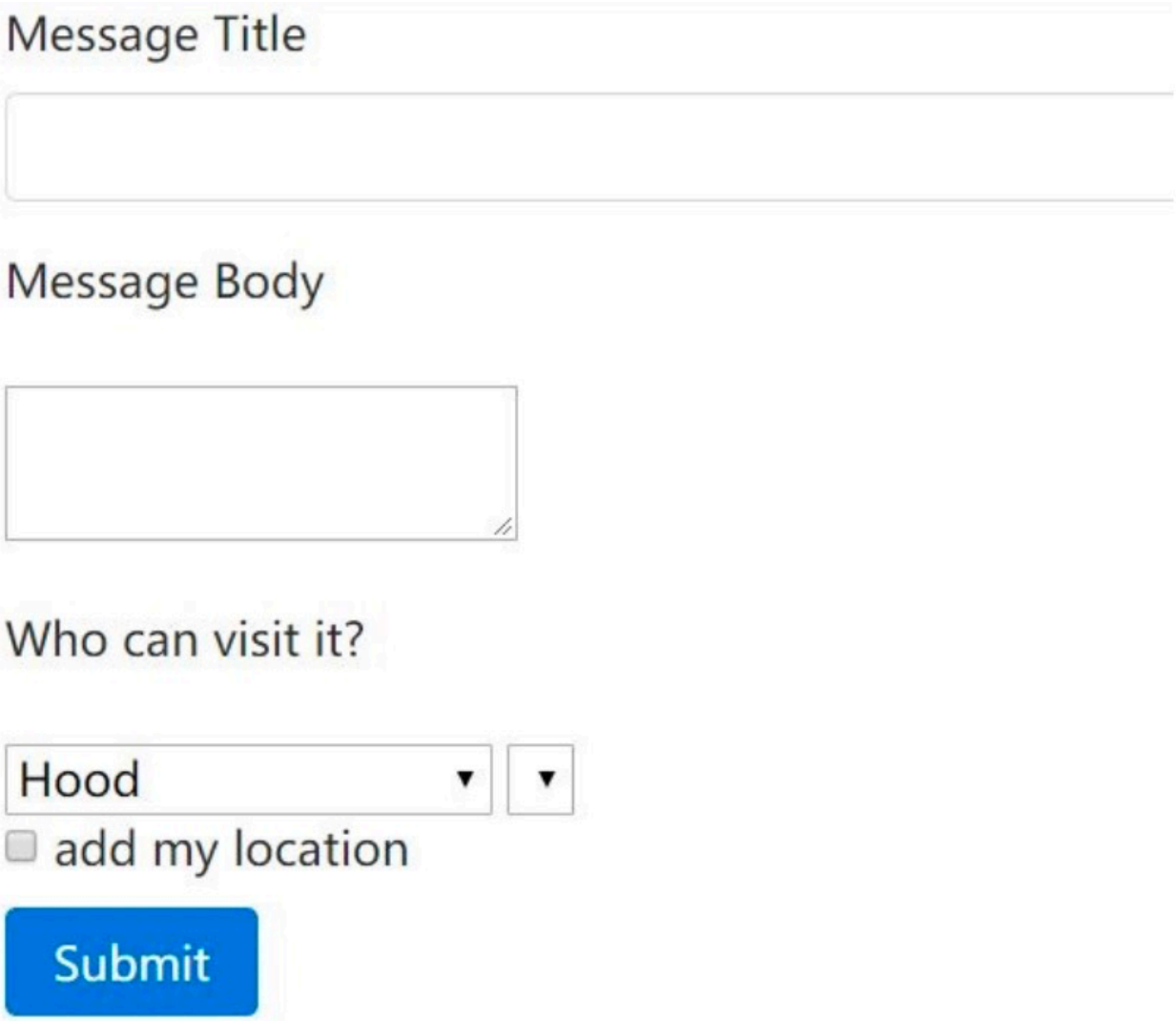


- User login interface

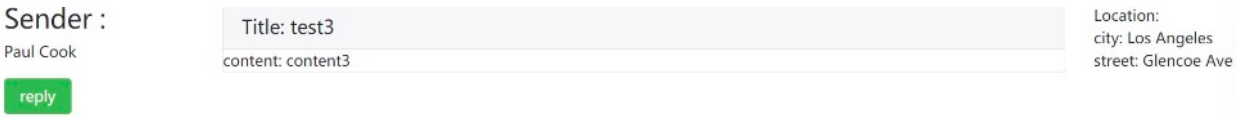
- The main interface after the user logs in



- The interface where users send message



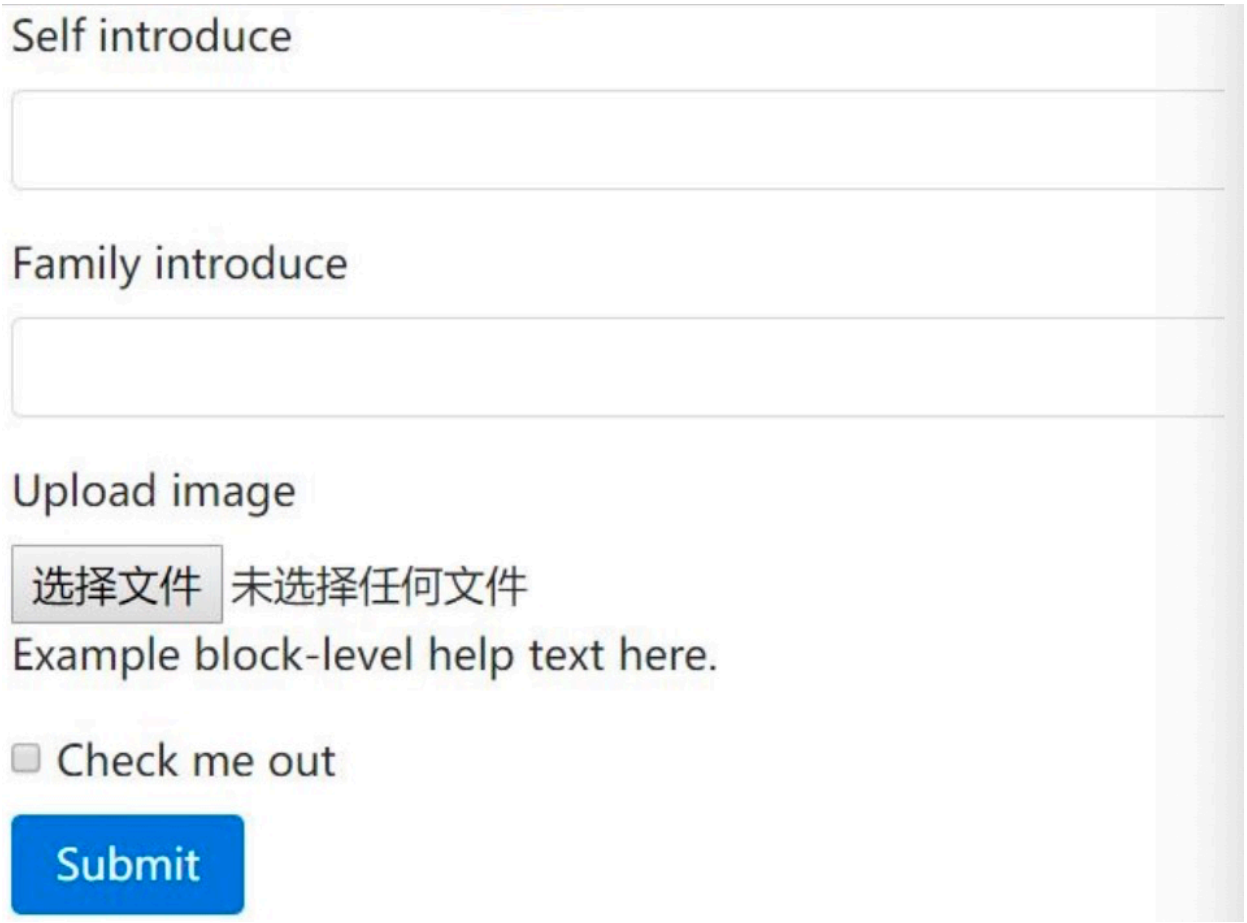
- The interface where users read message



- The interface where users add friends

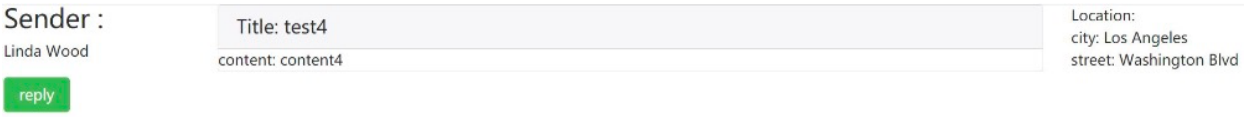
- The interface where users add neighbors

- The interface where users edit profile

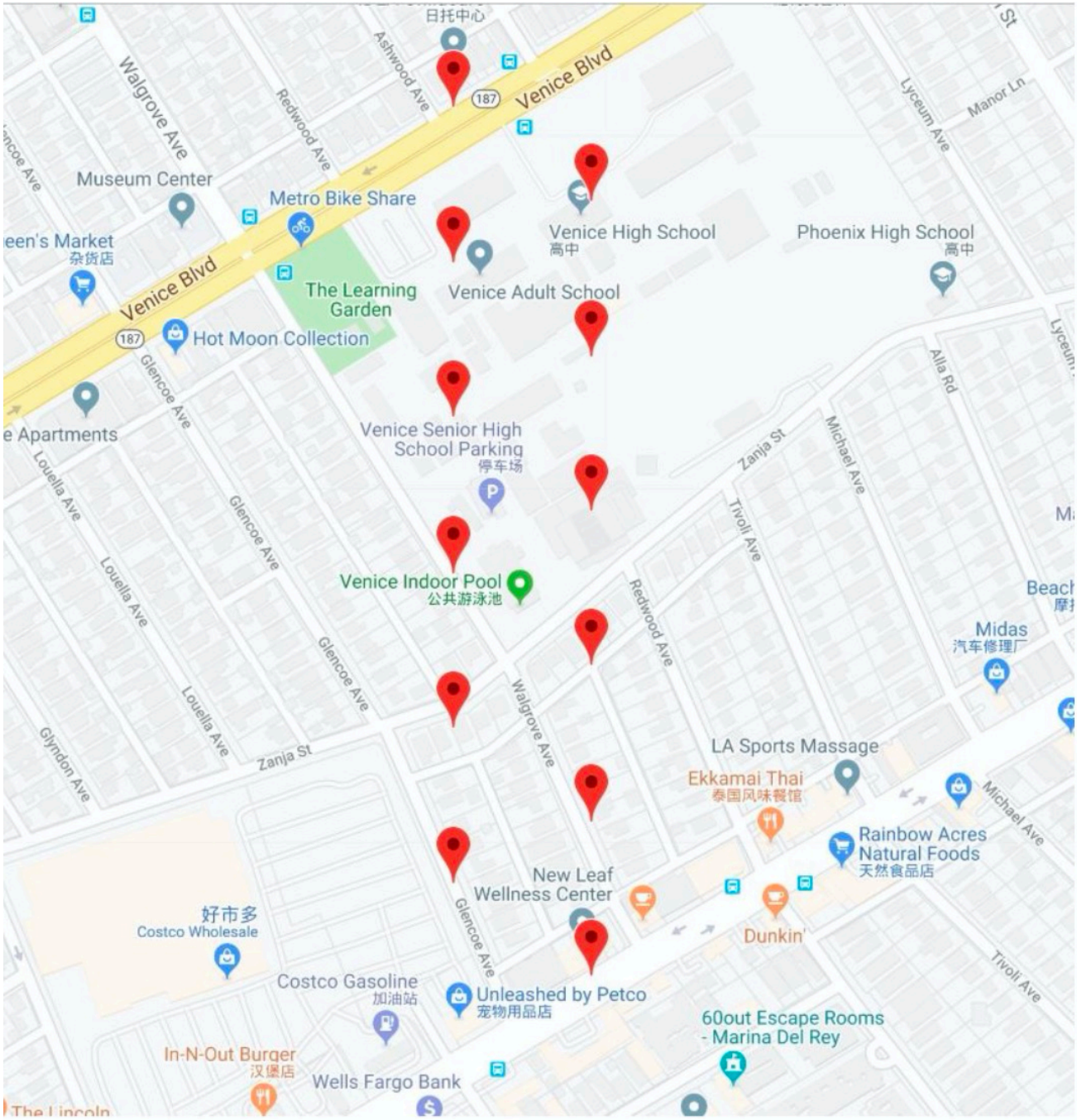


- The interface where users view friend application request

- The interface where users view block message



- The map view where users can view the location information of other users in the same block



- The join block interface shows the available blocks(join options according to the location)

New Leaf Wellness Center	join
Pause Studio	join
The Studio (MDR) East	join
Unleashed by Petco	join
OneWest Bank	join
Marina Collection	join

The website code directory and corresponding stored contents are as follows:

```
|--dbs  db operation
|--www |-----  js
       |-----  css
       |-----  font
       | web  html and php
```

Website code deployment:
deploy the website code in the website root directory of the web server, configure the correct mysql connection information to run the code.

The detailed website code directory is as follows:

```
├── dbs
│   ├── block_op.php
│   ├── conn_db.php
│   ├── friend_op.php
│   ├── login_db.php
│   ├── map_op.php
│   ├── neighbor_op.php
│   ├── profile_db.php
│   ├── profile.php
│   ├── register_db.php
│   └── view_mess.php
├── index.html
├── info.php
└── src
    ├── 1
    ├── add_friend.php
    ├── add_friend_request.php
    ├── add_neighbor.php
    ├── add_neighbor_request.php
    ├── css
    ├── deal_friend_request.php
    ├── edit_profile.html
    ├── edit_profile.php
    ├── fonts
    ├── friend_list.php
    ├── geocoordinates.php
    ├── index.html
    ├── join_block2.php
    ├── join_block.php
    ├── js
    ├── leave_block.php
```

```
    ├── login.html
    ├── login.php
    ├── logout.php
    ├── main.php
    ├── map_test.php
    ├── map_view.php
    ├── neighbor_list.php
    ├── photo
    ├── register.html
    ├── register.php
    ├── reply_block.php
    ├── reply_friend.php
    ├── reply_hood.php
    ├── send_message2.php
    ├── send_message.php
    ├── test1.html
    ├── test2.php
    ├── test.html
    ├── test.php
    ├── view_block_mess.php
    ├── view_block_request.php
    ├── view_friend_mess.php
    ├── view_friend_request.php
    ├── view_hood_mess.php
    ├── view_neighbor_mess.php
    ├── view_profile2.php
    └── view_profile.php
```

The description of the function of the code in the figure:

- All queries for the database update and delete are completed here, for different website function modules, build different PHP files, to achieve the classification and reuse of the code.

```
├── dbs
│   ├── block_op.php
│   ├── conn_db.php
│   ├── friend_op.php
│   ├── login_db.php
│   ├── map_op.php
│   ├── neighbor_op.php
│   ├── profile_db.php
│   ├── profile.php
│   ├── register_db.php
│   └── view_mess.php
```

- Realize adding friends and neighbors

```
    ├── add_friend.php
    ├── add_friend_request.php
    ├── add_neighbor.php
    ├── add_neighbor_request.php
```

- Realize the front-end interface of user editing profiles

```
    ├── edit_profile.html
    ├── edit_profile.php
```

- Pages where users view various types of information

```
— view_block_mess.php
    ├── view_block_request.php
    ├── view_friend_mess.php
    ├── view_friend_request.php
    ├── view_hood_mess.php
    ├── view_neighbor_mess.php
    ├── view_profile2.php
    └── view_profile.php
```

The following are specific implementations of some of the core pages

`view_friend_mess.php`

Use session to get the user id, and get the unread messages sent by all friends that the user receives from the message data table. Use PHP to generate HTML automatically to return the unread messages to the user's browser. The code is as follows:

```
foreach($re as $k=>$val)
{
   echo "<div class='row'>  <div class='col-md-2'> <h3>Sender : </h3>
<p>".$val["uname"] ;           echo "</p>
    <a href='reply_friend.php?id=".$val["sender"];
   echo" '><button type='button' class='btn btn-success' contenteditable='true'
name=".$val["sender"];
    echo ">reply</button></a>
    </div>
    <div class='col-md-8'>
        <div class='card'>
            <h5 class='card-header'>
            Title:   ".$val["title"];

    echo "  </h5>
```

```
    <div class='card-body'>
        <p height ='800' class='card-text'>  content:   ".$val["body"];
    echo "</p>
    </div>
</div>
</div>
<div class='col-md-2'>
<p>   Location:   <br>".$val["loca"];
echo "</p></div></div></div></div>
";
}
?>
```

`join_block.php`

First, check the block that the user belongs to in the User table. If the user already belongs to a block, reject the application of the user to join the block and return the prompt message to the user 'If you want to join a new block, you must first leave the existing block'.

If the user doesn't belong to any block, then the block list will be queried from the database and show to the user. The user can choose to send an application to one block of the list. After the user clicks, if the number of the members of the block is 0, the user directly become a member of the block. Otherwise, all the existing members in the block will receive the application of the user. And the process about this application will be recorded in the block_request_list. We can follow up the process using this table afterwards. The corresponding code is as follows:

```
if(!session_start())
{
    echo "you should login first";
    return;
}
$uid = $_SESSION["uid"];
$bid = get_block_id($uid);

if($bid>0)
{
    echo "you should leave block first! then you can join other block";
    return;
}else{
    $block_list_id = get_block_list();
}
```

If the user joins the block successfully, the user will also automatically join the hood that the block belongs to.

`add_friend.php`

This page first checks whether the user session is logged in and obtains the user id from the user session.

After getting the user id, first query the hood where the user is and get the ids of all members in the hood. Then get all the user ids that have been sent but are not determined and the user's friend ids. Delete these user ids from all the hood member ids, and the remaining user ids are the appropriate ids that the current logged-in user can add friends to. Corresponding codes are as follows:

```
$friend = get_all_friends($uid);
$friend_request = get_all_friends_request($uid);

$bid = get_hood_id($uid);
if($bid ==0)
{
    exit();
}

$block_memb = get_hood_mem($bid);

//extra self and friend
$final_re =array();
foreach ($block_memb as $key => $value) {

    if(in_array($value,$friend)){
        //friend
    }else{
        if(in_array($value,$friend_request))
        {
             //send request already and not reply
        }else{
            if($value == $uid)
            {
                //slef
            }else{
                //push into list
                array_push($final_re,$value);
            }
        }
    }
}
```

Use PHP to dynamically generate HTML to output all appropriate friend information for the current user to choose from.

`add_neighbor.php`

The page first checks whether the user session is logged in and obtains the user id from the user session.

It then gets the block id of the current logged-in user and query all the members in that block.

Find all the neighbor ids that the user has added in the `Relation_list` and remove them from the previous set of block member ids, leaving all the appropriate members of the neighbor that the user can add.

When the user clicks to add a member, the backend of the site adds a corresponding neighbor record to `Relation_list`, which completes the addition of neighbor.

`map_view.php`

This page first checks whether the user session is logged in and obtains the user id from the user session.

After obtaining the block id corresponding to the user, all street ids included in the block are found in the Block_and_Street data table.

All user information and coordinate information of street belonging to the corresponding street are obtained according to street id. Use a nested loop to iterate through all members location information that can be added to the same block. Call the Google map API to initialize the map, and then use marker to mark the locations of all members.

The core code is as follows:

```
<?php

 foreach($street_latlon as $sid=>$loc)
 {
   $lat = $loc["lat"];
   $lon = $loc["lon"];

    foreach($street_mem as $key=>$arr)
    {
      foreach($arr as $uid=>$name){
       //每人输出一个坐标点
       echo "lat".$uid ;
       echo "=".strval($lat).";";
```

```
        echo "lng".$uid;
        echo "=".strval($lon).";";

        echo " var myLatLng".$uid;
        echo "  = new google.maps.LatLng(."lat".$uid.","."lng".$uid.");";

        echo "var marker".$uid;
        echo "= new google.maps.Marker({
         position:";
         echo "myLatLng".$uid.",";
         echo "title: '".$name."'";
         echo "  });";
         echo "marker".$uid.".setMap(map);";
         $lat = $lat+0.001;

        }
    }
 }
?>
```

By dynamically generating Javascript code in PHP, we can dynamically draw the member location information from mysql and display it on the map.

`edit_profile.php`

When the user chooses to update profile, the profile is first queried.

If the User does not fill in the profile, the user-edited information is inserted directly into the profile table and the generated pid is updated into the User table.

If the User has already edited the profile, the old profile id is queried and the newly written information is inserted into the profile table, while the generated pid is updated into the User table. The outdated profile information is then deleted based on the old pid, which ensures that if the database processing fails, the original profile information will not be directly deleted, provided that the new profile has been stored and updated. The core code of this part is as follows:

```
$re = alter_profile($uid,$self,$family,$destination);   // insert new profile

if($re==0)
{

}else{
    if(pid == -1)  //no need to delete
    {
        echo "profile alter success! <a href = 'main.php'>return to main
page</a>";
    }else{
```

```
        delete_old_profile($pid);
        echo "profile alter success! <a href = 'main.php'>return to main
page</a>";
    }

}
```

Here are some security measures to keep the website safe:

- User password hash storage

  The user password is hashed and stored, effectively preventing the risk of user password leakage.

- SQL preprocessing prevents SQL injection

  PHP mysqli is used to preprocess PHP statements to prevent attacks such as SQL injection. SQL injection is a common attack method in web applications, and the use of preprocessing can effectively prevent the risk of SQL injection.

- Use session to verify user identity and keep user login to prevent user information leakage, fake access and so on.