程序说明

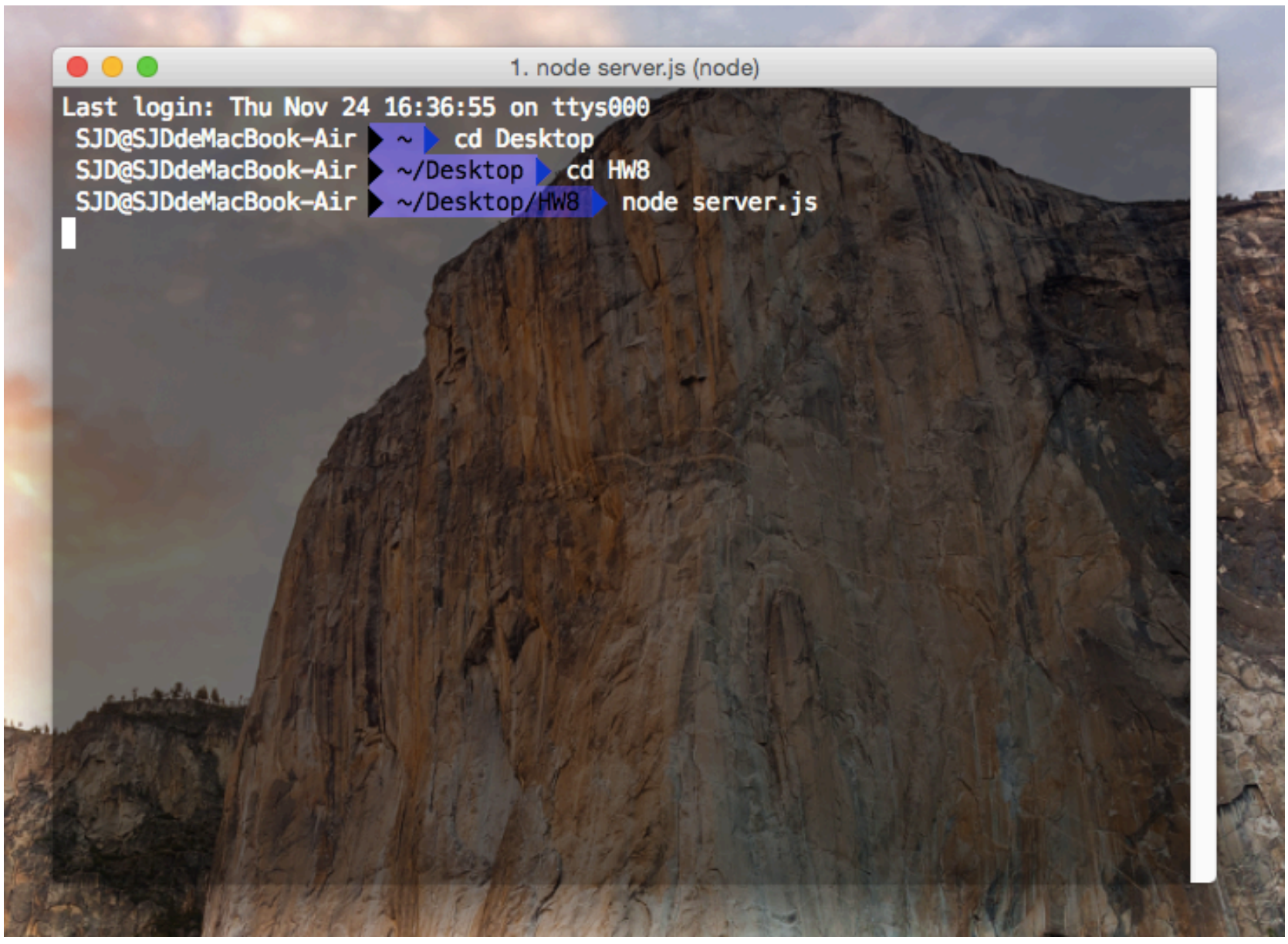
13331233 孙中阳

1. 设计说明

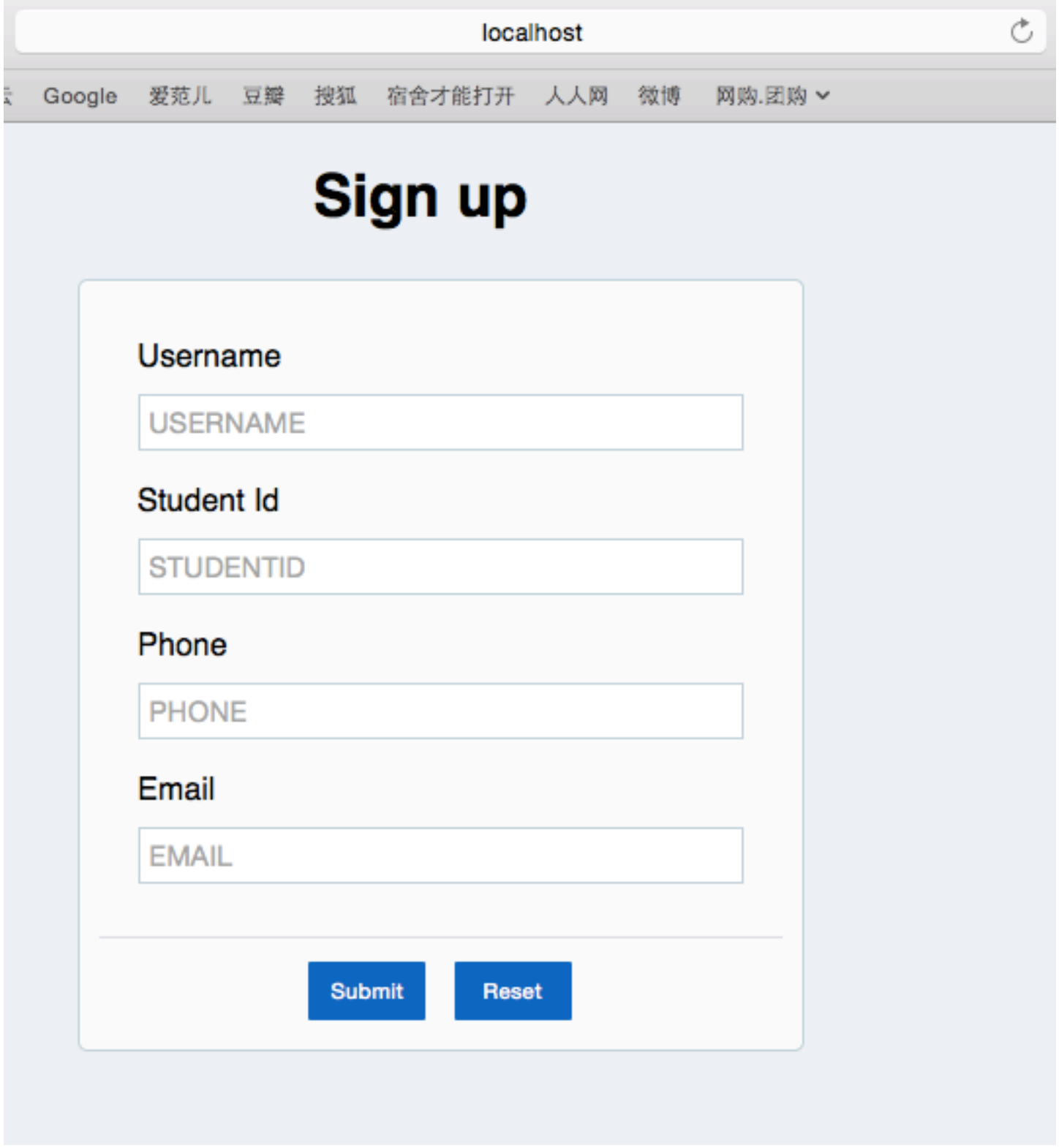
1.1 程序功能

一个简单的Signin 注册系统

可用node signin.js 启动程序



然后通过浏览器 `http://localhost:8000/` 访问 注册界面



能够显示合法的输入规则

Sign up

Username

USERNAME

用户名6~18位英文字母、数字或下划线，以英文字母开头

Student Id

STUDENTID

Phone

PHONE

Email

EMAIL

Submit Reset

自动检测输入是否合法

Sign up

Username

sunzhongyang

Student Id

1333123

学号貌似不是很对，检查一下? (长度不为8)

Phone

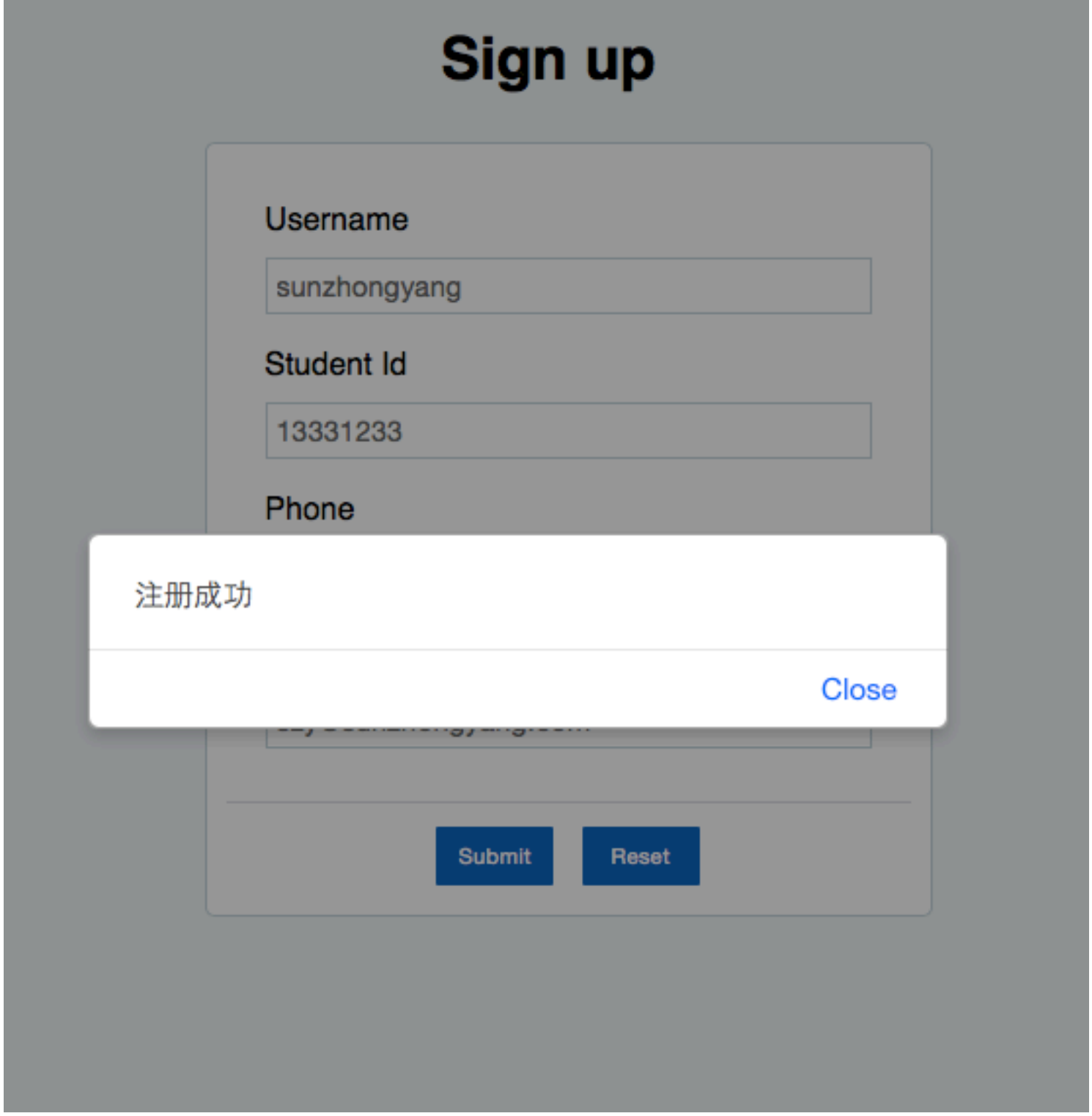
PHONE

Email

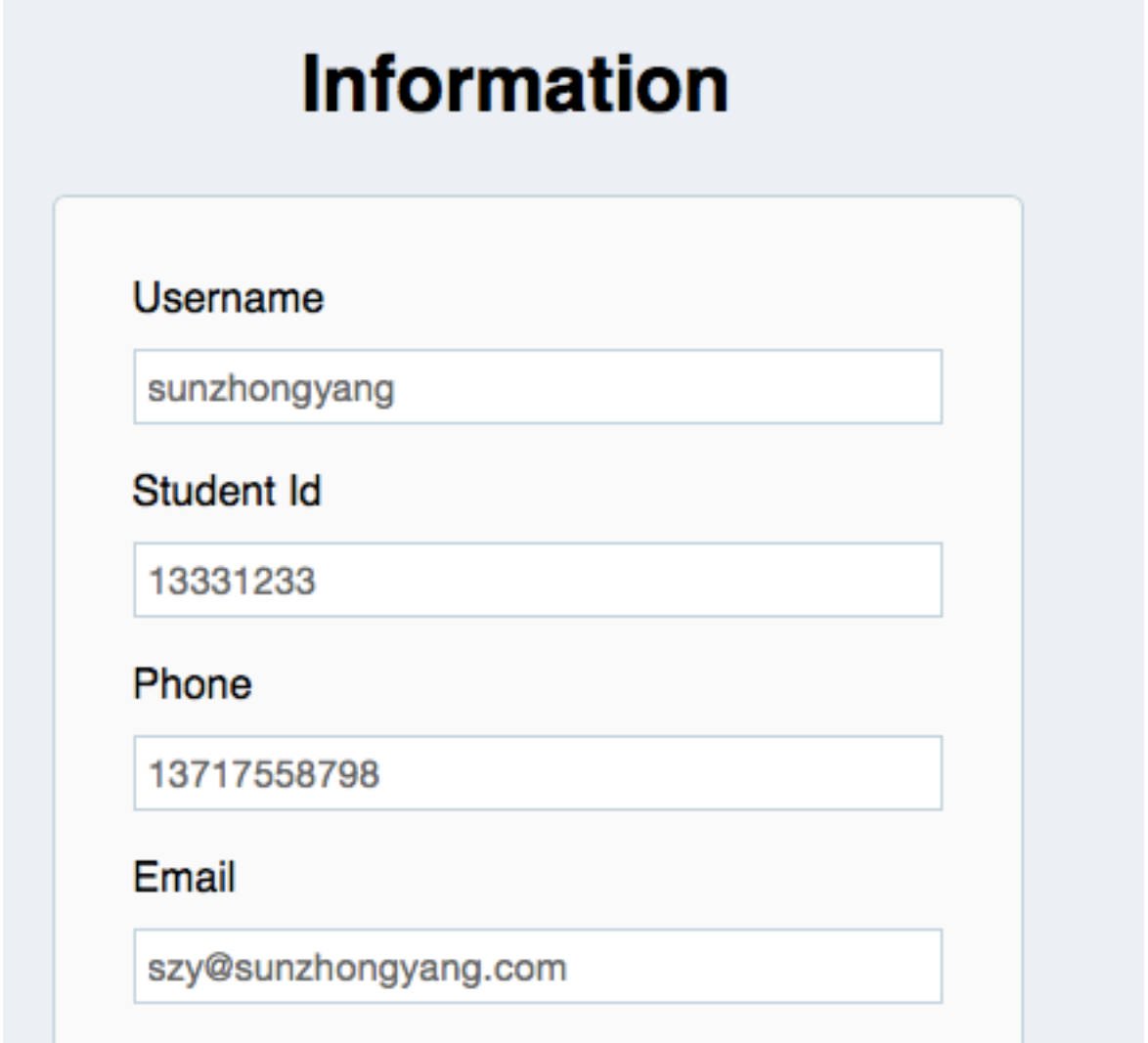
EMAIL

Submit Reset

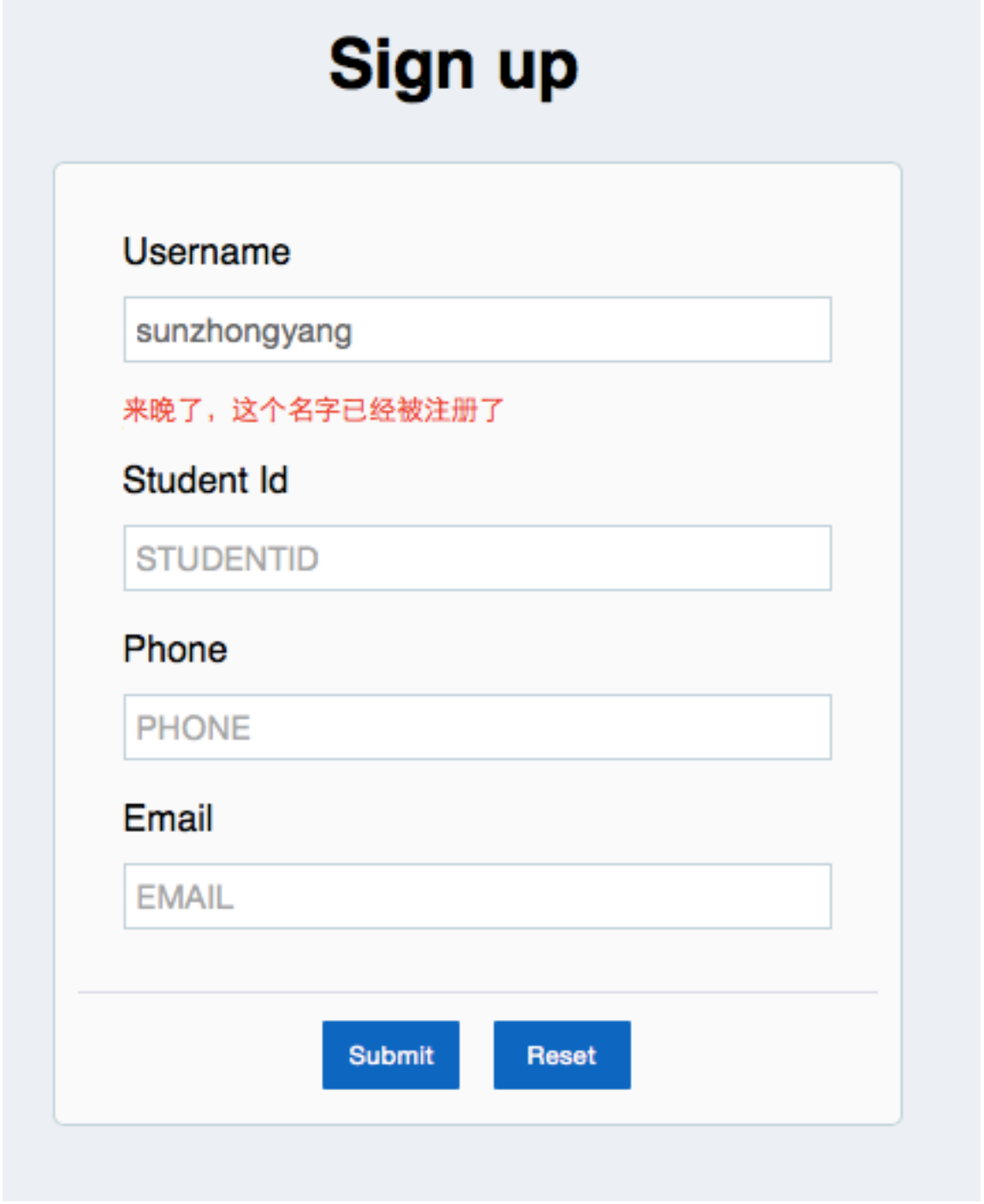
提示注册成功



注册成功后跳转到联系人详情页面



再次注册会自动检测服务器是否有重复输入



点击 `RESET` 恢复初始状态

Sign up

Username

USERNAME

Student Id

STUDENTID

Phone

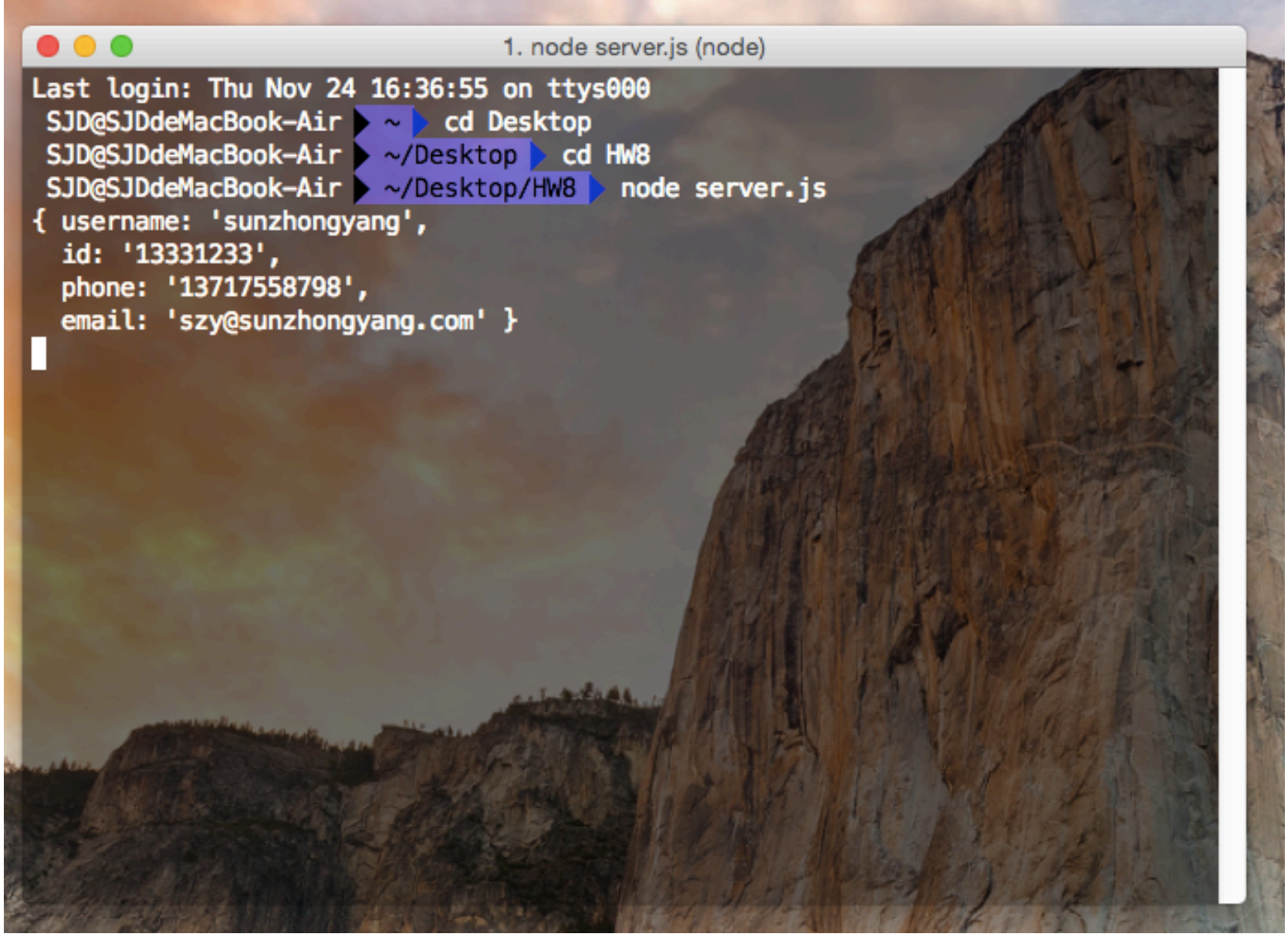
PHONE

Email

EMAIL

Submit Reset

服务端日志



1.2 设计思路

考虑到设计客户端服务端的交互，实验可以分为两个主要的模块，客户端和服务端，两个模块有一定的交互规则，其余部分互相独立

计划实现的目标中，客户端的主要功能是渲染页面，输入提示（合法格式，是否符合）及页面提交。服务端的主要功能是发送客户端内容给用户，保存联系人信息和输入是否合法的查询（是否有重复）

2. 程序结构

2.1 程序文件

客户端

sign_up.html
注册页面 html，主体部分为一个 form，规定了表格的提交方式等

css/sign_up.css
注册页面 css，控制页面布局

js/sign_up.js
注册页面 javascript，三个功能，根据光标设置提示信息，向服务端提交联系人信息，向服务器查询输入是否合法（重复） js/jquery.js

JQuery部分在本程序中主要的功能是向服务端发起HTTP连接并发送有关输入

服务端

server.js
服务端 node.js，提供返回前端文件，接收和保存联系人信息及查询输入是否合法的服务

2.1 主要函数

sign_up.js

首先每个输入框（input）都关联一个 `onfocus()` 函数

```
//鼠标移到用户名输入框
name_input.onfocus = function()
{
set_msg(name_message, "rgb(0,162,74)", "用户名6~18位英文字母、数字或下划线，以英文字母开头");
$("#name_msg").slideDown();
}
```

用于在光标移上时提示合法的输入

同时也有一个光标移开时提供输入合法性校验的函数，这个函数会根据输入是否符合格式，及是否和服务端已有信息相重复作出对应的提示

```
//鼠标离开用户名输入框
name_input.onblur = function()
{
    if(valid_name(name_input.value))
    {
        if(unique("name", name_input.value))
        {
            set_msg(name_message, "rgb(0,162,74)", "这个名字吼哇~");
            setTimeout("$('#name_msg').slideUp('slow');", 500);
            name_status = true;
        }
        else
        {
            set_msg(name_message, "red", "来晚了，这个名字已经被注册了");
            name_status = false;
        }
    }
    else
    {
        var detail = "非法字符";
        if(/^[^(a-zA-Z)]/.test(name_input.value))
        {
            detail = "没有以英文字母开头";
        }
        else if(name_input.value.length < 6 || name_input.value.length > 16)
        {
            detail = "长度不合适";
        }

        set_msg(name_message, "red", "同学你好，用户名有bug " + "(" + detail +
")");
        name_status = false;
    }
}
```

一些正则，判断格式

```
//判断用户名是否合法
function valid_name(name)
{
    return /^[a-zA-Z][a-zA-Z_0-9]{5,18}$/.test(name);
}
```

一个向服务端发起HTTP连接以查询是否重复的函数

```
//向服务器查询某个输入是否唯一
function unique(key, content)
{
    var result = "";

    $.post("http://localhost:8000/search",
    {
        k:key,
        c:content
    },
    function(data, status)
    {
        result = data;
    });

    if(result == "valid")
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

server.js

主体是一个建立HTTP服务器的函数

```
//创建服务器，并监听8000号端口
var http_server = http.createServer(function(req, res)
{
    if(req.url == '/')
    {
        res.writeHead(200, {"Content-type": "text/html"});
        res.end(fs.readFileSync('sign_up.html'));
    }
    else if(req.url == "/css/sign_up.css")
    {
        res.writeHead(200, {"Content-type": "text/css"});
        res.end(fs.readFileSync("css/sign_up.css"));
    }
    else if(req.url == "/css/information.css")
    {
        res.writeHead(200, {"Content-type": "text/css"});
        res.end(fs.readFileSync("css/information.css"));
    }
    else if(req.url == "/js/jquery.js")
    {
        res.writeHead(200, {"Content-type": "text/javascript"});
```

```
    res.end(fs.readFileSync("js/jquery.js"));
}
else if(req.url == "/js/sign_up.js")
{
    res.writeHead(200, {"Content-type": "text/javascript"});
    res.end(fs.readFileSync("js/sign_up.js"));
}
else if(req.url == "/search")
{
    var post = "";

    req.on('data', function(chunk)
    {
        post += chunk;
    });

    req.on('end', function()
    {
        post = querystring.parse(post);

        check(post["k"], post["c"], res);
    });
}
else if(req.url == "/submit")
{
    var post = "";

    req.on('data', function(chunk)
    {
        post += chunk;
    });

    req.on('end', function()
    {
        post = querystring.parse(post);
        console.log(post);
        var obj = {};
        obj["name"] = post["username"];
        obj["studentid"] = post["id"];
        obj["phone"] = post["phone"];
        obj["email"] = post["email"];

        user_array.push(obj);

        show_information(obj, res);
    });
}
else
{
    var username = querystring.parse(url.parse(req.url).query).username;
    for(var i = 0; i < user_array.length; i++)
```

```
        {
            var obj = user_array[i];
            if(obj["name"] == username)
            {
                show_information(obj, res);
            }
        }

        res.writeHead(200, {"Content-type": "text/html"});
        res.end(fs.readFileSync('sign_up.html'));
    }

}).listen(8000);
```

看起来很长，不符合要求，但是程序要考虑其适用范围。大规模的程序，设置一个 case 去判断每个 url 并为每个部分单独编写处理函数是合适的。如此小的程序，就像一只麻雀，执行有限而简单的功能，为每个情况单独开一个函数反而会降低可读性，提升编写难度，时间和出问题的概率。

把所有功能清清楚楚的写出来，写的读的改的都方便，小的原始的生物如果有特别复杂的内部结构，会很脆弱的。当有编写更复杂项目的需求的情况下，自然会使用到新的开发方法，忽略具体情况搞什么代码不超过10行，那我之前两个超过400行的游戏就会出现超过40个函数，这是难以想象的，难道 windows 有数百万个函数？

因为这些原因扣分，抓不住主要矛盾，偏离了学习的重点和本质。