Final Report: Automated CS Account Form Registration

Sainath Gopinath
Team Lead

David Donahue

Ajith Chowdary Attanti

Abstract

Today, students wishing to take computer-science related courses must manually fill out a paper form from the computer science IT department in order to create their account - this is not efficient or secure, and requires unnecessary paper. We propose to create an automated system for registration of UMass Lowell CS accounts. The proposed system will allow students to fill out their account registration form electronically via a webserver coupled with a MongoDB database for long-term storage of created accounts. We propose to add encryption to all exchanges between user, server and database to secure documents and prevent malicious man-in-the-middle attacks and intrusion.

1 Introduction

Every day in the UMass Lowell CS department, students walk into Dandenau 407 to create their own CS account. Staff hand them a paper form which they must fill out by hand. They cannot fill out the form remotely in case of absence. Finally, the form must be stored in a physical location where it could be accessed malicious students. We propose an automated system allowing students to fill out the form electronically and in a secure manner. Our project satisfies both network and security components while potentially providing benefit to students across the department.

2 Justification of Project Requirements

The requirements of the course are :

1. The project must use the materials covered in the course.
2. The project must have substantial coding and networking components.

We can satisfy the requirement 1 by using the concepts of AES Encryption for encrypting the data and we use AES decryption to demonstrate the correctness. Similarly we use RSA for key exchanges. We have a key ring to manage the users keys. We have an large amount of coding components to define the web page and also for the AES Encryption and decryption, similarly for the RSA. While requirement 2 networking will be of the packets transfer from client and CS Server and similarly from CS server to the MongoDB Database.

3 System Description

3.1 Networking components

Our network system has 3 components: Therefore, HTTP protocols are followed and GET, PUT and POST requests are implemented

1. User Web Client
2. Computer Science Dept. Web Server
3. MongoDB database

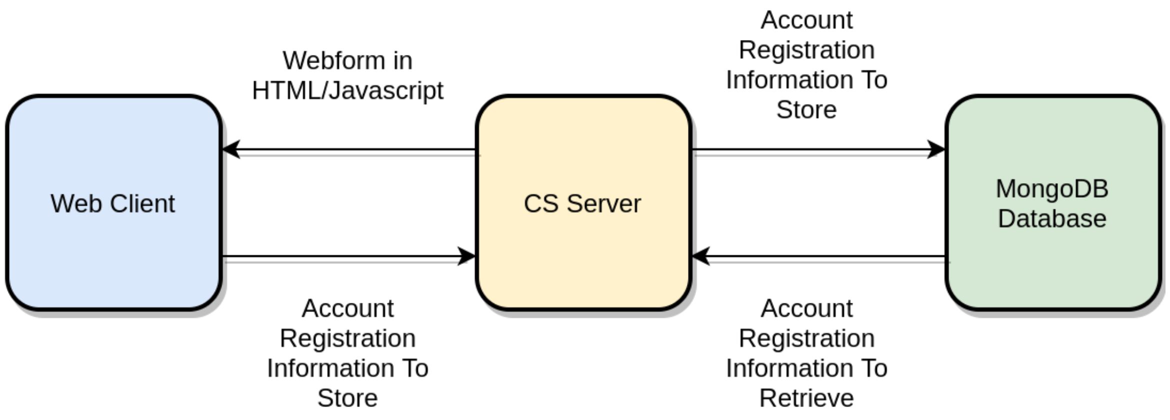

Webform in HTML/Javascript
Account Registration Information To Store
Web Client
CS Server
MongoDB Database
Account Registration Information To Store
Account Registration Information To Retrieve

Figure 1: Description of our system for automated CS user account registration. In this setup, the CS server sends a webform to the web client. The web client encrypts and returns the webform. The CS server then decrypts the webform, and constructs a secret key to encrypt webform data and stores the data in a MongoDB database. We also have added additional security features to the system like prevention of duplicate accounts, honeypot to detect spambots, email confirmation to the user, reCaptcha for anti-spam, validation of form fields. Lastly, the CS server queries database for return of form data and decrypts for viewing and demonstration.

3.2 Overview

In the system we developed, a user connects to the CS server (in the demo code, the server and client exist on the same machine) which returns a Javascript/HTML webpage containing a webform. The user then opens the web-form on the client browser and fills out the form with account information such as name, email and other sensitive information needed to successfully open a CS account. Before the client simply sends this form-data back to the server, it needs to encrypt it for greater security. For encryption, the client and server both agree on a common algorithm, mode of operation and exchange protocol. For example, the key exchange protocol established is RSA and data-encryption is achieved with AES algorithm and CBC mode. On top of that, the system includes additional security features like prevention of duplicate accounts, honeypot to detect spam-bots, email confirmation to the user, reCaptcha for anti-spam, validation of form fields.

Client: $C1 = AES\, E_{key1}(M)$
$CK1 = RSA\, E(K^u(Key1))$ sends to Server

Server: generates and publishes $RSA\, K^u$
$Key1 = RSA\, D(K^r(CK1))$
saves RSA keys to key-ring
$M = AES\, D_{key1}(C1)$
$C2 = AES\, E_{key2}(M)$
saves key2 to key-ring and sends C2 to Database

Demo:
Frontend - retrieves encrypted Database data and displays on HTML page
Backend - retrieves Key2 from key-ring and encrypted data from Database
$M = AES\, D_{key2}(C2)$
prints M into a file and console

3.3 Client-Server Interaction and Security

To enable greater security in our application, the client issues a GET request to retrieve the RSA public key of the server, which is published by server. The client then generates a secret key for AES encrypt of webform data, which itself will be encrypted with the retrieved RSA public key. This secret key (Key1) is generated through randomly generated bits with BBS. Finally, the client encrypts all webform data using AES and the secret key (Key1) as well as a random initialization vector (IV1). The client issues a POST request in which all webform data encrypted with AES is sent back to the server. With this data, the client also sends the secret key (Key1) and initialization vector encrypted with the RSA public key.

3.4 Server-Database Backend Security

Upon receiving POST with all data, the server uses its RSA private key to decrypt the AES secret key (Key1) and initialization vector (IV1). One more layer of protection is added here to secure RSA and AES keys, where the keys are stored in a local key-ring, in-case of power failure or server runtime errors. Next, it uses these values to decrypt the webform data fields. At this point, the server has received all required information from the client needed to make an account. The server then wishes to send this information to a database which exists either on the CS server or in the cloud (both options are available, in the demo the database is local). For additional protection, the server discards Key1 generated at the client side and generates its own secret key (Key2) for further encryption of data. So, the server generates its own secret key (Key2) and uses AES-256-CBC algorithm for all database encryption. It encrypts all user CS form data and stores it in the database using a proper ”person” schema.

3.5 Demo and Application Testing

In an end-to-end system such as this with multiple components, it is desired to have multiple ways of testing correct functionality. To produce the demo, the server then queries the database to return the user data, obtains AES Key2 and IV2 from the local key-ring and decrypts using the database AES secret key (Key2), and displays the user information in the console to prove the information was correctly stored. The client webform can also request an updated list of CS account form submissions. This browser based view of submissions should be disabled in a production version, but shows the encrypted data fields for each user as a visualization of correct behavior.

3.6 Anti-Spamming and Email Confirmation

1. While RSA and AES encryption allow us to exchange user webform information between client and server, we add additional security measures. To protect against form spamming, we introduce a honeypot measure. It operates by adding an extra field to the webform which is hidden from the user, however an automated web-filling bot would automatically fill this (honeypot) field out. The client program detects if the field is filled out, and if so rejects the request and in the demo, the user is alerted saying a bot has been identified. This rejection will only occur for auto-filling, and thus adds an extra layer of protection for the server.

2. In addition, we also send an automated email confirmation to the email address of the user who requests the account, to ensure that if the webform submission was falsely created that the user can contact us to delete the account.

3. One more layer of protection is added on the server-side as well. The server sends out a NoSQL query to MongoDB, before creating a new account, in order to check whether the new account email ID matches any of the existing email IDs in the database. If it does match, the form is not submitted to the database. This prevents not only duplicate accounts but also spamming of the system. This is a Searchable Encryption problem where the database or the cloud cannot know search-key (email) information, so everything is done encrypted with AES.

4. We are checking whether the entered email format is valid in the Frontend to prevent spamming of database

5. We have added a feature (form validation) on the client-side to inactivate the Submit button automatically if the mandatory fields in the form are not filled to prevent users from over-loading back-end and database due to malicious behavior. For example, if users fill-in the email to activate submit button and if the email already exists in the database again the form is not sent to database

6. Finally, a Google re-Captcha is added to the form on the client-side to detect Spams on the forms. Google re-Captcha compares the programs loaded with the client page with an internal Spam database and it will block if it suspects any Spams loaded with the page. However, we are not able to Demo this because it works only if the webpage is hosted on a server or cloud. This demo is performed on the localhost.

   All of these measures serve to protect against malicious user behavior toward our system.

4 System Environment

The web form client frontend is implemented using an Angular server with ExpressJS and NodeJS. NodeJS allows for reliable installation of Javascript packages and dependency tracking and automatic building of Javascript applications. Angular builds a typescript web application on top of NodeJS which allows for running dedicated code on the client browser. ExpressJS similarly helps with web-based applications.

The CS server backend is implemented using NodeJS purely in Javascript with a MongoDB database for storage of user data. Encryption between the server and the MongoDB database is handled using the NodeJS crypto library. However, this library is not able to be used easily in the frontend Angular typescript language files, so an alternative library node-forge is used for AES and RSA encryption between the server and the client Angular web form.

MongoDB, ExpressJS, Angular and NodeJS make up a M.E.A.N. stack application, a full stack for handling client interaction with the server. The code and additional descriptions are available on GitHub [1].

5 Future Work

While the created demo system is fully functional, we outline more ways to enhance it for future work. This system produces all the required information to produce a CS account. To further automate the process of creating a webform account, the server can present to system admins each account creation request sequentially, and ask for approval per account. Upon approval, the system could automatically run bash commands to create a new user using the ”useradd” Linux command. This account could be automatically created using the first character of the first name followed by the last name (David Donahue –¿ ddonahue). If an email of that name exists, a numeric value can be added (ddonahue1).

In future work, we can further increase the security of the webform by requiring the student to login using their university email account (through the university standard authentication portal) before filling out the form. This login will be logged and will further prevent form spamming and impersonation of users. This is in addition to the email notification sent by the current system.

Right now, the email field only checks if it is a valid email format and not whether it is an UML email ID (like @student.uml.edu email). In the future, we should use REGEX to add this capability by pattern matching.

At this point, we are unable to demonstrate the working of Google reCaptcha since we didnt host the webpage on the cloud. In the future, the webpage could be hosted on the cloud and reCaptcha can be demonstrated.

The team plans to integrate with the UML CS server and Linux account creation bash script so the project can be used by students.

[1]Code Repository: `https://github.com/djd1283/CSAccountFormSystem`

6 Member Contributions

6.1 Sainath Gopinath

1. Came up with the project idea that is to automate the CS account registration and to store student confidential information in an encrypted database

2. Researched MEAN stack and updated team with information and methodology. Enabled MEAN stack in Windows environment

3. Developed backend NodeJS server-side coding, ExpressJS framework coding and MongoDB database key-value schema and interaction coding

4. Researched and implemented NodeJS-compatible encryption between frontend browser and backend server using RSA and AES in node-forge

5. Researched and implemented NodeJs-compatible encryption between backend server and database using AES in Crypto

6. Implemented local key-rings on the server to store AES keys and IVs and RSA public and private keys

7. Developed a NodeJs-compatible AES Decrypt program using Crypto that obtains key information from local key-ring for Demo purpose (to show that we are able to both encrypt and decrypt from the database using key-ring)

8. Integrated Nodemailer (sends email confirmation) coding into the backend server

9. Implemented anti-spamming of emails coding in Node-js backend by checking whether the email already exists in the encrypted database (searchable-encryption problem). It also prevents account duplication

10. Contributed to project report documentation and presentation

6.2 David Donahue

1. Developed Angular web form frontend with all fields corresponding to the computer science account creation paper form data in Typescript language

2. Enabled MEAN stack to work in a Linux environment for compatibility with CS Linux server solving packaging issues

3. Researched and implemented Typescript-compatible encryption between frontend browser and backend server using RSA and AES in node-forge, solving compatibility issues in Angular

4. Modified database schema to reflect the additional data fields and modified REST communication between frontend and backend to pass this data along with encryption information and serialization of node-forge PKC objects

5. Implemented AES encryption of additional data fields between backend server and database with checks for empty fields

6. Managed Overleaf latex and Github repository creation and contributed a majority to proposal, final report, and final presentation

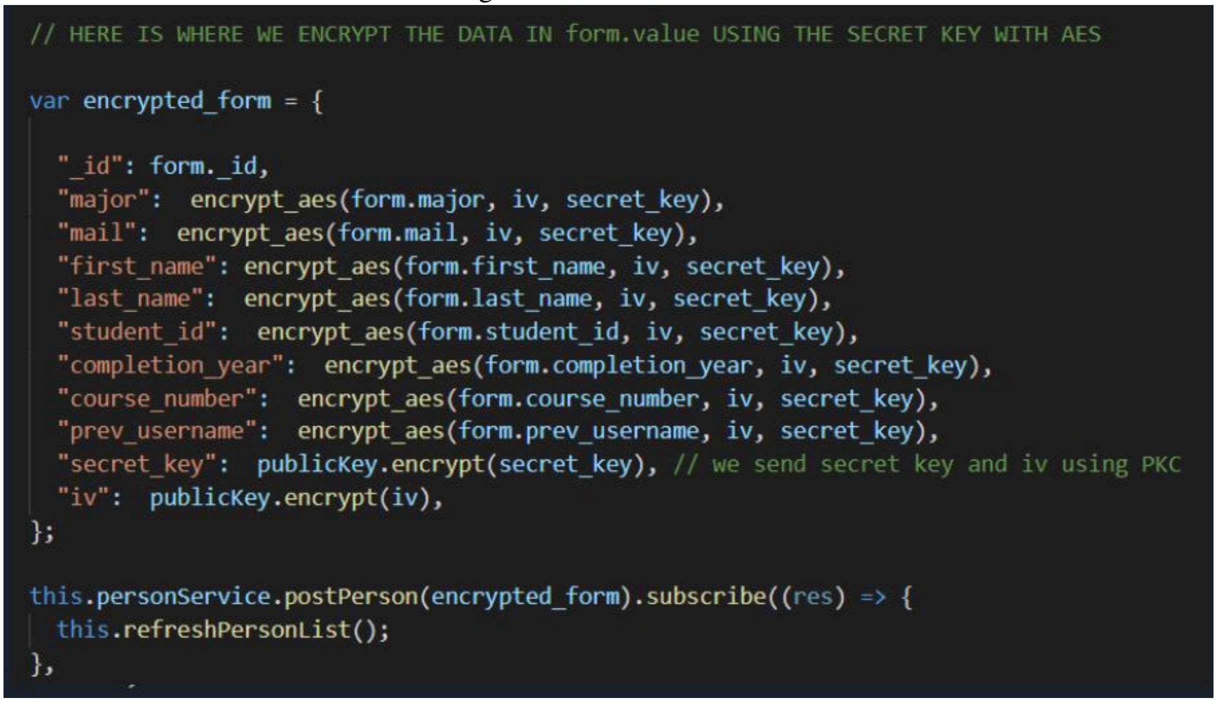
7. Coordinated and presented demo in class

6.3 Ajith Chowdary Attanti

1. Researched for the available type of Network security components for the front end Application
2. Developed and Enabled the Email configuration for the web application for confirmation of submissions of the form to the client
3. Developed the Honeypot Anti-spam feature for the Application to avoid the bots for spamming and auto-filling of the form
4. The two features helps in the Spamming of the form and other bots from the client side
5. Researched and implemented the Google recaptcha to the Application for detection of Bots
6. Implemented other security features of the form such as Disabling the form untill all the fields entered to reduce the traffic on the server and correctness of the data

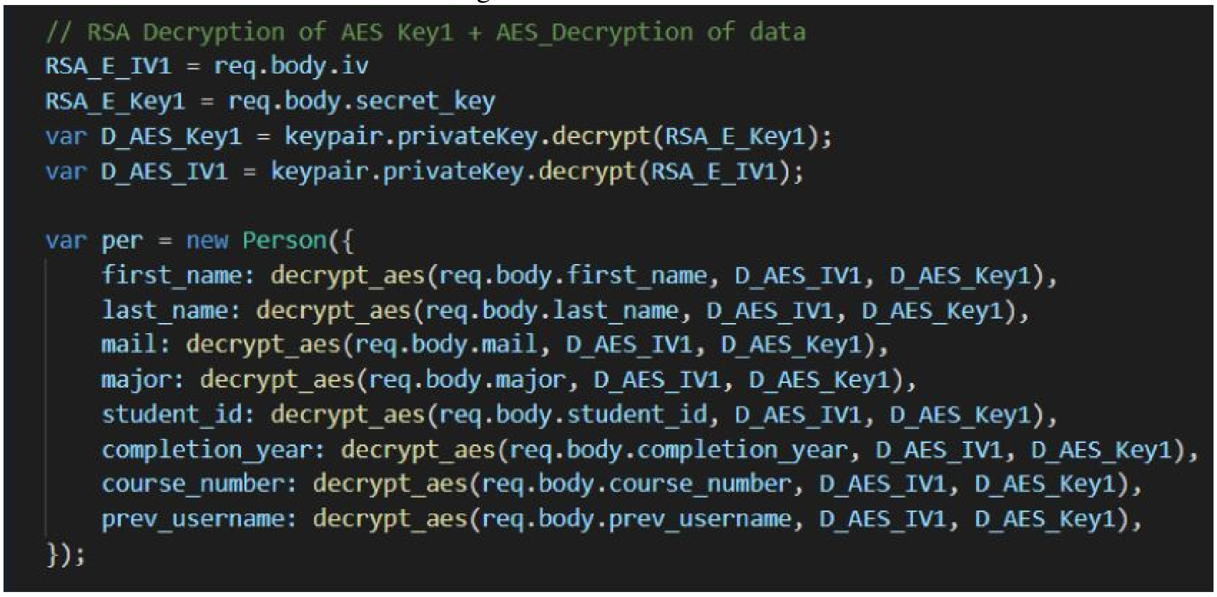
7 Sample code

Figure 2: Client Side code



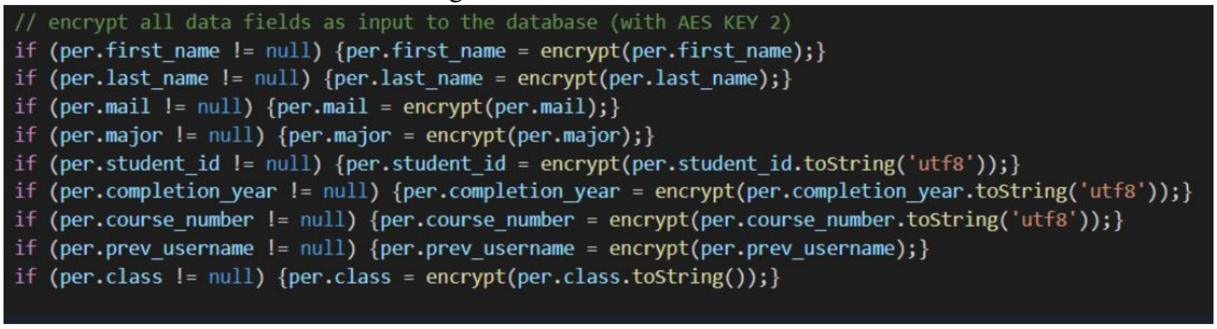
```
// HERE IS WHERE WE ENCRYPT THE DATA IN form.value USING THE SECRET KEY WITH AES

var encrypted_form = {

  "_id": form._id,
  "major":  encrypt_aes(form.major, iv, secret_key),
  "mail":  encrypt_aes(form.mail, iv, secret_key),
  "first_name": encrypt_aes(form.first_name, iv, secret_key),
  "last_name":  encrypt_aes(form.last_name, iv, secret_key),
  "student_id":  encrypt_aes(form.student_id, iv, secret_key),
  "completion_year":  encrypt_aes(form.completion_year, iv, secret_key),
  "course_number":  encrypt_aes(form.course_number, iv, secret_key),
  "prev_username":  encrypt_aes(form.prev_username, iv, secret_key),
  "secret_key":  publicKey.encrypt(secret_key), // we send secret key and iv using PKC
  "iv":  publicKey.encrypt(iv),
};

this.personService.postPerson(encrypted_form).subscribe((res) => {
  this.refreshPersonList();
},
```

Figure 3: Server Side code



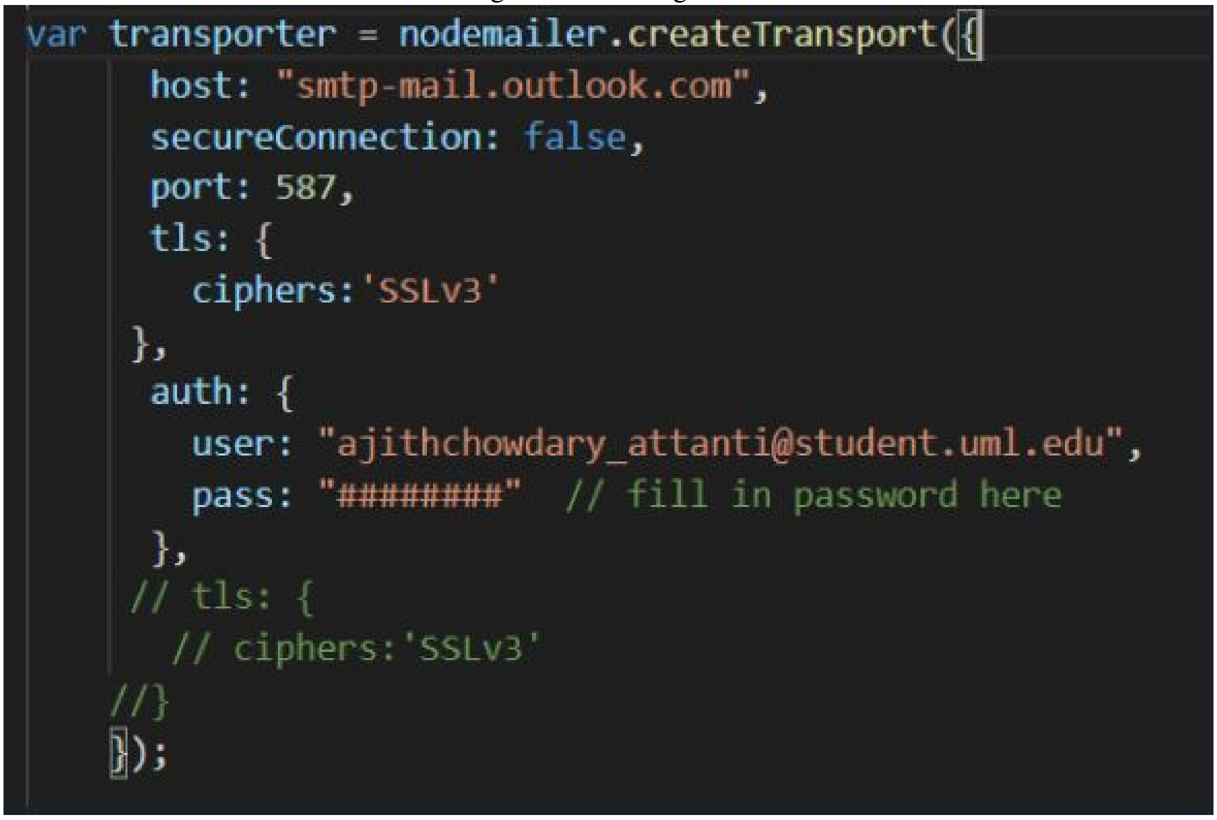
```
// RSA Decryption of AES Key1 + AES_Decryption of data
RSA_E_IV1 = req.body.iv
RSA_E_Key1 = req.body.secret_key
var D_AES_Key1 = keypair.privateKey.decrypt(RSA_E_Key1);
var D_AES_IV1 = keypair.privateKey.decrypt(RSA_E_IV1);

var per = new Person({
    first_name: decrypt_aes(req.body.first_name, D_AES_IV1, D_AES_Key1),
    last_name: decrypt_aes(req.body.last_name, D_AES_IV1, D_AES_Key1),
    mail: decrypt_aes(req.body.mail, D_AES_IV1, D_AES_Key1),
    major: decrypt_aes(req.body.major, D_AES_IV1, D_AES_Key1),
    student_id: decrypt_aes(req.body.student_id, D_AES_IV1, D_AES_Key1),
    completion_year: decrypt_aes(req.body.completion_year, D_AES_IV1, D_AES_Key1),
    course_number: decrypt_aes(req.body.course_number, D_AES_IV1, D_AES_Key1),
    prev_username: decrypt_aes(req.body.prev_username, D_AES_IV1, D_AES_Key1),
});
```

Figure 4: Server Database code



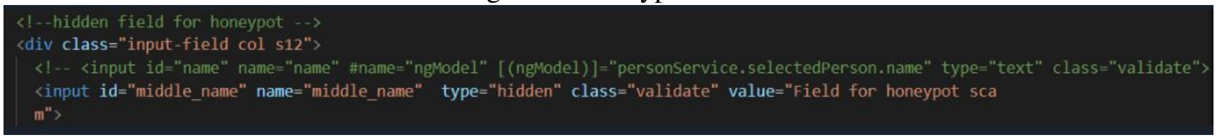
```
// encrypt all data fields as input to the database (with AES KEY 2)
if (per.first_name != null) {per.first_name = encrypt(per.first_name);}
if (per.last_name != null) {per.last_name = encrypt(per.last_name);}
if (per.mail != null) {per.mail = encrypt(per.mail);}
if (per.major != null) {per.major = encrypt(per.major);}
if (per.student_id != null) {per.student_id = encrypt(per.student_id.toString('utf8'));}
if (per.completion_year != null) {per.completion_year = encrypt(per.completion_year.toString('utf8'));}
if (per.course_number != null) {per.course_number = encrypt(per.course_number.toString('utf8'));}
if (per.prev_username != null) {per.prev_username = encrypt(per.prev_username);}
if (per.class != null) {per.class = encrypt(per.class.toString());}
```

Figure 5: Emailing code



```
var transporter = nodemailer.createTransport({
      host: "smtp-mail.outlook.com",
      secureConnection: false,
      port: 587,
      tls: {
        ciphers:'SSLv3'
     },
      auth: {
        user: "ajithchowdary_attanti@student.uml.edu",
        pass: "########"  // fill in password here
      },
     // tls: {
       // ciphers:'SSLv3'
    //}
    });
```

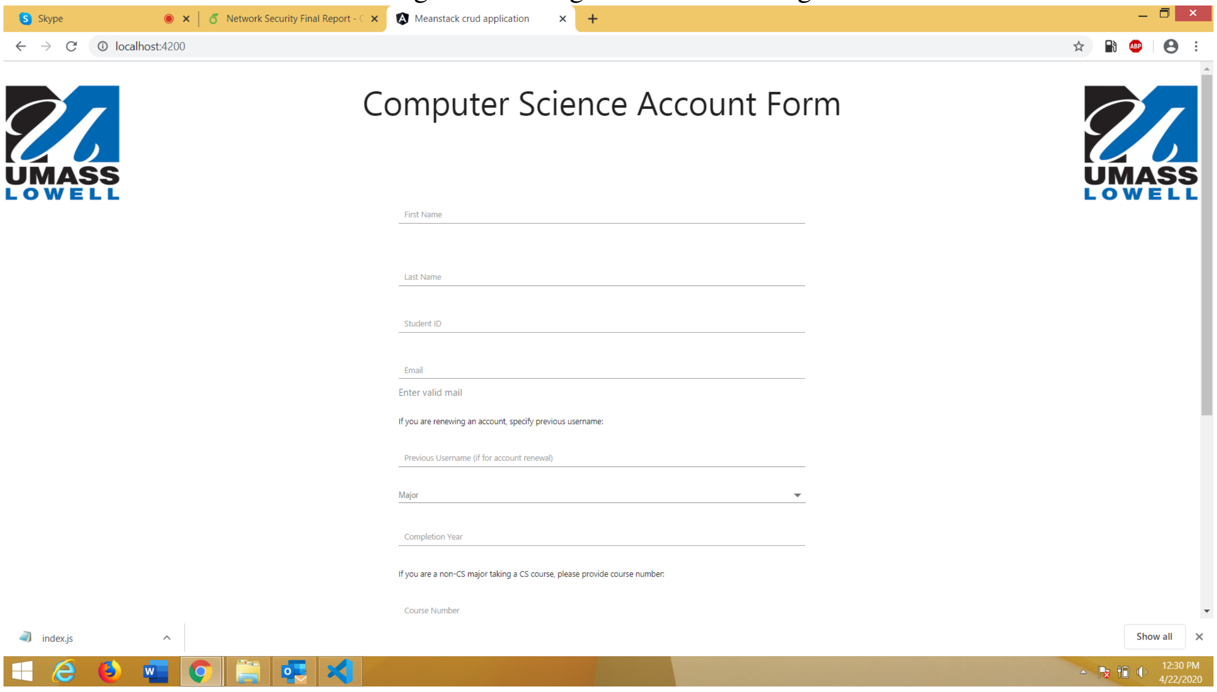
Figure 6: Honeypot code



```
<!--hidden field for honeypot -->
<div class="input-field col s12">
  <!-- <input id="name" name="name" #name="ngModel" [(ngModel)]="personService.selectedPerson.name" type="text" class="validate">
  <input id="middle_name" name="middle_name"  type="hidden" class="validate" value="Field for honeypot sca
  m">
```

8 Demonstration

Figure 7: CS registration form design


Computer Science Account Form

First Name

Last Name

Student ID

Email

Enter valid mail

If you are renewing an account, specify previous username:

Previous Username (if for account renewal)

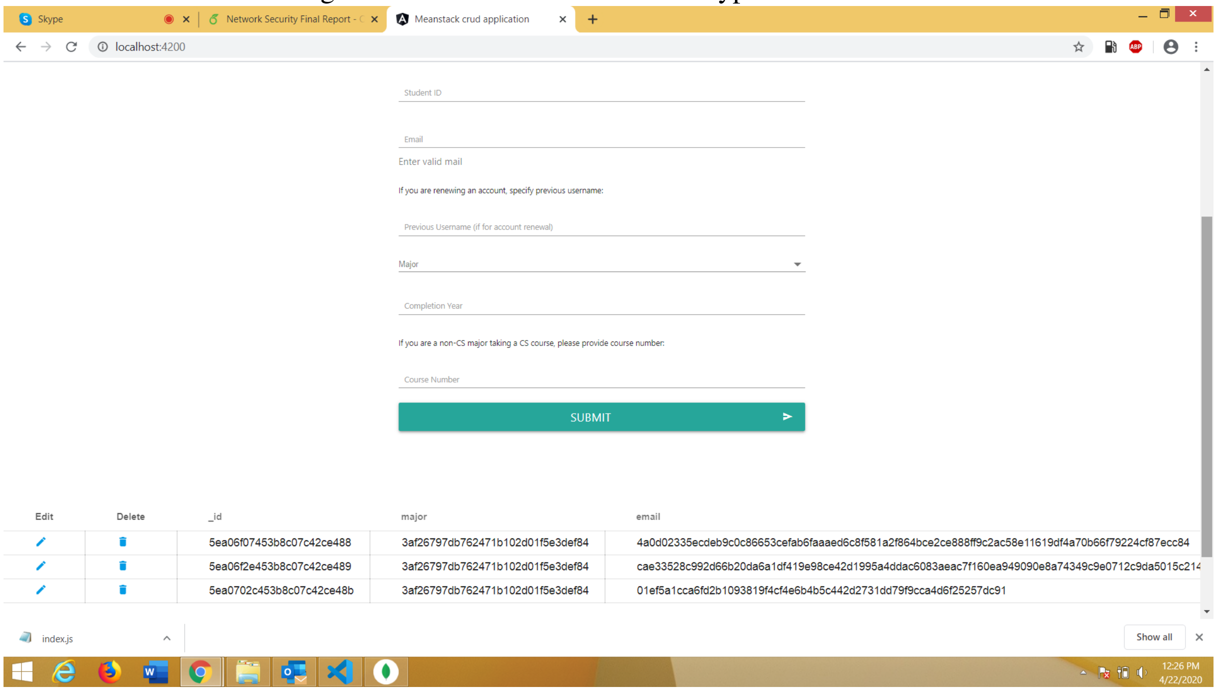
Major

Completion Year

If you are a non-CS major taking a CS course, please provide course number:

Course Number

Figure 8: Insert data and stored encrypted in database


Student ID

Email

Enter valid mail

If you are renewing an account, specify previous username:

Previous Username (if for account renewal)

Major

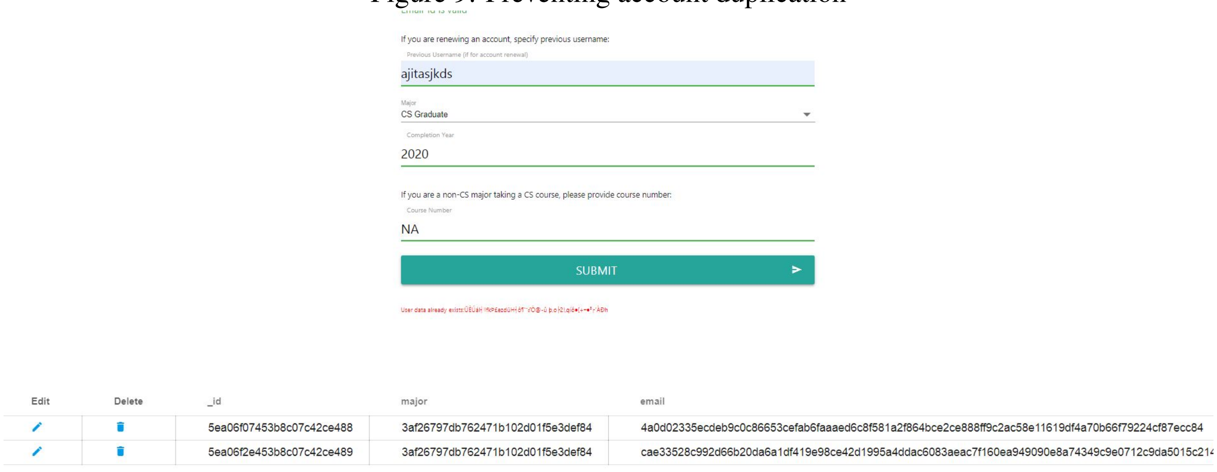
Completion Year

If you are a non-CS major taking a CS course, please provide course number:

Course Number

Edit	Delete	_id	major	email
✎	🗑	5ea06f07453b8c07c42ce488	3af26797db762471b102d01f5e3def84	4a0d02335ecdeb9c0c86653cefab6faaaed6c8f581a2f864bce2ce888ff9c2ac58e11619df4a70b66f79224cf87ecc84
✎	🗑	5ea06f2e453b8c07c42ce489	3af26797db762471b102d01f5e3def84	cae33528c992d66b20da6a1df419e98ce42d1995a4ddac6083aeac7f160ea949090e8a74349c9e0712c9da5015c214
✎	🗑	5ea0702c453b8c07c42ce48b	3af26797db762471b102d01f5e3def84	01ef5a1cca6fd2b1093819f4cf4e6b4b5c442d2731dd79f9cca4d6f25257dc91

Figure 9: Preventing account duplication


If you are renewing an account, specify previous username:

Previous Username (if for account renewal): ajitasjkds

Major: CS Graduate

Completion Year: 2020

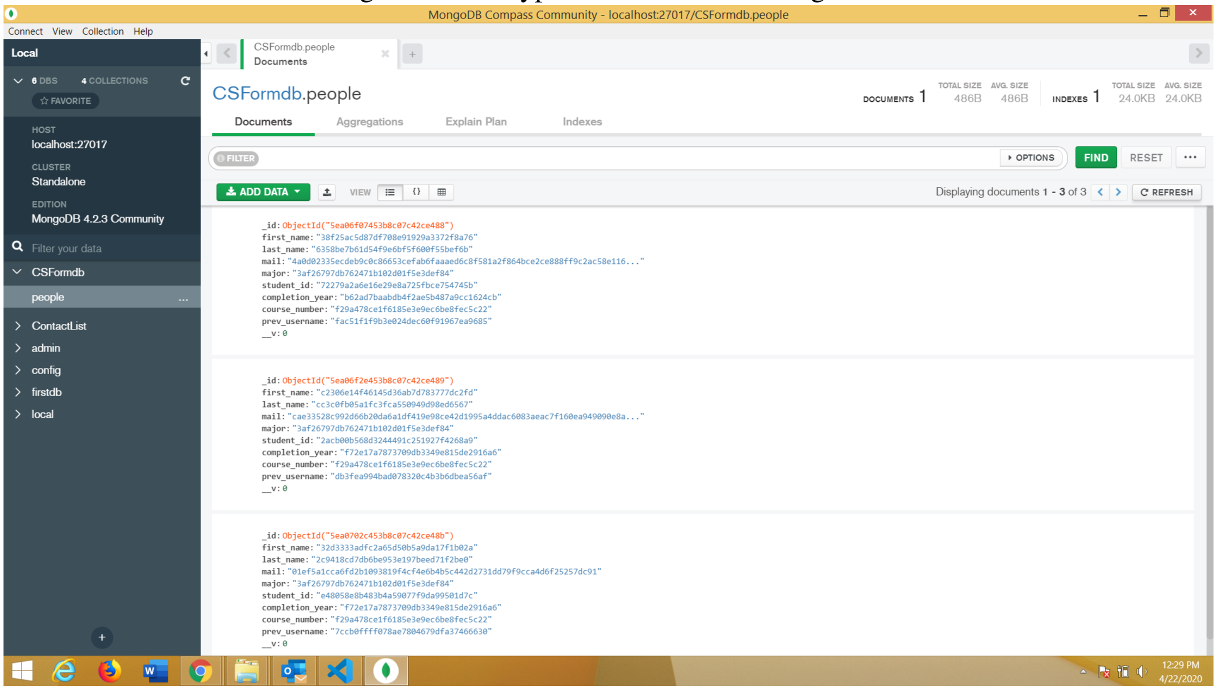
If you are a non-CS major taking a CS course, please provide course number:

Course Number: NA

User data already exists:ÜÈÚáH !!fkP£ezdüH┤ô¶¨√Ò@-û þ.o├2!.qíó•(÷¬•²┌´ÀÐh

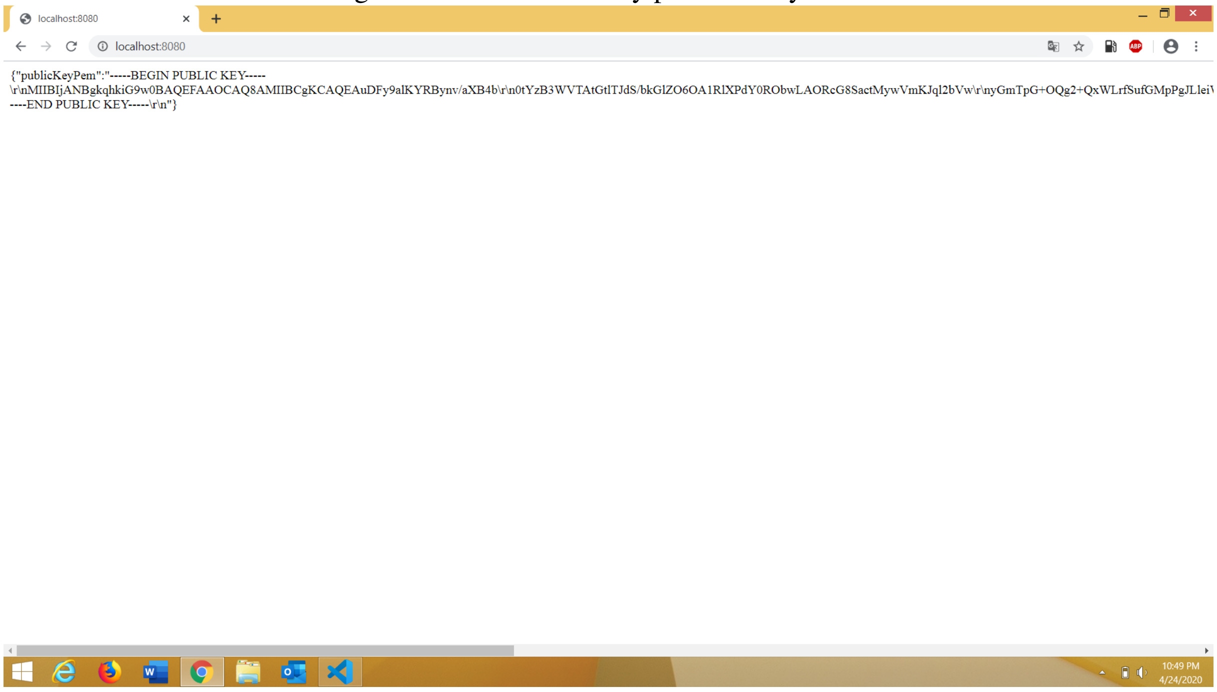
Edit	Delete	_id	major	email
✎	🗑	5ea06f07453b8c07c42ce488	3af26797db762471b102d01f5e3def84	4a0d02335ecdeb9c0c86653cefab6faaaed6c8f581a2f864bce2ce888ff9c2ac58e11619df4a70b66f79224cf87ecc84
✎	🗑	5ea06f2e453b8c07c42ce489	3af26797db762471b102d01f5e3def84	cae33528c992d66b20da6a1df419e98ce42d1995a4ddac6083aeac7f160ea949090e8a74349c9e0712c9da5015c214

Figure 10: Encrypted data stored in MongoDB



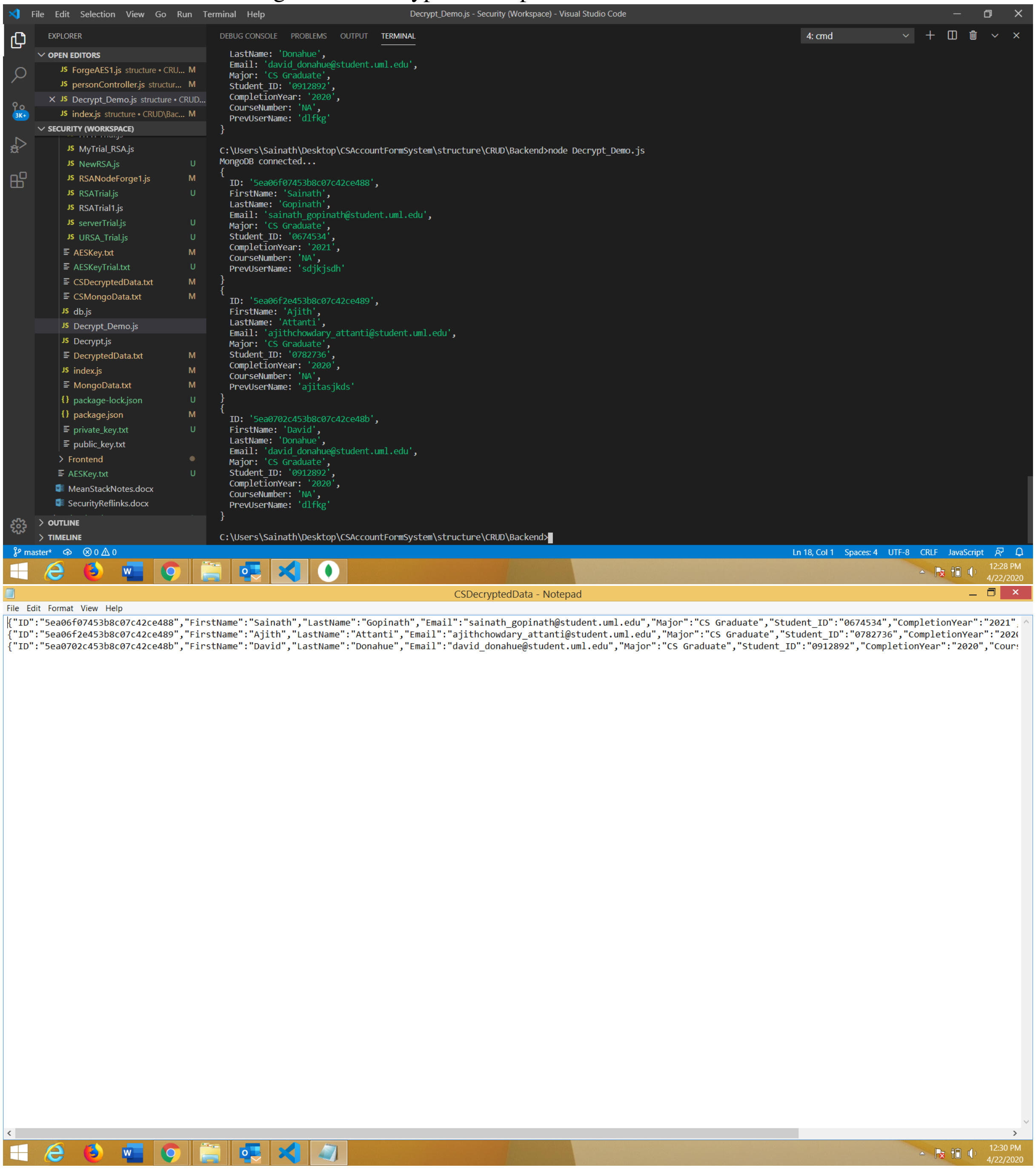
```
_id: ObjectId("5ea06f07453b8c07c42ce488")
first_name: "38f25ac5d87df708e91929a3372f8a76"
last_name: "6358be7b61d54f9e6bf5f600f55bef6b"
mail: "4a0d02335ecdeb9c0c86653cefab6faaaed6c8f581a2f864bce2ce888ff9c2ac58e116..."
major: "3af26797db762471b102d01f5e3def84"
student_id: "72279a2a6e16e29e8a725fbce754745b"
completion_year: "b62ad7baabdb4f2ae5b487a9cc1624cb"
course_number: "f29a478ce1f6185e3e9ec6be8fec5c22"
prev_username: "fac51f1f9b3e024dec60f91967ea9685"
__v: 0

_id: ObjectId("5ea06f2e453b8c07c42ce489")
first_name: "c2306e14f46145d36ab7d783777dc2fd"
last_name: "cc3c0fb05a1fc3fca550949d98ed6567"
mail: "cae33528c992d66b20da6a1df419e98ce42d1995a4ddac6083aeac7f160ea949090e8a..."
major: "3af26797db762471b102d01f5e3def84"
student_id: "2acb00b568d3244491c251927f4268a9"
completion_year: "f72e17a7873709db3349e815de2916a6"
course_number: "f29a478ce1f6185e3e9ec6be8fec5c22"
prev_username: "db3fea994bad078320c4b3b6dbea56af"
__v: 0

_id: ObjectId("5ea0702c453b8c07c42ce48b")
first_name: "32d3333adfc2a65d50b5a9da17f1b02a"
last_name: "2c9418cd7db6be953e197beed71f2be0"
mail: "01ef5a1cca6fd2b1093819f4cf4e6b4b5c442d2731dd79f9cca4d6f25257dc91"
major: "3af26797db762471b102d01f5e3def84"
student_id: "e48058e8b483b4a59077f9da99501d7c"
completion_year: "f72e17a7873709db3349e815de2916a6"
course_number: "f29a478ce1f6185e3e9ec6be8fec5c22"
prev_username: "7ccb0ffff078ae7804679dfa37466630"
__v: 0
```

Figure 11: RSA Public key published by Server



{"publicKeyPem":"-----BEGIN PUBLIC KEY-----
\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAuDFy9alKYRBynv/aXB4b\r\n0tYzB3WVTAtGtlTJdS/bkGlZO6OA1RlXPdY0RObwLAORcG8SactMywVmKJql2bVw\r\nyGmTpG+OQg2+QxWLrfSufGMpPgJLlei
----END PUBLIC KEY-----\r\n"}

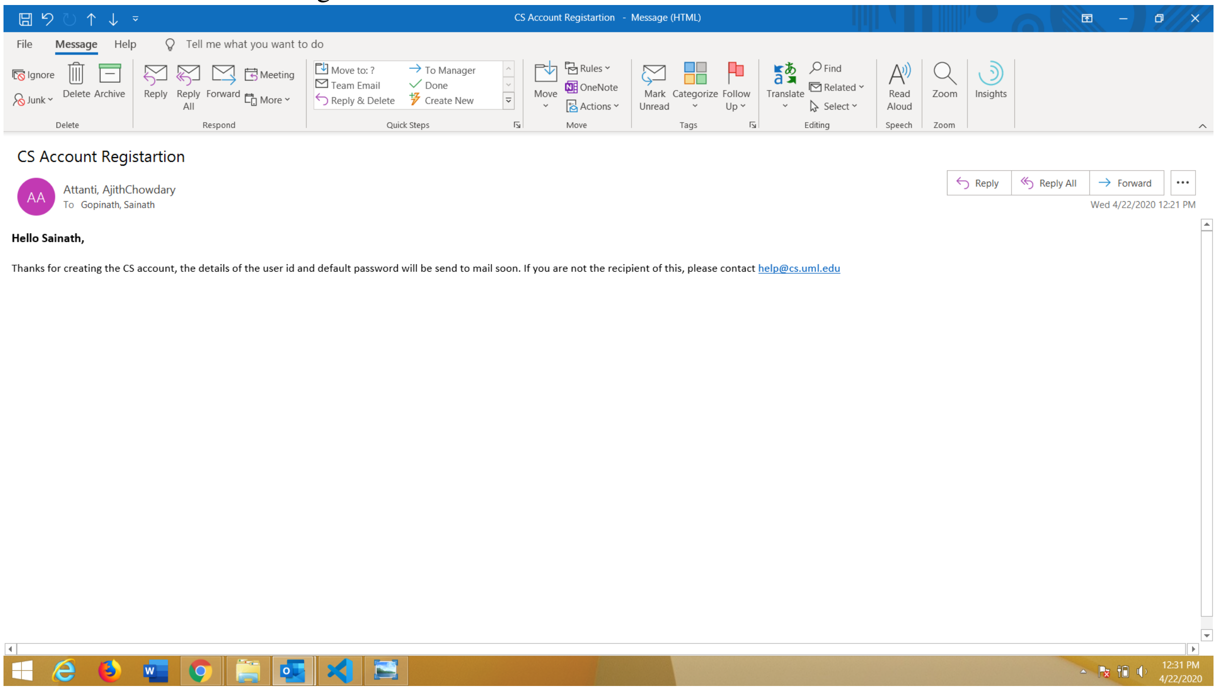
Figure 12: Decrypted data printed on console and file



```
  LastName: 'Donahue',
  Email: 'david_donahue@student.uml.edu',
  Major: 'CS Graduate',
  Student_ID: '0912892',
  CompletionYear: '2020',
  CourseNumber: 'NA',
  PrevUserName: 'dlfkg'
}

C:\Users\Sainath\Desktop\CSAccountFormSystem\structure\CRUD\Backend>node Decrypt_Demo.js
MongoDB connected...
{
  ID: '5ea06f07453b8c07c42ce488',
  FirstName: 'Sainath',
  LastName: 'Gopinath',
  Email: 'sainath_gopinath@student.uml.edu',
  Major: 'CS Graduate',
  Student_ID: '0674534',
  CompletionYear: '2021',
  CourseNumber: 'NA',
  PrevUserName: 'sdjkjsdh'
}
{
  ID: '5ea06f2e453b8c07c42ce489',
  FirstName: 'Ajith',
  LastName: 'Attanti',
  Email: 'ajithchowdary_attanti@student.uml.edu',
  Major: 'CS Graduate',
  Student_ID: '0782736',
  CompletionYear: '2020',
  CourseNumber: 'NA',
  PrevUserName: 'ajitasjkds'
}
{
  ID: '5ea0702c453b8c07c42ce48b',
  FirstName: 'David',
  LastName: 'Donahue',
  Email: 'david_donahue@student.uml.edu',
  Major: 'CS Graduate',
  Student_ID: '0912892',
  CompletionYear: '2020',
  CourseNumber: 'NA',
  PrevUserName: 'dlfkg'
}

C:\Users\Sainath\Desktop\CSAccountFormSystem\structure\CRUD\Backend>
```

```
{"ID":"5ea06f07453b8c07c42ce488","FirstName":"Sainath","LastName":"Gopinath","Email":"sainath_gopinath@student.uml.edu","Major":"CS Graduate","Student_ID":"0674534","CompletionYear":"2021"
{"ID":"5ea06f2e453b8c07c42ce489","FirstName":"Ajith","LastName":"Attanti","Email":"ajithchowdary_attanti@student.uml.edu","Major":"CS Graduate","Student_ID":"0782736","CompletionYear":"202
{"ID":"5ea0702c453b8c07c42ce48b","FirstName":"David","LastName":"Donahue","Email":"david_donahue@student.uml.edu","Major":"CS Graduate","Student_ID":"0912892","CompletionYear":"2020","Cour
```

Figure 13: Email confirmation after account is created



CS Account Registartion - Message (HTML)

CS Account Registartion

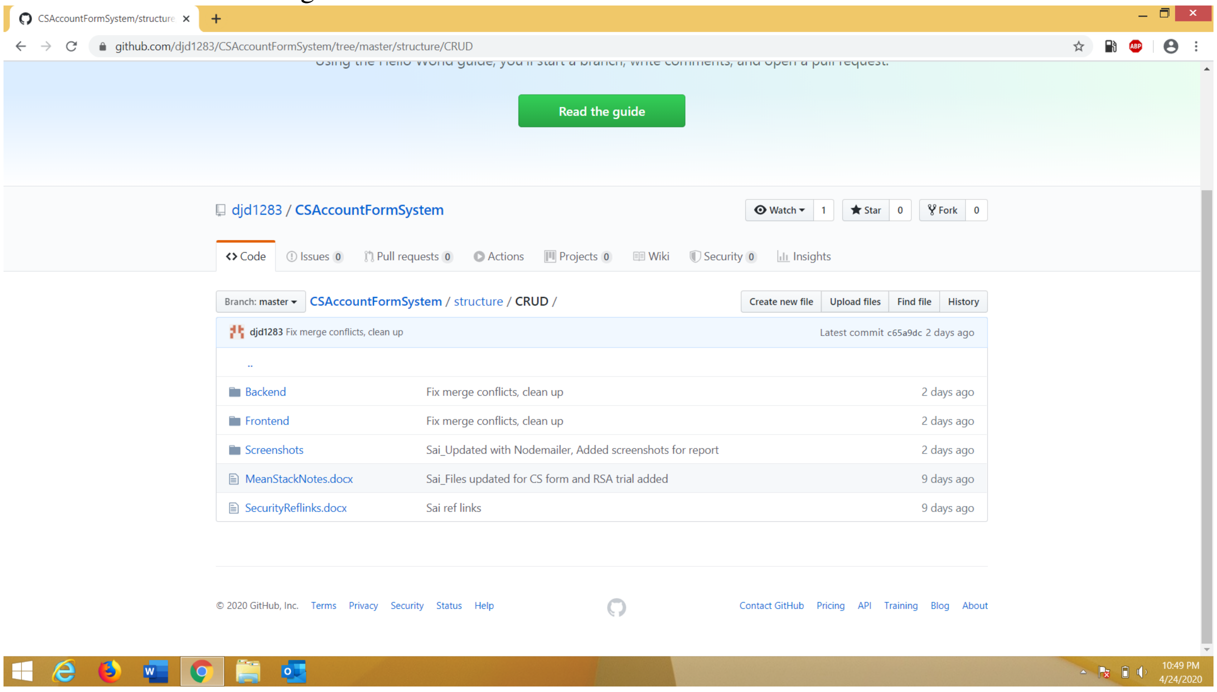
Attanti, AjithChowdary
To Gopinath, Sainath

Wed 4/22/2020 12:21 PM

Hello Sainath,

Thanks for creating the CS account, the details of the user id and default password will be send to mail soon. If you are not the recipient of this, please contact help@cs.uml.edu

Figure 14: Git for team collaboration and version control



github.com/djd1283/CSAccountFormSystem/tree/master/structure/CRUD

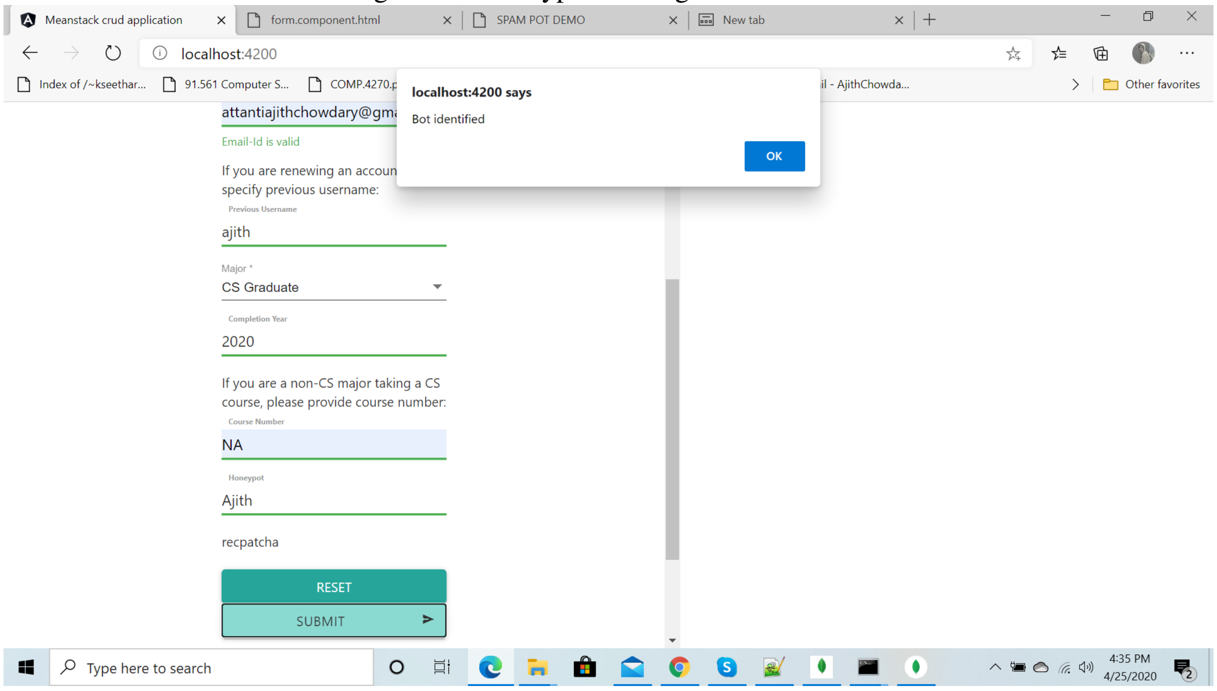
djd1283 / CSAccountFormSystem

Branch: master ▾ CSAccountFormSystem / structure / CRUD /

djd1283 Fix merge conflicts, clean up — Latest commit c65a9dc 2 days ago

..		
Backend	Fix merge conflicts, clean up	2 days ago
Frontend	Fix merge conflicts, clean up	2 days ago
Screenshots	Sai_Updated with Nodemailer, Added screenshots for report	2 days ago
MeanStackNotes.docx	Sai_Files updated for CS form and RSA trial added	9 days ago
SecurityReflinks.docx	Sai ref links	9 days ago

Figure 15: Honeypot alerting user about Bot



localhost:4200 says

Bot identified

attantiajithchowdary@gma

Email-Id is valid

If you are renewing an accoun
specify previous username:

Previous Username: ajith

Major *: CS Graduate

Completion Year: 2020

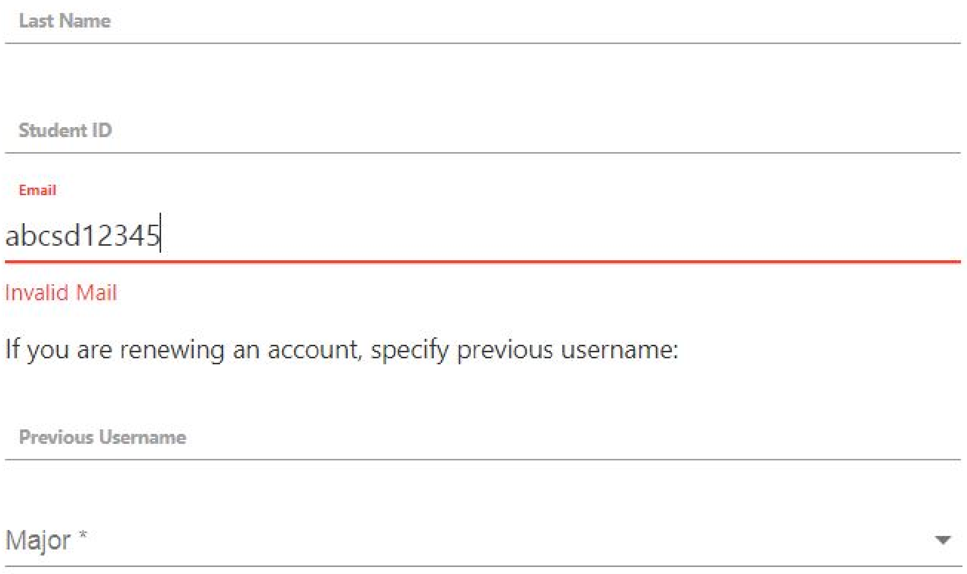
If you are a non-CS major taking a CS course, please provide course number:

Course Number: NA

Honeypot: Ajith

recpatcha

Figure 16: System detecting invalid email



Last Name

Student ID

Email

abcsd12345

Invalid Mail

If you are renewing an account, specify previous username:

Previous Username

Major *

Figure 17: Submit button inactivated when form is not filled

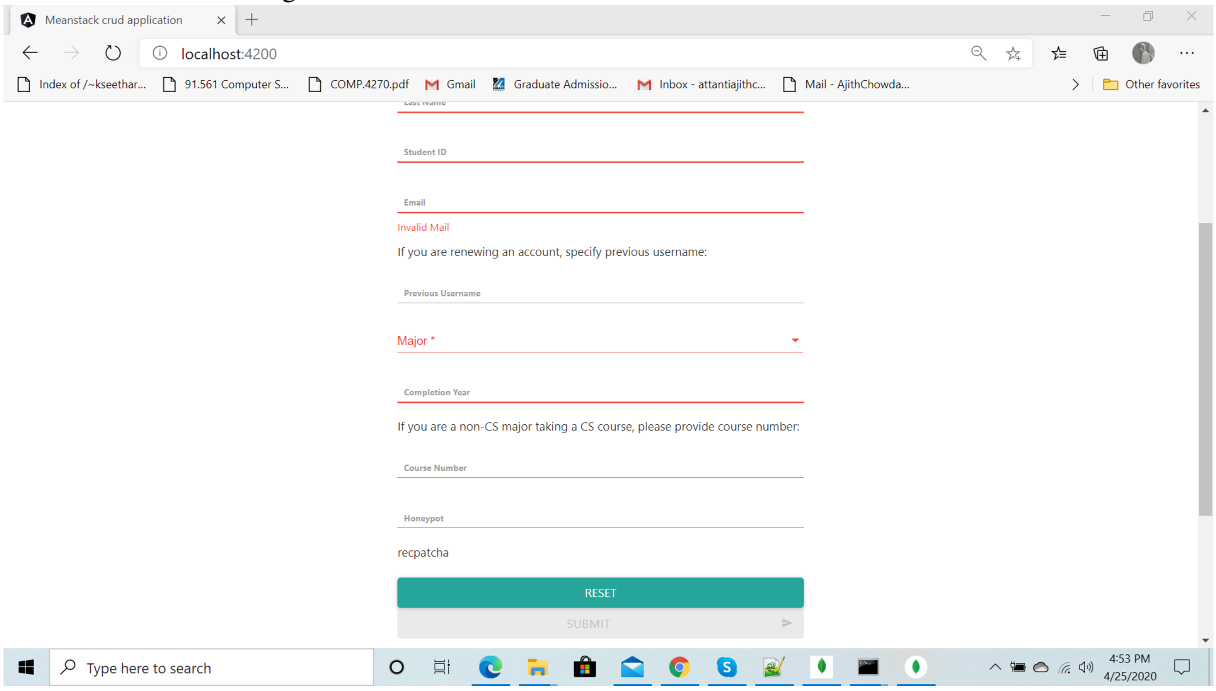
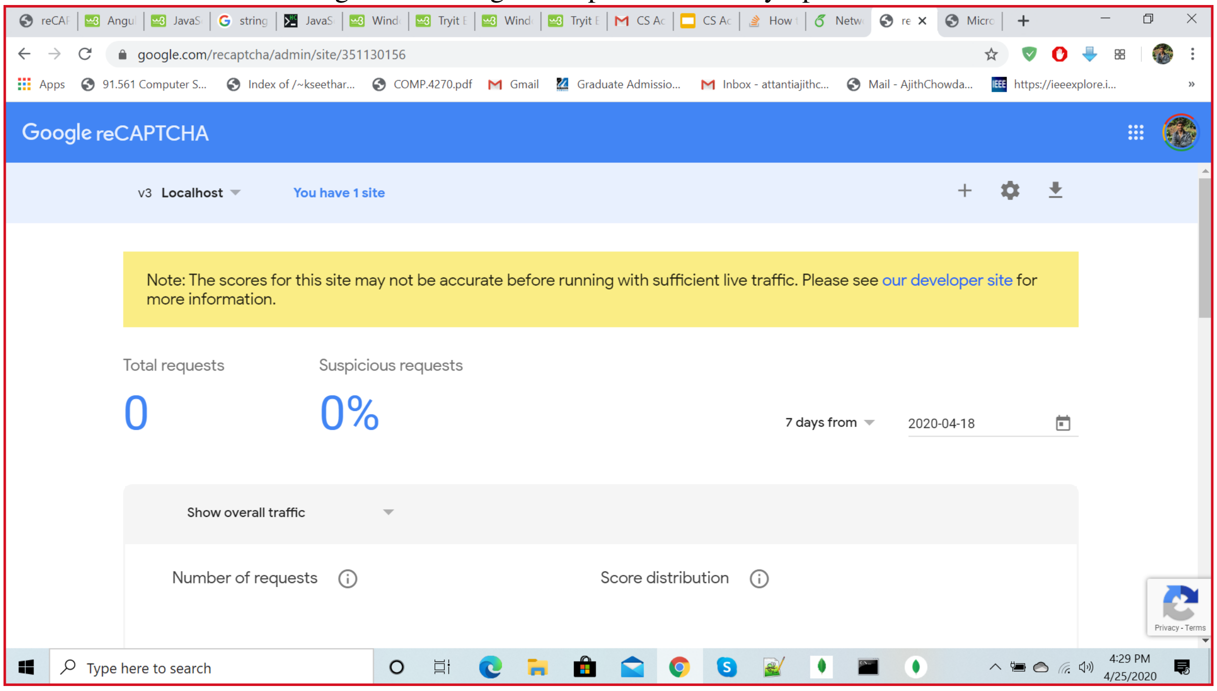

Meanstack crud application
localhost:4200
Student ID
Email
Invalid Mail
If you are renewing an account, specify previous username:
Previous Username
Major *
Completion Year
If you are a non-CS major taking a CS course, please provide course number:
Course Number
Honeypot
recpatcha
RESET
SUBMIT

Figure 18: Google reCaptcha to identify SpamBots


google.com/recaptcha/admin/site/351130156
Google reCAPTCHA
v3 Localhost
You have 1 site
Note: The scores for this site may not be accurate before running with sufficient live traffic. Please see our developer site for more information.
Total requests
0
Suspicious requests
0%
7 days from
2020-04-18
Show overall traffic
Number of requests
Score distribution