Software Engineering- CSC 4350 Spring 2017

# An encryption and decryption system for message communication

**ADEPT**

Amani Konduru

Benjamin Garber (Daniel)

Edward Bull

Paul David Utesch

Team

4/18/2017

Table of Contents

[I. Introduction 3](#_Toc480770119)

[II. Requirements Traceability Matrix (RTM) 4](#_Toc480770120)

[III. Use Case, Sequence and Interaction diagrams 5](#_Toc480770121)

[IV. Object Design 5](#_Toc480770122)

[V. Test Cases 6](#_Toc480770123)

[VI. Rationale 13](#_Toc480770124)

[VII. Fictional Point Cost Analysis and COCOMO 14](#_Toc480770125)

[VIII. Project Legacy 15](#_Toc480770126)

[IX. WSD 16](#_Toc480770127)

[X. Gantt Chart 16](#_Toc480770128)

[XI. Dictionary 18](#_Toc480770129)

[XXII. Resumes 20](#_Toc480770130)

[XXII. User Guide 24](#_Toc480770131)

[XXIII. Database 24](#_Toc480770132)

# I. Introduction

Topic: A secure mail server and client pair

The Adept Mail system will be composed of two parts, a server and client. The first part, the Adept Mail Server, will listen on specified ports for IMAP and SMTP communication. It will be able to receive and store emails between its list of authenticated users. All network communication will be secured via SSL/TLS, and client requests will be authenticated via an IMAP authentication exchange. The Adept Mail Server will support multiple concurrent connections and will use a PostgreSQL database for storage. The Adept Mail Server will be decoupled from the database, so that multiple Adept Mail Servers could communicate with the same database or database system

The second part, the Adept Mail Client, will interact with a user via either a CLI or GUI interface. The client can authenticate, update local storage of emails, and manage their email account on the Adept Mail Server where the canonical storage of their emails will take place. The Adept Mail Client will communicate over SSL/TLS for security, and all requests will be made in properly formed IMAP or SMTP exchanges as appropriate. Local storage will be encrypted and only decrypted upon viewing. Unencrypted emails will not be stored in anything but RAM during the process.

While interoperability with other mail servers may not be feasible as a student project in a single semester, by adhering to the IMAP and SMTP protocol definitions in a minimally compliant fashion we can demonstrate how interoperability is accomplished in the real world. Additionally, while a truly secure program may also not be feasible (often for professional teams as well), we can demonstrate the fundamentals about how network connections, passwords, local data storage, and SQL queries can be secured.

# II. Requirements Traceability Matrix (RTM)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Entry | Specification | Type | Build | Use Case Name | Category |
| 1 | The Adept Mail Server shall store user e-mails in a database. | SW | 1 | Send Email, Edit Emails | Server |
| 2 | The Adept Mail Server shall move user e-mails between mailboxes upon an authenticated request from that user. | SW | 2 | Edit Folders | Server |
| 3 | The Adept Mail Server shall delete user-designated e-mails from its database upon an authenticated request from that user. | SW | 2 | Delete Emails | Server |
| 4 | The Adept Mail Server shall serve user data when authenticated requests are received from the Adept Mail Client via a minimally compliant IMAP protocol. | SW | 1 | Serve Updates | Server |
| 5 | The Adept Mail Server shall send user emails to other Adept Mail Servers upon an authenticated request from that user. | SW | 2 | Send Email | Server |
| 6 | The Adept Mail Server shall receive user emails from other Adept Mail Servers via a minimally compliant SMTP protocol. | SW | 2 | Receive Email | Server |
| 7 | The Adept Mail Server shall encrypt all incoming and outgoing connections using the TLS 1.2 standard. | SW | 1 | Receive Email, Send External Email, Serve Updates, Edit Emails, Edit Folders, Authenticate | Server |
| 8 | The Adept Mail Server shall support multiple concurrent connections. |  | 1 | Receive Email, Serve Updates, Edit Emails, Edit folders | Server |
| 9 | The Adept Mail Client shall request user email data from the Adept Mail Server via a minimally compliant IMAP protocol. | SW | 1 | Request Update | Client |
| 10 | The Adept Mail Client shall store user email data locally in a local database. | SW | 1 | Request Update | Client |
| 11 | The Adept Mail Client shall send user emails to the Adept Mail Server via a minimally compliant SMTP protocol. | SW | 1 | Send Email | Client |
| 12 | The Adept Mail Client shall provide a graphical user interface to allow users to generate requests and view their emails. | SW | 2 | Authenticate, view Email, Manage Emails, Manage Folders | Client |
| 13 | The Adept Mail Client shall require local authentication from the user. | SW | 1 | Authenticate | Client |
| 14 | The Adept Mail Client shall provide remote authentication to the Adept Mail Server prior to executing any requests. | SW | 1 | Manage Emails, Manage Folders | Client |
| 15 | The Adept Mail Client shall locally encrypt and decrypt the subject and body of every email it sends and receives, respectively, using symmetric-key block encryption based on a user provided password. | SW | 2 | Request Update | Client |

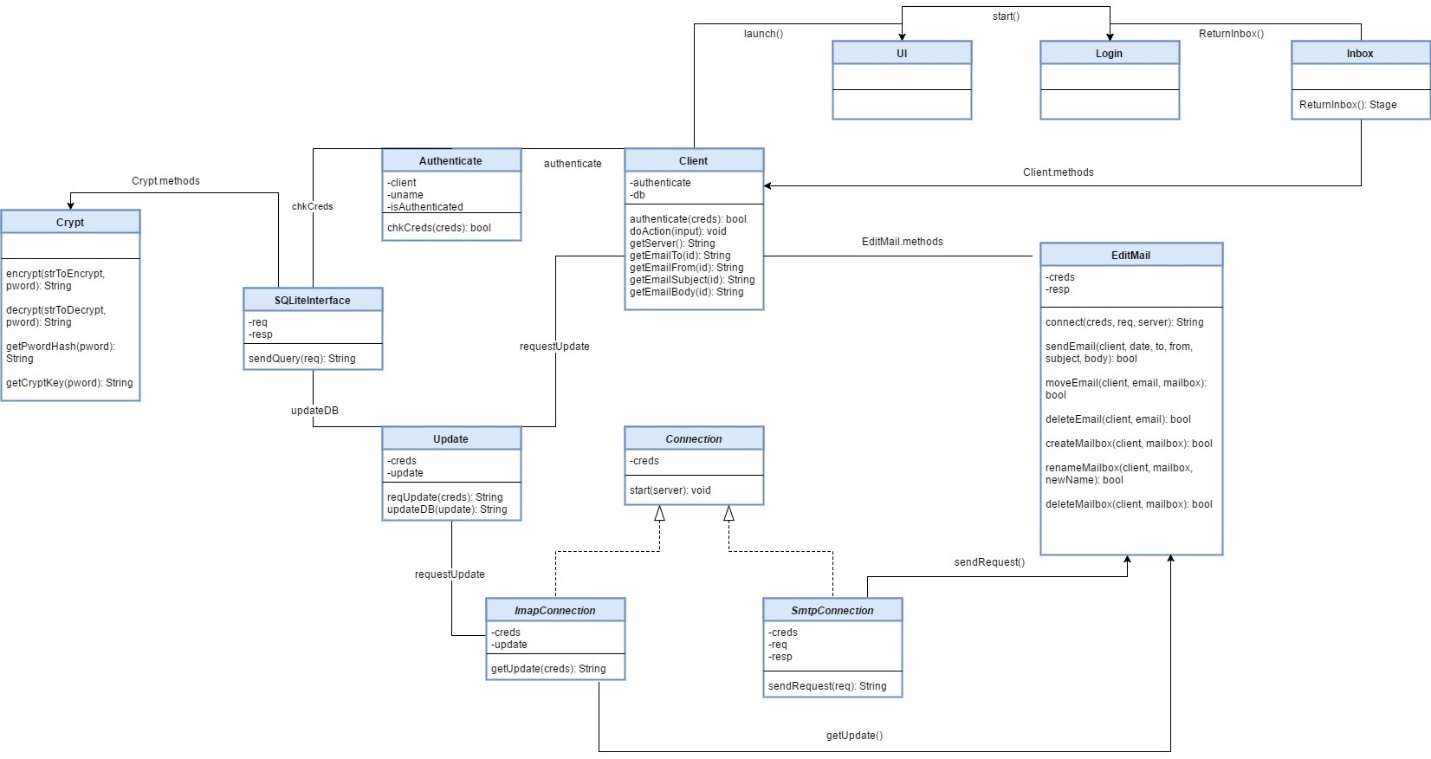
# III. Use Case, Sequence and Interaction diagrams

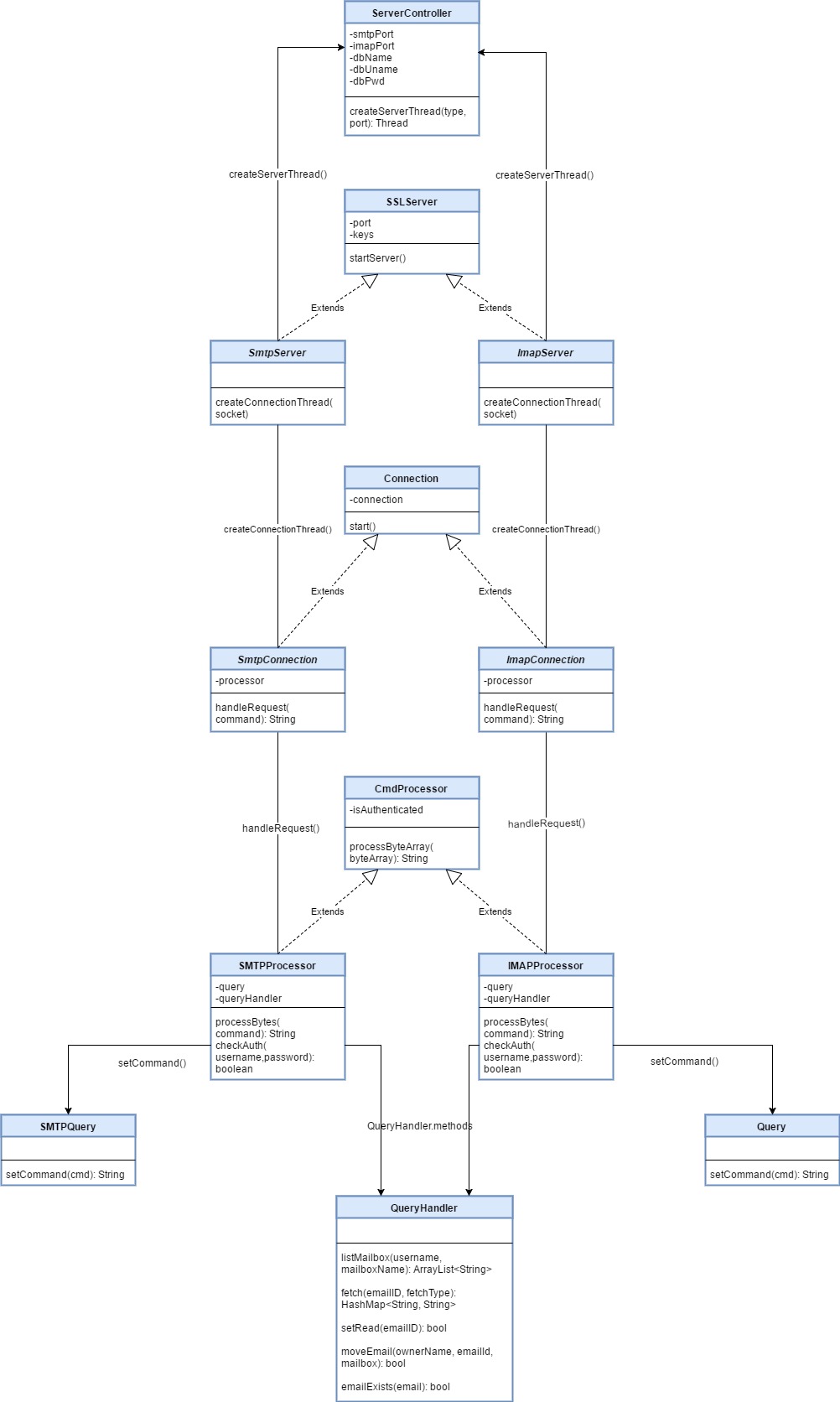
Use Case Diagrams



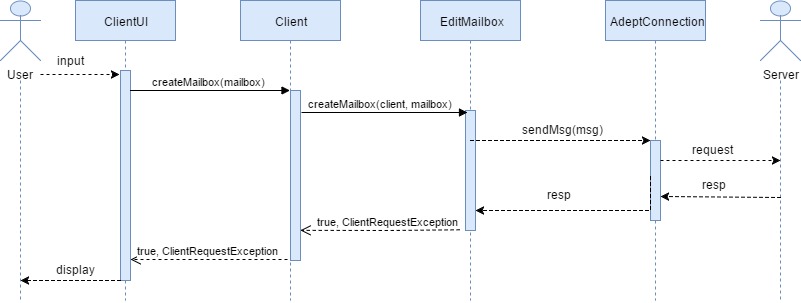


Class Diagrams

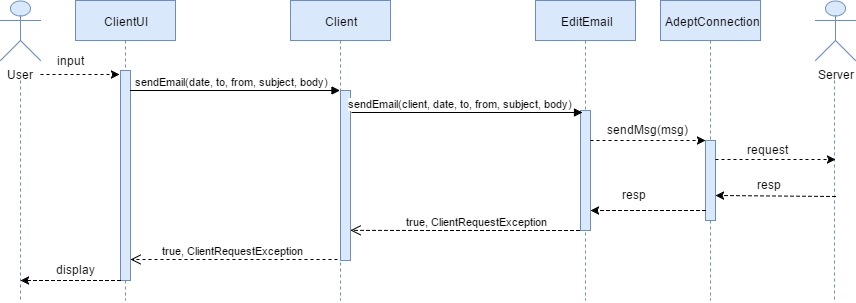




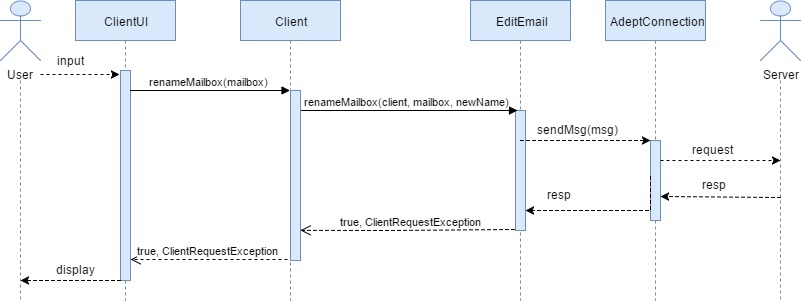
Sequence Diagrams



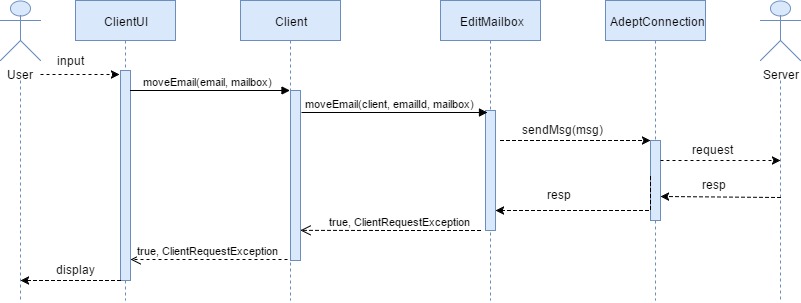
Here, the user wants to create a new mailbox. Using the UI to select the appropiate action, the user will be prompted for a mailbox name. After pressing enter, a new mailbox will be created will the provided string as the name.

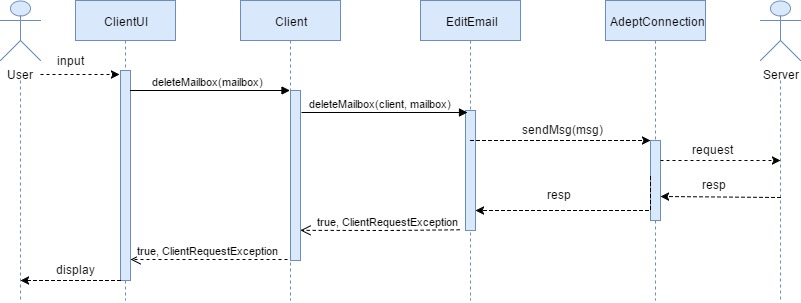


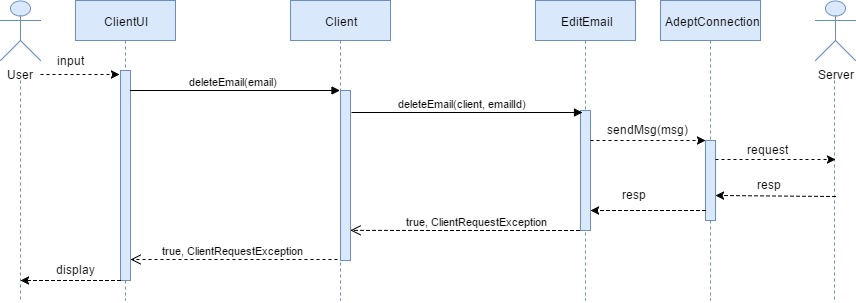
Here, the user wants to send an email. Using the UI to select the appropriate action, the user will be prompted for an email address to send to, a subject and a body. Upon pressing send, an email will be sent to that user’s mailbox if they are in the server.



Here, the user wants to rename an existing mailbox. First the user must select the mailbox they wish to rename. The user will then be prompted to enter the new name for the mailbox. Upon pressing enter, the selected mailbox will be renamed to the string provided by the user. All emails will be moved to the new mailbox with it.

 Here, the user wants to change the mailbox that an email resides in. First, the user must select the email they would like to move. Then, the user must select the appropriate UI element. They will be prompted for the name of the existing mailbox they would like to move the email to. Upon pressing enter, the email will be moved to the provided mailbox if the mailbox exists. If it doesn’t, the operation will fail.

 Here, the user wants to delete a mailbox. First, the user must select the mailbox they want to delete. Then, the user will select the appropriate UI element. The user will be prompted to confirm that they want to delete the mailbox. After pressing enter, the mailbox will be deleted.



Here, the user wants to delete an email. First, the user must select the email that they want to delete. Then, they will select the appropriate UI element. The user will be prompted for a confirmation. Upon confirming, the selected email will be deleted.

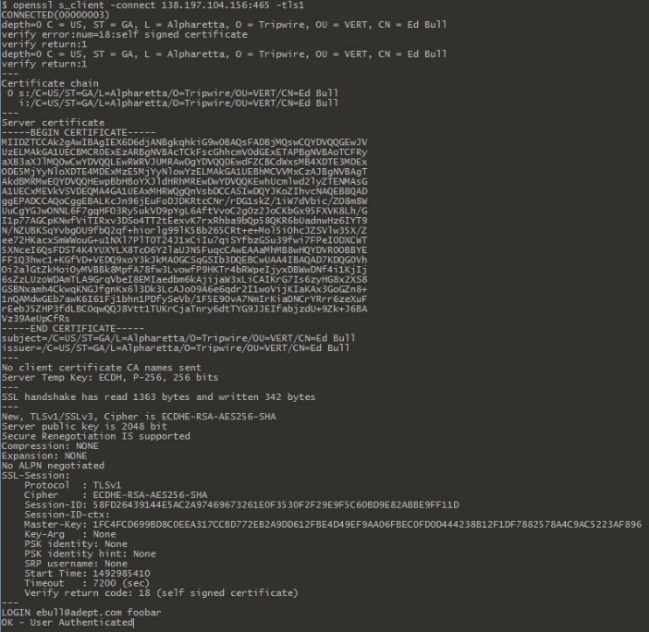
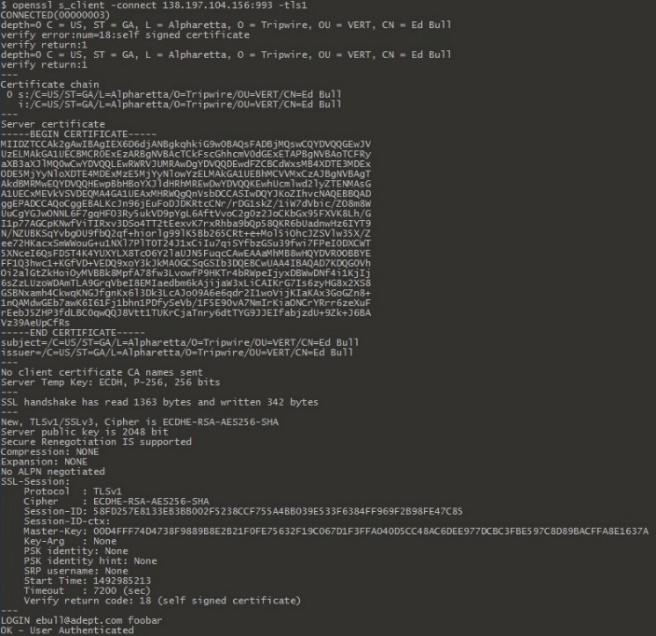
# IV. Object Design

# V. Test Cases

Note: We have omitted the SendExternalEmailFunctionality and RenameMailboxFunctionality test cases due to time and development constraints.

|  |  |
| --- | --- |
| Test-case Identifier | ServerConnectivity |
| Feature | RTM: 7, 8, 14 |
| Feature Pass/Fail Criteria | The test passes if each request receives a properly formed protocol compliant response. |
| Means of Control | IMAP, SMTP direct connections |
| Data | A series of protocol specific commands (both well-formed and mal-formed) and their expected protocol-specific responses. |
| Test Procedure | Using a third party tool, the tester will create a series of concurrent connections to the server on both its IMAP and SMTP interfaces. |
| Special Requirements | An authenticated account on the test server. |

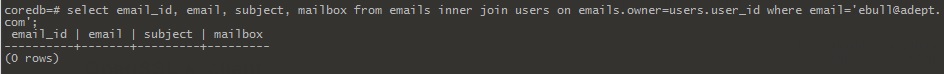
Target: Our Digital Ocean VM at 138.197.104.156 running the latest version of server.jar to demonstrate internet connectivity.

Test 1: IMAP Test 2: SMTP

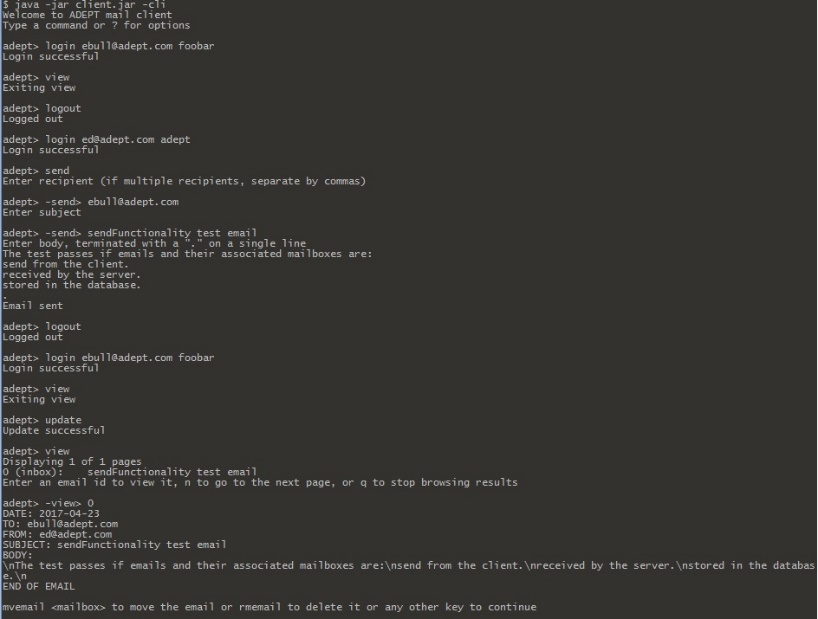
As is visible from the openssl s\_client information, SSL authentication was successful (if naïve) and the LOGIN command received the expected output on both ports. Note that we are using IMAP authentication on the SMTP port. This is an intentional time-saving implementation.

|  |  |
| --- | --- |
| Test-case Identifier | SendFunctionality |
| Feature | RTM: 1, 5, 6, 7, 8, 11, 13, 14 |
| Feature Pass/Fail Criteria | The test passes if emails and their associated mailboxes are sent from the client, received by the server, and stored in the database. |
| Means of Control | Client UI or Client CLI |
| Data | Placeholder content and names will be used for both the email and mailboxes. |
| Test Procedure | Using an authenticated account, the tester will use the client UI to send an e-mail to another authenticated account on the same server. |
| Special Requirements | Two authenticated accounts on the test server. |

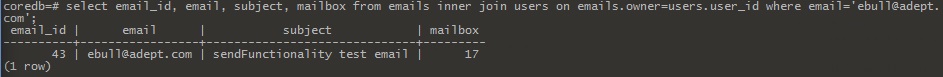
Pre-test condition: The email [ebull@adept.com](mailto:ebull@adept.com) has no current emails before the test.



Then the test is run and verified in the Adept Mail Client CLI. The second email account [ed@adept.com](mailto:ed@adept.com) is used to send the email to [ebull@adept.com](mailto:ebull@adept.com).

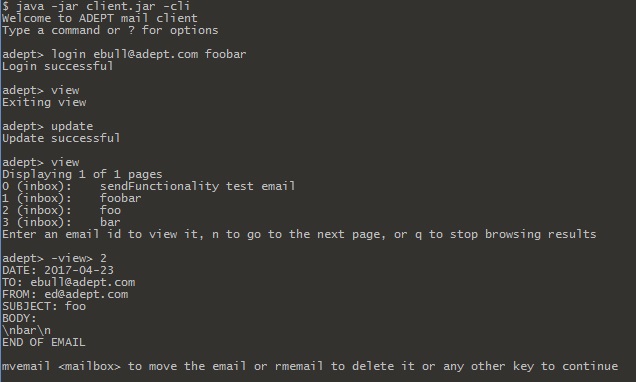


Post-test condition: The email [ebull@adept.com](mailto:ebull@adept.com) has received the test email.



|  |  |
| --- | --- |
| Test-case Identifier | UpdateFunctionality |
| Feature | RTM: 4, 7, 8, 9, 10, 12, 13, 14, 15 |
| Feature Pass/Fail Criteria | The test passes if the server receives the update request from the client, sends back e-mail data, and the client updates its local database with that data. |
| Means of Control | Client UI or Client CLI |
| Data | Placeholder content and names will be used for both the email and mailboxes. |
| Test Procedure | Using an authenticated account, the tester will use the client UI to request a mailbox update from the server. Once the update is complete, the tester will verify that all data has transferred correctly. |
| Special Requirements | An authenticated account on the test server. No emails stored on local storage. |

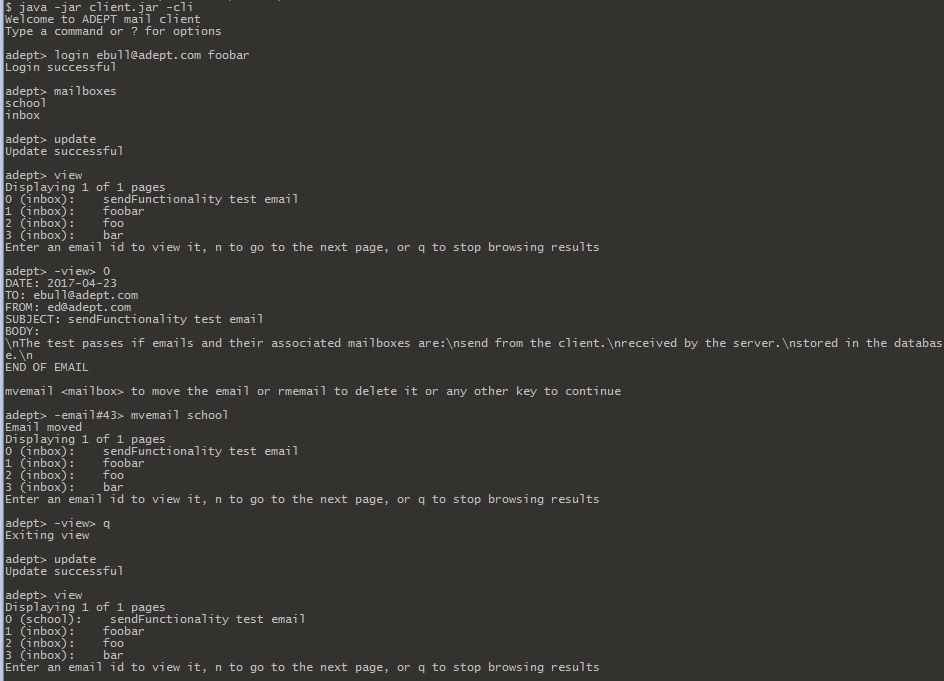
Pre-test condition: By deleting the Adept Mail Client's local sqlite database, we can ensure a valid testing pre-condition. The emails do not show in the [ebull@adept.com](mailto:ebull@adept.com) inbox.



Post-test condition: After running the update, the emails are visible in the [ebull@adept.com](mailto:ebull@adept.com) inbox.

|  |  |
| --- | --- |
| Test-case Identifier | MoveEmailFunctionality |
| Feature | RTM: 1, 2, 4, 8, 9, 12, 13, 14 |
| Feature Pass/Fail Criteria | The test passes if the e-mail is moved from one mailbox to another mailbox, and then deleted. |
| Means of Control | Client UI or Client CLI |
| Data | Placeholder content and names will be used for both the email and mailboxes. |
| Test Procedure | Using an authenticated account, the tester will use the client UI to move an email from one mailbox to another mailbox. The tester will confirm the move. |
| Special Requirements | An authenticated account on the test server. |

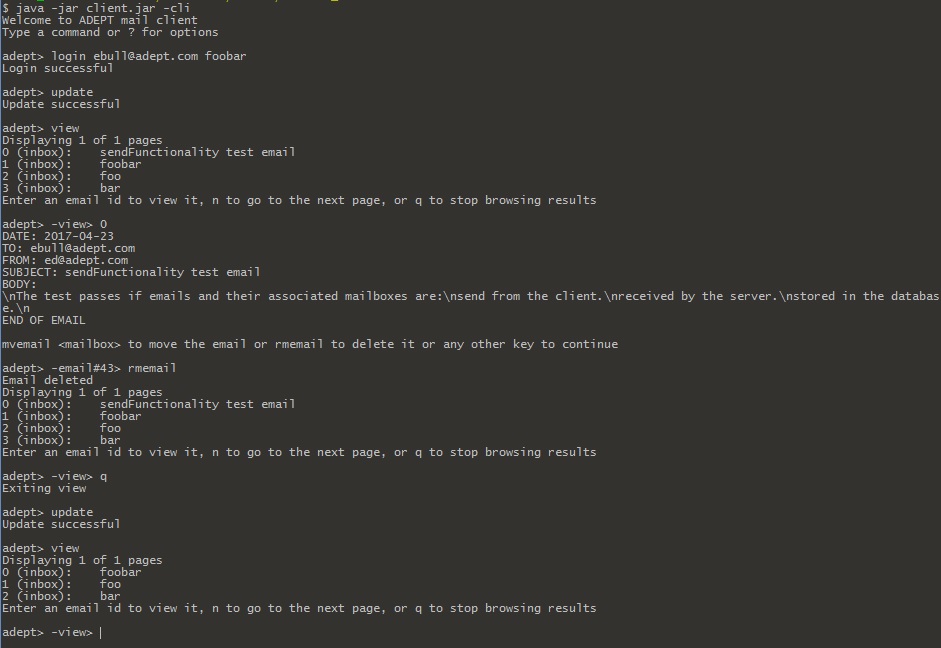
Pre-test condition: A valid email for [ebull@adept.com](mailto:ebull@adept.com) in the "inbox" mailbox.



Post-test condition: The email is now in the "school" mailbox.

|  |  |
| --- | --- |
| Test-case Identifier | DeleteEmailFunctionality |
| Feature | RTM: 1, 3, 4, 8, 9, 12, 13 |
| Feature Pass/Fail Criteria | The test passes if the e-mail is moved from one mailbox to another mailbox, and then deleted. |
| Means of Control | Client UI or Client CLI |
| Data | Placeholder content and names will be used for both the email and mailboxes. |
| Test Procedure | Using an authenticated account, the tester will use the client UI to delete an e-mail. The tester will confirm the deletion. |
| Special Requirements | An authenticated account on the test server. |

Pre-test condition: An email is in the [ebull@adept.com](mailto:ebull@adept.com) mailbox.



Post-test condition: The email is no longer in the [ebull@adept.com](mailto:ebull@adept.com) mailbox.

|  |  |
| --- | --- |
| Test-case Identifier | CreateMailboxFunctionality |
| Feature | RTM: 1, 4, 8, 9, 12, 13 |
| Feature Pass/Fail Criteria | The test passes if a new mailbox is created. |
| Means of Control | Client UI or Client CLI |
| Data | Placeholder content and names will be used for both the email and mailboxes. |
| Test Procedure | Using an authenticated account, the tester will use the client UI to create a new mailbox. The tester will confirm that the new mailbox is present. |
| Special Requirements | An authenticated account on the test server. |

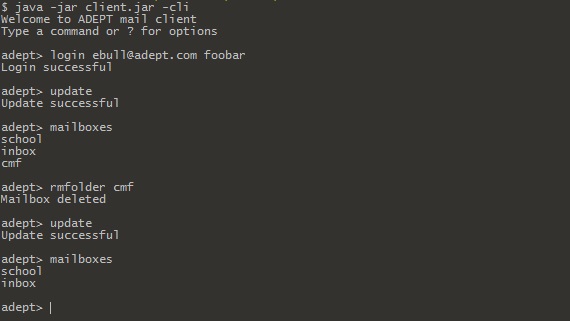
Pre-test condition: The account [ebull@adept.com](mailto:ebull@adept.com) does not have the mailbox 'cmf'.



Post-test condition: The account [ebull@adept.com](mailto:ebull@adept.com) has the mailbox 'cmf'.

|  |  |
| --- | --- |
| Test-case Identifier | DeleteMailboxFunctionality |
| Feature | RTM: 1, 4, 8, 9, 12 13 |
| Feature Pass/Fail Criteria | The test passes if the e-mail is moved from one mailbox to another mailbox, and then deleted. |
| Means of Control | Client UI or Client CLI |
| Data | Placeholder content and names will be used for both the email and mailboxes. |
| Test Procedure | Using an authenticated account, the tester will use the client UI to move an email from one mailbox to another mailbox. The tester will confirm the move, then delete the e-mail. The tester will confirm the deletion. |
| Special Requirements | An authenticated account on the test server. |

Pre-test condition: The account [ebull@adept.com](mailto:ebull@adept.com) has the mailbox 'cmf'.



Post-test condition: The account [ebull@adept.com](mailto:ebull@adept.com) no longer has the mailbox 'cmf'.

Our test cases cover the following requirements.

* ServerConnectivity: 7, 8, 14
* SendFunctionality: 1, 5, 6, 7, 8, 11, 13, 14
* UpdateFunctionality: 4, 7, 8, 9, 10, 12, 13, 14, 15
* MoveEmailFunctionality: 1, 2, 4, 8, 9, 12, 13, 14
* DeleteEmailFunctionality: 1, 3, 4, 8, 9, 12, 13
* CreateMailboxFunctionality: 1, 4, 8, 9, 12, 13
* DeleteMailboxFunctionalilty: 1, 4, 8, 9, 12 13

The set of these requirements is the set of the RTM {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}. If these tests complete, we can say with confidence that we have tested the use cases in our system.

In a real world scenario, there would need to be further software testing, including unit testing and regression testing. Because a large numbers of errors can be reported between the Server and Client via the SMTP and IMAP protocols, significant handling would need to be incorporated in that respect.

In addition on the software side, the current implementation of the server would need to incorporate real logging functionality and alerts. Security penetration testing would be important to identify and sanitize possible SQL injection vectors.

Finally, implementation of the Adept Mail System would require significant hardware testing. Any Adept Mail Server would need to undergo stress testing to verify that it could withstand the estimated maximum number of user connections. There might need to be security testing on the hardware aspect as well (ex. using an Adaptive Security Appliance or firewall to defend against possible denial of service attacks).

# VI. Rationale

The Adept Mail Server and Adept Mail Client will offer a secured email alternative for users interested in the confidentiality and integrity of their communications. Like competitive mail options, the Adept system will communicate over a secured TLS channel. That means that network sniffers and nodes between the Client and Server will not be able to intercept traffic as plain text. Unlike many other options, the Client will encrypt sensitive fields of its local storage and only decrypt those fields in RAM upon viewing. This ensures that even if the device on which the Client is installed is compromised physically or otherwise, the local storage won't be.

Our system will demonstrate some key points on security, including secure network communications, encryption, password hashing, and SQL sanitization. While our first iteration will be far from fully secure (many professional products are as well!) it will be an excellent learning experience and demonstration of techniques. The framework is in place for later improvements including parameterization of SQL inputs, TLS certificate verification, and more robust password hashing algorithms.

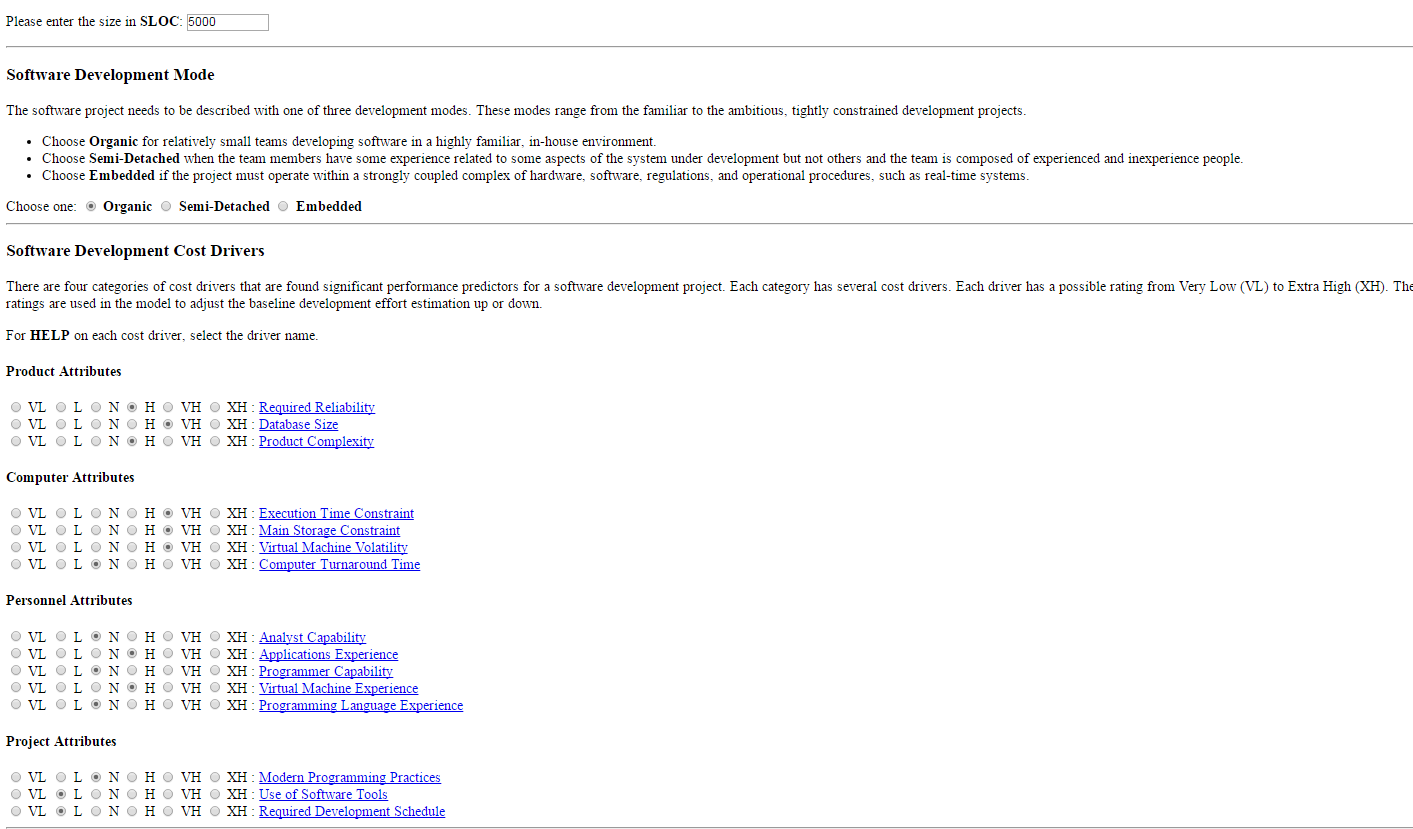
Another important feature of the Adept Mail System is protocol compliance. The Adept Mail Client communicates with the Server using minimally compliant but properly formed IMAP and SMTP commands over appropriate network ports. This means that extending the Adept Mail Client and Server for interoperability with third party mail clients and servers is already close at hand. Imagine sending emails to Gmail and Microsoft accounts via your own homebrewed solution! There are obstacles, of course, and full compliance is a tall order even for professional teams and products that have been established for decades. But the Adept Mail System demonstrates that TLS, IMAP, and SMTP are not black magic—it's achievable even for students to research and comply with standard communications protocols.

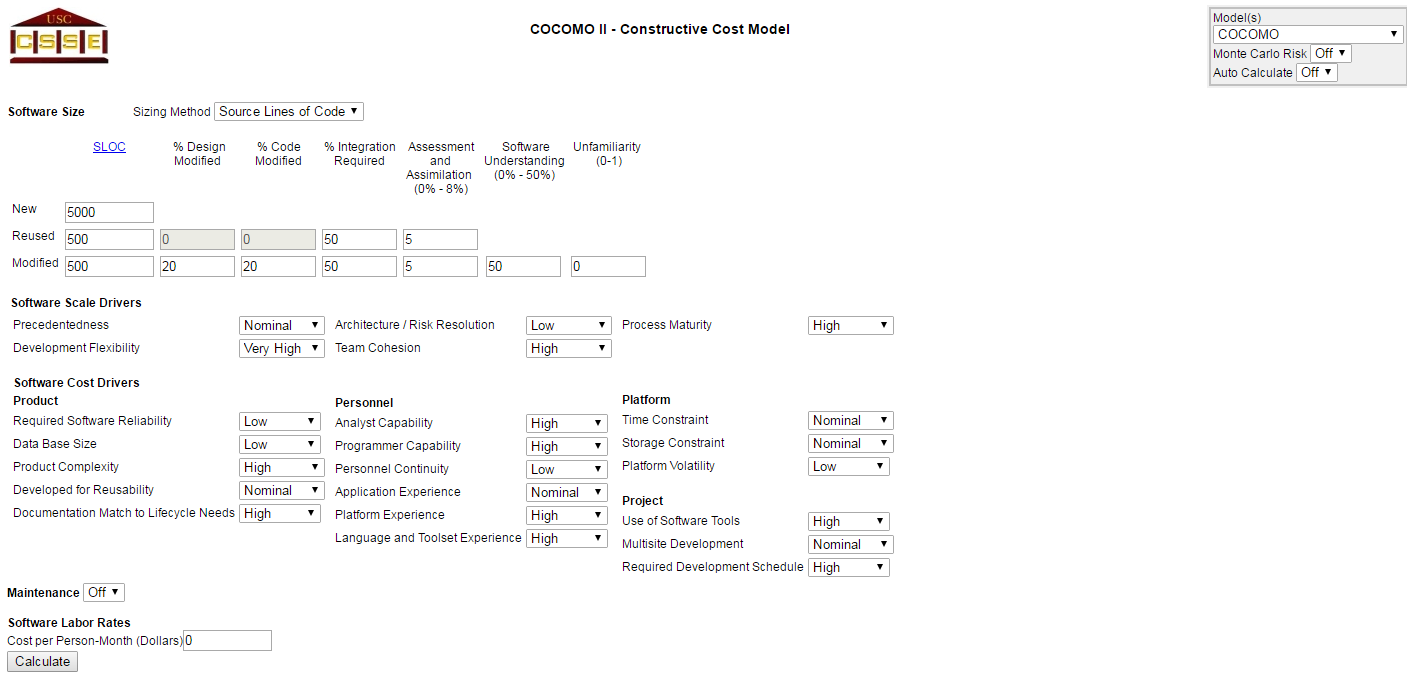
The functionality goals of the Adept Mail System are:

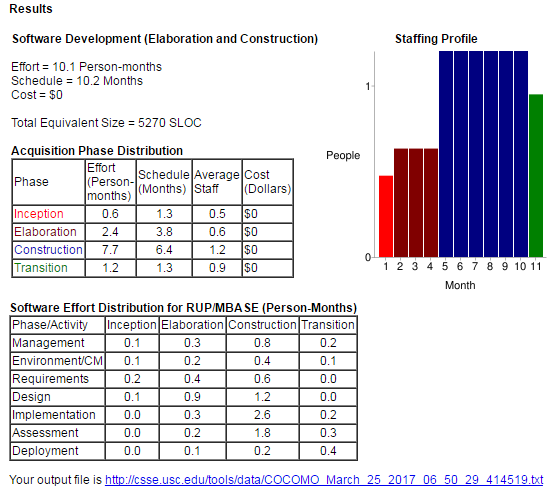
* To allow users to send and receive emails with other Adept Mail Clients.
* To allow users to manage their mails—including creating and deleting mailboxes and moving and deleting emails.
* To allow users to securely decrypt and view their stored encrypted emails.
* To allow the creation of local user accounts, each of which specifies its own parent Adept Mail Server, port settings, and secret key.
* To serve concurrent requests to the Adept Mail Server as quickly as possible.

# VII. Fictional Point Cost Analysis and COCOMO

(with comparison and conclusions)







# VIII. Project Legacy

# 

# IX. WSD

|  |  |
| --- | --- |
| Name | Role |
| Amani Konduru | Project manager, back-end (PostgreSQL), documentation |
| Benjamin Garber (Daniel) | Server developer (IMAP LIST, SMTP MAIL TO), documents |
| Edward Bull | Server developer, Client developer (except GUI), back-end (SQLite3), back-end (PostgreSQL), documents |
| Paul David Utesch | GUI developer and back-end (PostgreSQL) |

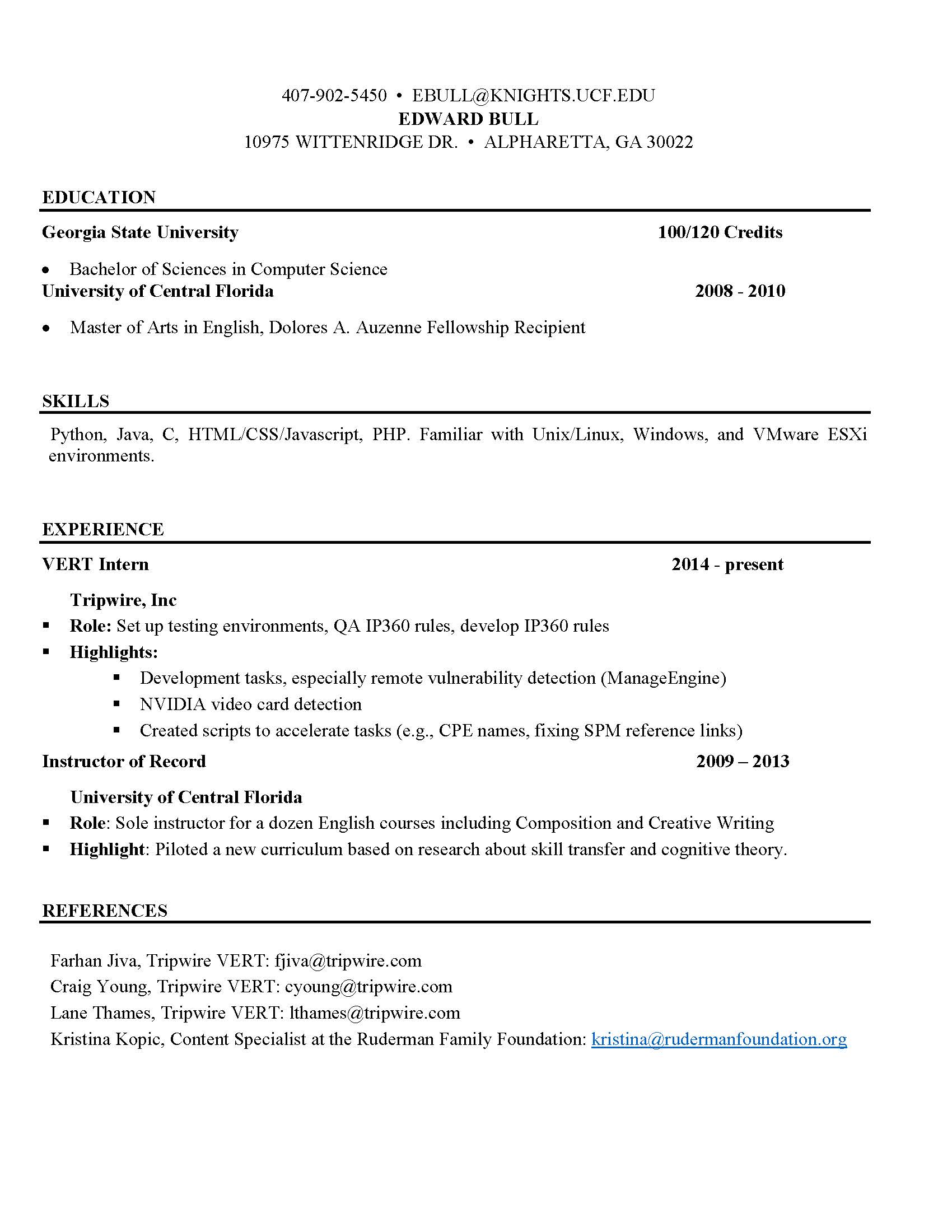
# X. Gantt Chart

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Title | ID | Participants | Status | Estimate Pts. |
| Create V1 Stories | S-01001 | Ed Bull,akonduru | Done | 1 |
| Document 2 - Requirements Elicitation | S-01017 | akonduru | Done | 1 |
| Learn how to use VersionOne | S-01022 | Ed Bull,bgarber,akonduru | Done | 1 |
| Set up group Github | S-01018 | Ed Bull | Done | 1 |
| Plan Server Database And Write CreateTables SQL Script | S-01019 | Putesch,akonduru | Future | 1 |
| Set up PostgresSQL test server | S-01009 | akonduru | In Progress | 1 |
| Implement Test Database | S-01020 | akonduru | Future | 1 |
| Server ServerController | S-01002 | Ed Bull | In Progress | 1 |
| Server SmtpServer | S-01003 | Ed Bull | In Progress | 1 |
| Server ImapServer | S-01004 | Ed Bull | In Progress | 1 |
| Server SmtpConnection | S-01005 | Ed Bull | In Progress | 1 |
| Server ImapConnection | S-01006 | Ed Bull | In Progress | 1 |
| Server CmdProcessor | S-01007 | bgarber | In Progress | 1 |
| Server QueryGenerator | S-01008 |  | Future | 1 |
| Server SmtpClient | S-01021 |  | Future | 1 |
| Document 2 Title Page | S-01037 | akonduru | Done | 1 |
| Document 2 Problem Statements | S-01038 | Ed Bull | Done | 1 |
| Document 2 RTM | S-01039 | Ed Bull | Done | 1 |
| Document 2 WSD | S-01040 | akonduru | Done | 1 |
| Document 2 Gantt | S-01041 | akonduru | Done | 1 |
| Document 2 Dictionary | S-01042 | Ed Bull,bgarber | Done | 1 |
| Set up PostgresSQL test server | S-01043 | akonduru | In Progress | 2 |
| Implement Test Database | S-01044 | akonduru | Future | 2 |
| Server ServerController | S-01045 | Ed Bull | In Progress | 2 |
| Server SmtpServer | S-01046 | Ed Bull | In Progress | 2 |
| Server ImapServer | S-01047 | Ed Bull | In Progress | 2 |
| Server SmtpConnection | S-01048 | Ed Bull | In Progress | 2 |
| Server ImapConnection | S-01049 | Ed Bull | In Progress | 2 |
| Server CmdProcessor | S-01050 | bgarber | In Progress | 2 |
| Server QueryGenerator | S-01051 |  | Future | 2 |
| Server SmtpClient | S-01052 |  | Future | 2 |
| Collate Document 3 | S-01024 | akonduru | In Progress | 2 |
| Document 3 Title Page | S-01025 | akonduru | Accepted | 2 |
| Document 3 RTM (5 columns) | S-01027 | Ed Bull | Accepted | 2 |
| Document 3 Use Cases and Int. Diagrams | S-01028 | Ed Bull | Done | 2 |
| Document 3 Function Point Analysis | S-01029 | bgarber | Done | 2 |
| Document 3 Database To Be Used | S-01030 | akonduru | Future | 2 |
| Document 3 Updated WSD | S-01031 | akonduru | Done | 2 |
| Document 3 Updated Gantt Chart | S-01032 | Ed Bull | In Progress | 2 |
| Document 3 Dictionary | S-01033 | Ed Bull | Accepted | 2 |
| Document 3 Use Cases Rationale | S-01034 | Ed Bull | Done | 2 |
| Document 3 horizontal prototype | S-01035 | Putesch | In Progress | 2 |
| Document 4 #2 | S-01063 | akonduru | Accepted | 1 |
| Document 4 #4 | S-01064 | Putesch | Accepted | 1 |
| Document 4 #5 | S-01065 | Ed Bull | Accepted | 1 |
| Document 4 #9 | S-01066 | Ed Bull,Putesch,akonduru | Accepted | 1 |
| Document 4 | S-01067 | Ed Bull,akonduru | Accepted | 1 |
| Document 6 #3 | S-01068 | Ed Bull | In Progress | 2 |
| Document 6 #4 | S-01069 | bgarber | In Progress | 1 |
| Document 6 #5 | S-01070 | akonduru | In Progress | 2 |
| Document 6 #9 | S-01071 | akonduru | In Progress | 2 |
| Set up PostgresSQL test server | S-01053 | akonduru | In Progress | 2 |
| Implement Test Database | S-01074 | akonduru | Accepted | 2 |
| Server ServerController | S-01075 | Ed Bull | Accepted | 2 |
| Server SmtpServer | S-01076 | Ed Bull | Accepted | 2 |
| Server CmdProcessor | S-01079 | bgarber | Accepted | 2 |
| Server QueryGenerator | S-01078 | Ed Bull | Accepted | 2 |
| Server SmtpClient | S-01077 | Ed Bull | Accepted | 2 |
| Doc6 #3 | S-01068 | bgarber | Accepted | 1 |
| Doc6 #4 | S-01069 | akonduru | Accepted | 1 |
| Doc6 #5 | S-01070 | akonduru | Accepted | 1 |
| Doc6 #9 | S-01071 | Ed Bull | Accepted | 1 |
| Final Document | S-01072 | Ed Bull, akonduru, bgarber | In Progress |  |

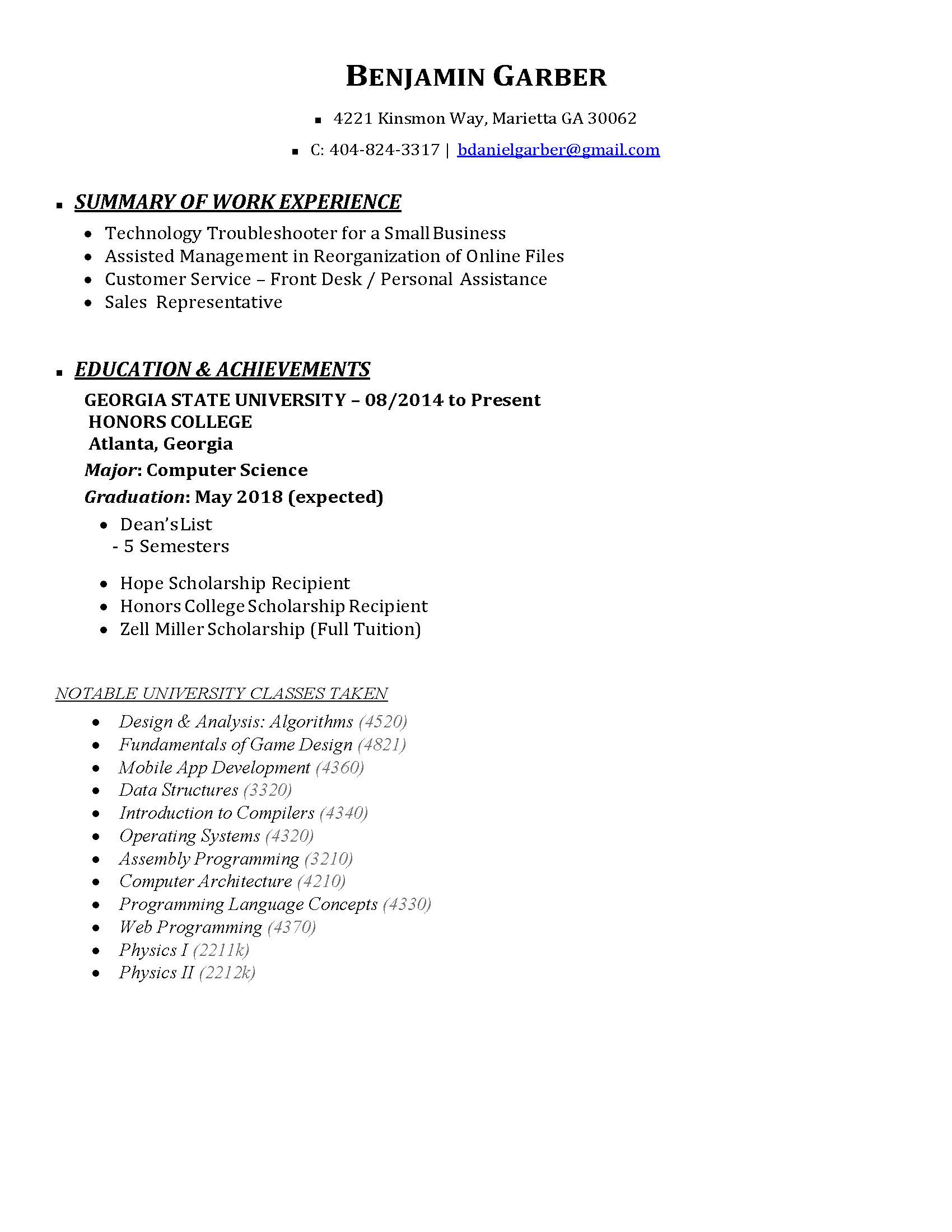
# XI. Dictionary

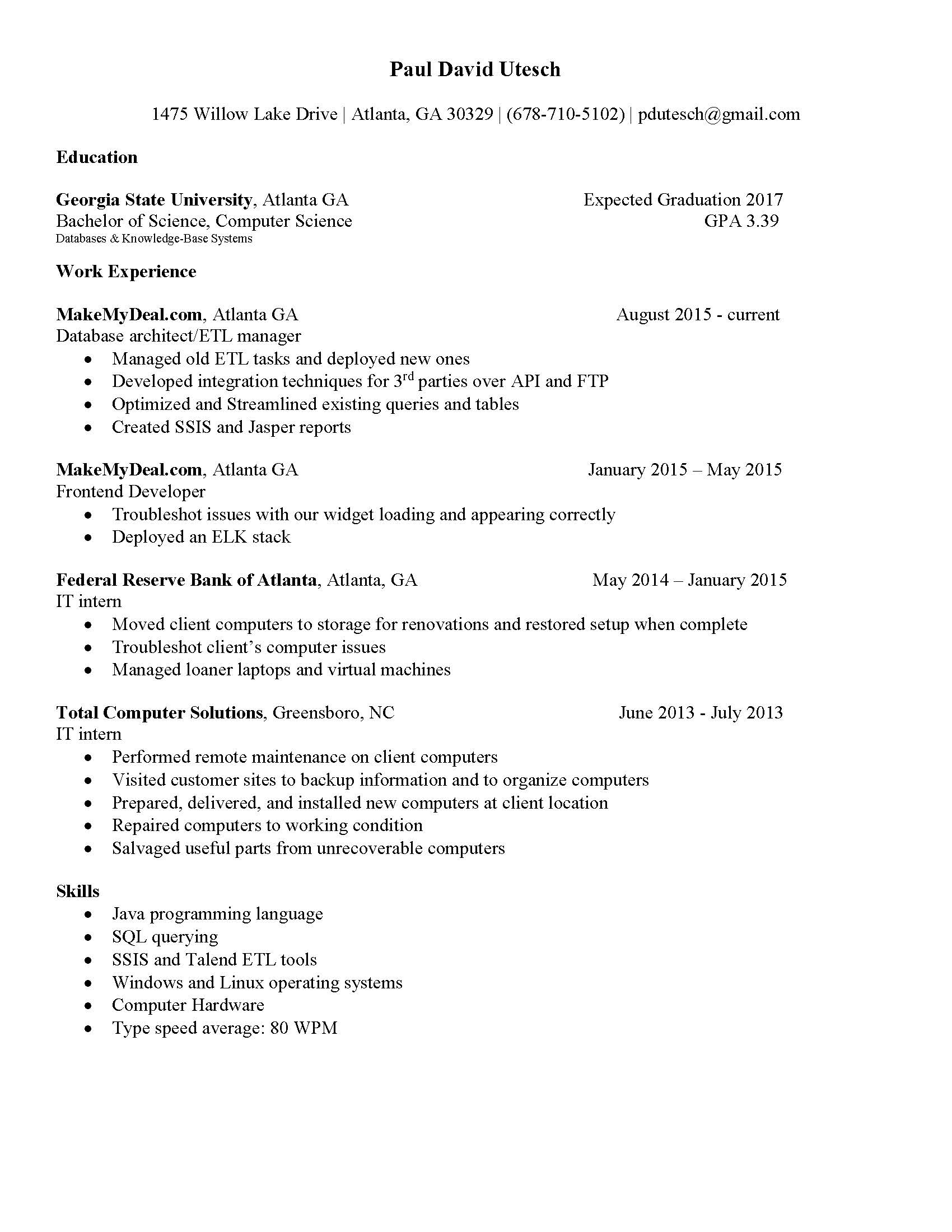
1. **Encryption:** the process of converting data into a code, to prevent unauthorized access. Encryption is the process of transforming data into an unreadable, encrypted form. The transformation is done using one of several cryptographic algorithms that leverage computationally difficult mathematical problems to make reversing the transformation difficult if not effectively impossible
2. **Symmetric Encryption:** Symmetric Encryption uses a key or set of keys to both encrypt and decrypt data. If data is to be shared between two parties, they must both have the key or keys to decrypt or encrypt the data.
3. **Asymmetric Encryption:** Asymmetric Encryption, also known as Public Key Encryption, is a type of encryption where anyone in possession of a public key can encrypt a message. That message can then only be decrypted with a private key. This method is often used for identity authentication because it is computationally expensive. Once authentication is completed, communications will then often transition into symmetric encryption after generating a symmetric encryption key.
4. **End to End Encryption:** Only the communicating users can read the messages.
5. **SSL/TLS (Secure Sockets Layer / Transport Layer Protocol):** TLS and the now- deprecated SSL it is based on are network security protocols meant to secure client-server connections using both symmetric encryption for data transfer and asymmetric encryption for identity authentication. While there are many options that can be set in an SSL/TLS session, the foundation of the protocols lie in using encryption to authenticate the identities of the connected parties and to secure the privacy of the data transferred between them.
6. **Server:** a server program awaits and fulfills requests from client programs, which may be running in the same or different computers.
7. **Client:** requesting program or user.
8. **Socket:** Is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.
9. **SMTP protocol:** Simple Mail Transfer Protocol. It is an Internet standard for electronic mail (email) transmission. SMTP was first defined by RFC 821 and updated in RFC 5321.
10. **IMAP protocol:** Internet Message Access Protocol. Itis an Internet standard protocol used by e-mail clients to retrieve e-mail messages from a mail server over a TCP/IP connection. IMAP is defined by RFC 3501.
11. **TCP/IP:** IP (Internet Protocol) is the basic communication language or protocol of the ozInternet. It can also be used as a communications protocol in a private network (either an intranet or an extranet). TCP (Transmission Control Protocol) is layered on top of IP to provide certain network control and data validation features for many internet communications.

# XXII. Resumes









# XXII. User Guide

# XXIII. Database

We used two databases: PostgreSQL and SQLite. The metadata is stored in a simple PostgreSQL database, with the user\_id as a primary key. The client program has some basic login functionality and a SQLite database was used to store usernames and passwords as well as the user’s local emails and mailbox structure. Both are open source and were best for our project from the limited options that are available. PostgreSQL is good for concurrency and was used for our server to support concurrent connections and operations. SQLite is easy to set up and work with, so it’s an ideal solution for our client program. Digital Ocean virtual machine “droplet” was used for testing our server database. Digital Ocean droplets are inexpensive, reliable, and can be snapshotted for easy rollbacks should something go wrong during development. Our Digital Ocean runs on Ubuntu for easy terminal access over SSH.

PostgreSQL was used to manage the server database. Database ‘coredb’ was created to maintain the Adept mail server. The database coredb has three tables: users, mailboxes and emails. Currently the server is running on a command line prompt using sequel queries. The users table contains six attributes: user\_id, email, password, certificate, last\_login and isadmin. The user\_id attribute is the primary key for this table. This allows us to distinguish each person in the Adept server. The user\_id has a serial datatype so it auto increments as each user registers. The email attribute is unique for each user and lower case index was added to ignore case. The email attribute uses character varying datatype which limits the email to hundred and fifty characters. The password attribute also users character varying and limits the password of the user to hundred characters. Password-hash was supposed to be implemented, but was unable to due to time constraints. Certificate attribute is set to character varying with no limit it stores the SMPT certificate. The users table contains a last\_login attribute. A timestamp attribute with time zone datatype was used which allows the users to see when they last logged in. Unlike other database PostgreSQL allows us to implement date and time values very easily and efficiently. The last attribute in the users table was the isadmin attribute which is a boolean attribute, which flags to see if the user is an admin or not. All the attributes except for certificate are not NULL. The mailboxes table consists of three attributes: mailbox\_id, mailbox, and owner. The attribute mailbox\_id is the primary key and is being auto incremented. The attribute mailbox being the name of the mailbox such as inbox, spam, etc. The owner attribute of this table determines that different users could have the same mailbox name but it might belong to some other owner. Therefore, owner attribute is linked to user\_id in the users table as a foreign key. The mailbox\_id and owner are not NULL. The emails table consists of nine attributes: email\_id, owner, mailbox, date, to, from, subject, body and read. Where email\_id is the primary key and is auto incremented. The owner attribute here is also linked to the user\_id in the users table as a foreign key. The mailbox attribute is linked to mailbox attribute in the mailboxes table as a foreign key. Together the database keeps track of the user data by their user\_id, owner, and mailbox\_id attributes along with their written emails.

Figure 0.0: Relational diagram for the PostgreSQL database ‘coredb’

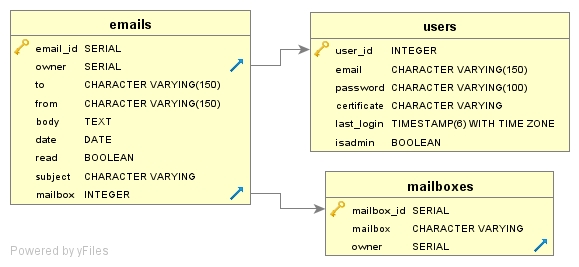


Figure 0.0 : Sample data for the users table

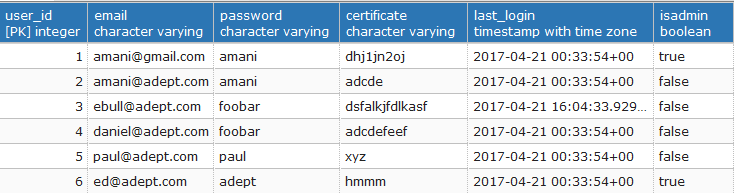


Figure 0.0 : Sample data for the mailboxes table

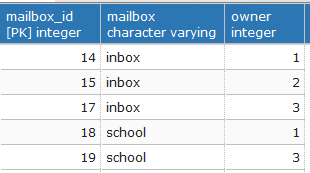
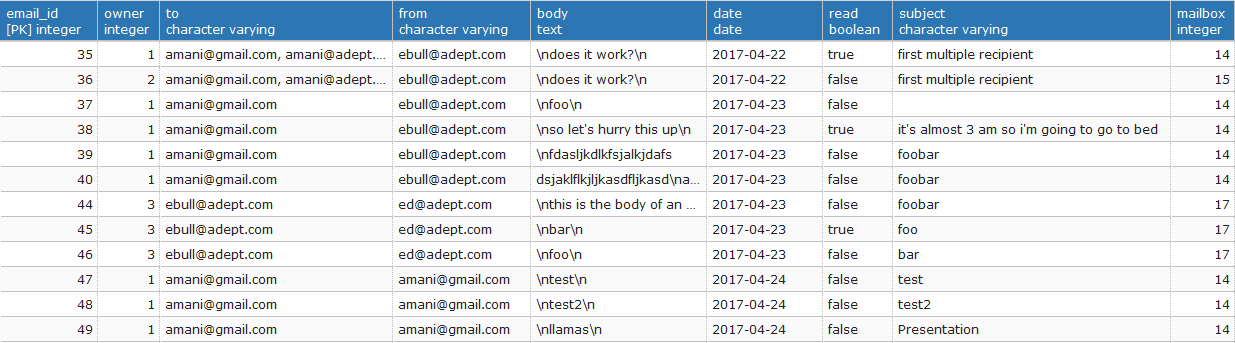


Figure 0.0 : Sample data for the emails table



SQLite

Figure 0.0: Relational diagram for the PostgreSQL database ‘localdb’

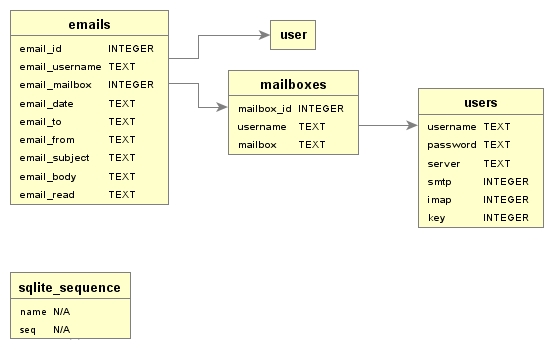


Figure 0.0 : Sample data for the users table

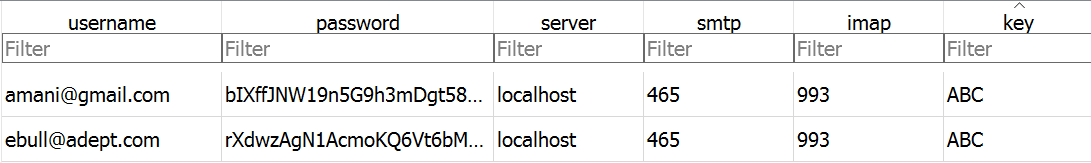


Figure 0.0 : Sample data for the emails table

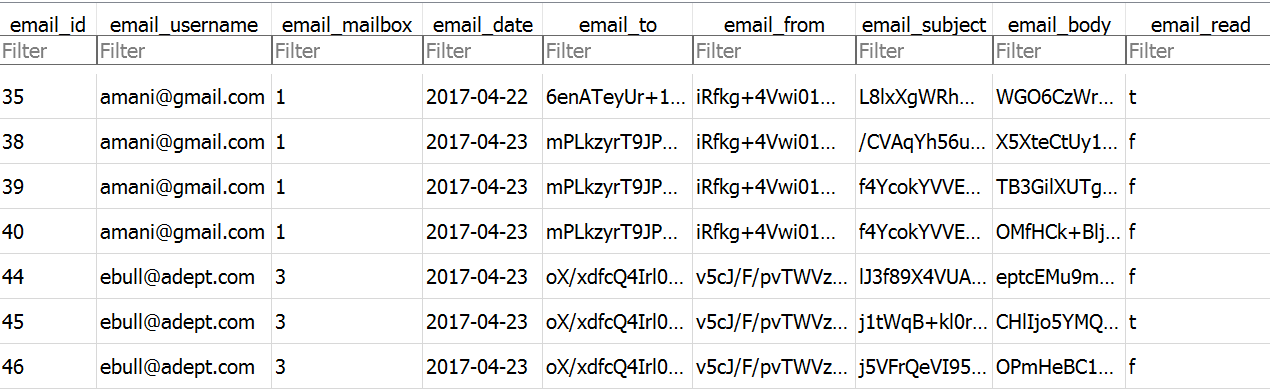


Figure 0.0 : Sample data for the mailboxes table

