

# **CS157A Database Project**

## **A Web Application for Study Materials**

Team 29

Jan Rodriguez

Aaron Smith

Rabia Mohiuddin

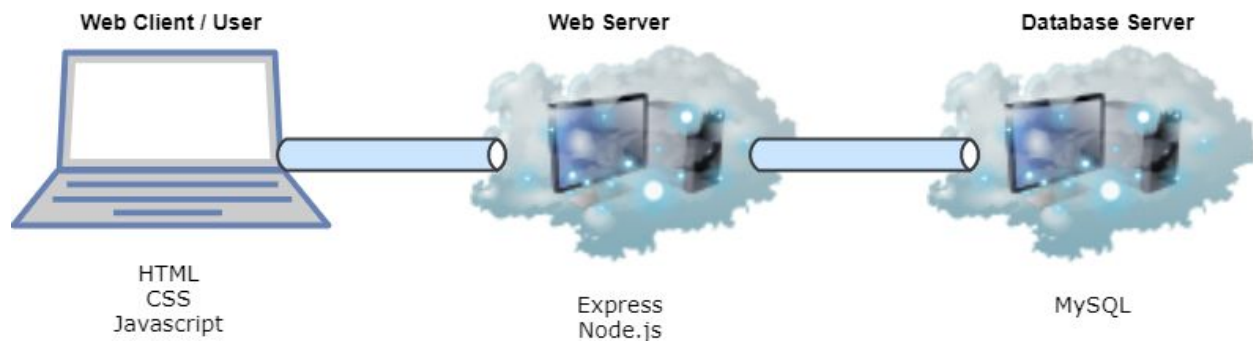
## 1. Project Description

- For our project, we will be making a web application that allows users to easily create, use, and organize study materials. These study materials will include tools like flashcards and quizzes.
- The goal of this project is to use a database management system to create something useful and powerful. As a result of this, team members can expect to learn more about using databases to create useful applications. This is a skill useful to all who will work with software and databases. A secondary goal is to create an application that enables users to create new and engaging methods to review content. Perhaps this application can even be used to study for CS157A quizzes and exams.
- The stakeholders of this project will primarily comprise of students across all levels of education who are looking for different methods to study.

## 2. System Environment

- The program is meant to run within a client's browser. The GUI interface will thus be made using HTML, CSS, and Javascript. This will then connect to a web server made using Express and Node.js.

This web server will provide the client with the code necessary to run the program, including Javascript code that will assist in delivering content from the database. This server will thus be responsible for connecting the client with the database. The database will be managed using MySQL version 8. This can all be seen in the following diagram.



### 3. Functional Requirements

#### How users will access the system:

- Users and stakeholders will be able to access our system through using any popular browser and typing in the url link to their respective address bar. Once on the website, they will be able to either browse as an unregistered user with only read capabilities or login with credentials to acquire registered user and admin privileges to respective study material.

### Functions:

- Sign up - Users will be able to become registered users by filling out all required fields in the account creation page.
- Login - Users will need to authenticate themselves through a username and password to acquire registered user/admin privileges.
- View Folders/Accounts/StudySets - Registered users will be able to view various pages related to all the folders, accounts, and study sets in the database. These pages give detailed information, like what folders a user contributes to, what studysets a folder contains, the owner of a studyset/folder, what users a user follow, etc. These details can be seen in the screenshots in the implementation section.
- Follow/Unfollow users: Users can follow and unfollow other users.
- Create/Delete sets of flashcards - Registered users will be given the ability to make/rename sets of flashcards. The user that creates the set will have admin access to the set and the flashcards within.
- Create/Read/Update/Destroy Flashcards - Users with access will be given CRUD functionality for flash cards based on their privilege level. Each flashcard will have text or images on either side of the

flashcard to denote key terms, questions and concepts. The GUI implementation will closely resemble that of flashcards in real life.

- Generate practice test from flashcards - Users will be given the functionality of simulating a practice test by randomly shuffling their flashcards order. The GUI implementation will consist of showing one side of the flashcard and allowing the user to flip over to the other side of the flashcard to see if they answered the question right.

#### 4. Non- Functional Issues

- Graphical User Interface (GUI) - The Graphical User Interface will be hosted on a web application using HTML,CSS and javascript that the user will be able to directly interact with, including editing and viewing the terms/definitions in different forms to enhance the learning experience.

Our GUI will feature many of the generally accepted Graphical User Interface Design principles including but not limited to, clarity, comprehensibility, configurability, consistency, control, simplicity, and efficiency. Clarity and comprehensibility will be seen through user controlled functions, visual elements, and text that will allow the user to implicitly know what actions to do next. Configurability will

be demonstrated through the nature of the application as the user will be able to create their own study set with their own terms and diagrams. With these study terms, the user will be able to use a variety of learning methods and tools to study the set. The User Interface will be simple and consistent across screens with a simple, similar look and use such that when the same function is invoked multiple times, it will yield the same result. With simplicity, we will incorporate efficiency such that the user does not need to go through multiple actions or screens in order to complete a task. Lastly, the user will control all interactions with the interface in a manner that all actions will result from a user request in a quick manner.

- Security - Each user account will be protected by a username and password created by the user. All data and study sets will be associated with the respective user with access controls specified in the next section. All user account information will be stored on the server and logins will only take place when the user provides an email and password that exists for an associated account in the application. An encrypted https connection will be implemented between the user

and server. Passwords will not be directly stored in the database but rather passwords hashes.

- Access Control - Users will have view access to all publicly available flashcard sets regardless if the user is logged in or not. Write and edit access will be limited to the author of the study set using login credentials to verify so. From a high level, the user that creates the study set is the administrator of the study set while everyone else is a consumer and/or viewer of it.

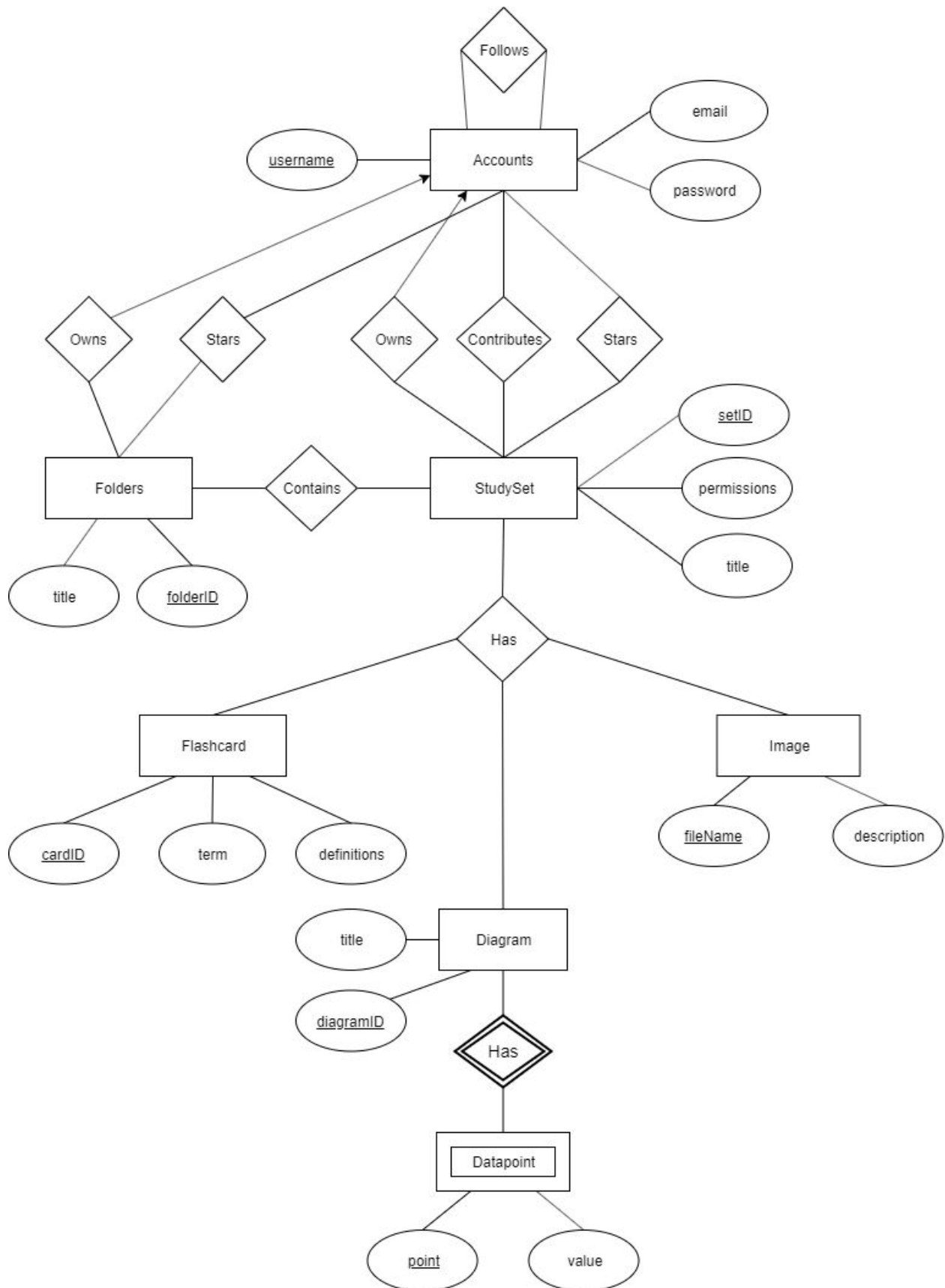
## 5. ERD Design

- Details about tables in the diagram:
  - Accounts represents user accounts which are identified by username and have attributes for email and password.
  - Contains is a relationship mapping StudySets to Folders. It represents all studysets that a folder contains. A folder can contain multiple study sets, and a study set can be contained in various folders.
  - Contributes is relationship that represents study sets that a user has contributed to, but doesn't necessarily own. Multiple accounts can contribute to multiple StudySets, and each study set can have many contributors.
  - Datapoint is a weak entity that represents points on a diagram. It is associated with a particular diagram and identified through its point and has a value.
  - Diagram is an entity that is identified by diagramID. It has an image and can have multiple Datapoints.

- Flashcard is an entity with a term and definitions, the two sides of a flashcard, identified by the cardID.
- Folders is an entity that represents folders. Each folder can contain StudySets and is identified by folderID.
- Follows is a relationship matching accounts to itself. An account can follow other accounts to see their study sets and any other data associated with the account.
- Has is a relationship representing what a StudySet has: any combination of flashcards, diagrams, and images.
- Image is an entity that simply represents an image that a user would upload to be contained in a study set identified with its fileName and has a description.
- Owns is a relationship matching Accounts to either Folders or StudySets. Since each study set and each folder can only have one owner, this relationship is modeled in the database as an extra attribute in the Folders and StudySets tables.
- Stars is a relationship matching Accounts to either Folders or Studysets. Each user can highlight or 'star' a folder/studysset in order to keep track of it.
- StudySet is an entity that represents study sets. Each studySet has Flashcards, Images, and/or Diagrams. Study sets are uniquely identified by setID.

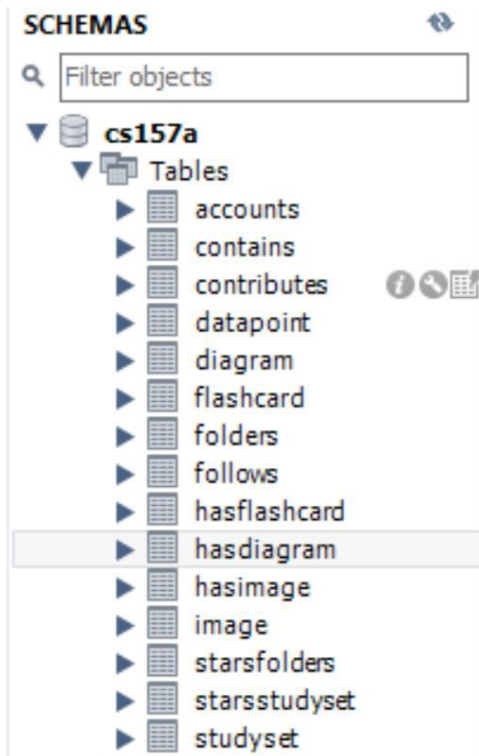
- Diagram on next page





## Database Tables

Database schema



*Entity* **accounts**

username	email	password
aaronsmith	aaron.smtih@sjsu.edu	sha1\$de40904e\$1...
as	testing@gmail.com	sha1\$de40904e\$1...
bones	bones@sjsu.edu	sha1\$de40904e\$1...
data	data@sjsu.edu	sha1\$de40904e\$1...
dudeman	dudeman@sjsu.edu	sha1\$86217096\$1...
jadzia	jadzia@sjsu.edu	sha1\$de40904e\$1...
jimkirk	mrkirk@sjsu.edu	sha1\$de40904e\$1...
picard	picard@sjsu.edu	sha1\$de40904e\$1...
redshirt	redshirt@sjsu.edu	sha1\$de40904e\$1...
spock	spock@sjsu.edu	sha1\$de40904e\$1...
test	test@gmail.com	sha1\$adaf2950\$1...
test1	test1@email.com	sha1\$de40904e\$1...
test12	test12@gmail.com	sha1\$de40904e\$1...
test123	test123@gmail.c	sha1\$de40904e\$1...
testing	something@gmail.com	sha1\$de40904e\$1...
testinga...	testingagain@gmail....	sha1\$3a78a2e3\$1...

### Entity contains

folderID	setID
1	1
1	2
2	3
2	4
3	1
3	2
4	3
4	4
5	1
5	2
5	5
6	1
6	2
6	5
7	3

### Entity contributes

username	setID
bones	4
data	1
data	3
data	5
jimkirk	2
picard	2
picard	3
spock	1
spock	2
spock	3
spock	4
spock	5
uhura	4
worf	2
worf	6
worf	7

### Entity datapoint

point	diagramID	value
1	10	12
1	1	123
11	1	13
12	2	213
2	2	234
14	4	32
3	3	345
15	5	434
4	4	456
13	3	553
5	5	567
6	6	678
7	7	789
8	8	890
9	9	901

### Entity diagram

diagramID	title
1	merge sort
2	insertion sort
3	quick sort
4	heap sort
5	counting sort
6	selection sort
7	bubble sort
8	depth first se...
9	breadth first ...
10	graph example
11	example1
12	example2
13	example3
14	example4
15	example5

### *Entity flashcard*

cardID	term	definitions
1	ad interim	temporarily
2	ante meridiem	before midday
3	et alii	and others
4	et caetera	and the others
5	exempli gratia	for example
6	inter alia	among other t...
7	id est	that is
8	per annum	through a year
9	post meridiem	after midday
10	pro tempore	temporarily
11	post scriptum	after what ha...
12	quaque die	every day
13	quod erat dem...	which was to ...
14	videlicet	namely
15	Latin	a language

### *Entity folders*

folderID	title	owner
1	CS146	aaronsmith
2	CS157A	aaronsmith
3	CS151	aaronsmith
4	CS152	aaronsmith
5	CS166	aaronsmith
6	CS154	aaronsmith
7	CS149	aaronsmith
8	CS158A	aaronsmith
9	CS185C	aaronsmith
10	CS174	aaronsmith
11	math	spock
12	science	spock
13	logic	spock
14	technology	spock
15	literature	spock

### Entity follows

username1	username2
aaronsmith	as
aaronsmith	jadzia
aaronsmith	picard
bones	aaronsmith
bones	jimkirk
bones	spock
jimkirk	aaronsmith
jimkirk	bones
jimkirk	picard
jimkirk	spock
spock	aaronsmith
spock	bones
spock	jimkirk
worf	data
worf	picard
worf	spock

### Entity hasflashcard

setID	cardID
1	1
1	2
1	3
1	4
1	5
1	6
1	7
1	8
1	9
1	10
1	11
1	12
1	13
1	14
1	15

### Entity hasdiagram

setID	diagramID
1	11
2	12
3	13
4	14
5	15
10	1
10	2
10	3
10	4
10	5
10	6
10	7
10	8
10	9
10	10

### Entity hasimage

setID	fileName
4	test1.png
4	test2.png
4	test3.png
4	test4.png
4	test5.png
5	JPG_1.jpg
5	JPG_2.jpg
5	JPG_3.jpg
5	JPG_4.jpg
5	JPG_5.jpg
10	PNG_1....
10	PNG_2....
10	PNG_3....
10	PNG_4....
10	PNG_5....



*Entity image*

fileName	description
JPG_1.jpg	enterprise (NX-01)
JPG_2.jpg	enterprise (NCC-1701)
JPG_3.jpg	enterprise (NCC-1701-A)
JPG_4.jpg	enterprise (NCC-1701-B)
JPG_5.jpg	enterprise (NCC-1701-C)
PNG_1....	merge sort
PNG_2....	heap sort
PNG_3....	insertion sort
PNG_4....	quick sort
PNG_5....	counting sort
test1.png	image example
test2.png	image example
test3.png	image example
test4.png	image example
test5.png	image example

*Entity starsfolder*

username	folderID
bones	1
bones	3
jimkirk	2
jimkirk	5
picard	2
picard	3
picard	4
picard	5
spock	1
spock	4
worf	11
worf	12
worf	13
worf	14
worf	15

*Entity starsstudysset*

username	setID
bones	2
bones	3
picard	1
picard	2
picard	3
spock	1
spock	4
spock	5
uhura	1
worf	1
worf	2
worf	3
worf	4
worf	5
worf	6

*Entity studysset*

setID	Title	permissions	owner
1	Latin Terms	public	aaronsmith
2	Intevieew Prep	public	aaronsmith
3	Algorithms	public	aaronsmith
4	Data Structures	public	aaronsmith
5	Star Trek	publc	aaronsmith
6	CS 157A	private	aaronsmith
7	CS 185C	private	aaronsmith
8	CS 147	private	aaronsmith
9	CS 149	private	aaronsmith
10	CS 152	private	aaronsmith
11	test	public	aaronsmith
12	test	public	aaronsmith
13	test	public	aaronsmith
6635	helloooooo	public	aaronsmith
6699	Demo project	public	aaronsmith
7873	test	public	aaronsmith

## 6. Project Implementation

Our database application was implemented through the use of hosting our database on MySQL workbench. In order to work on the same data set as a team, we exported our MySQL workbench data to a data dump and imported it into our individual workbenches. In order to start the website, clone the repo and go into the the repo directory. Copy the database export from the Database Data export into own MySQL workbench.

Then run: `node app.js`

Then go to: `http://localhost:1337`

- Sign up

GUI - The login page has a sign up option. When pressed, the user is asked to input an email, username, and password for creating an account. The email is checked for validity and the username is checked in the account to make sure it isn't already taken. The password is also flagged if it isn't 5 characters long.

# Welcome to StudySets!

aaronsmith@email.com

✓

aaronsmith

✗

.....

✓

repeat password

✗

SIGN UP

NodeJS code -

For checking if username isn't already taken:

```
// Checks if the username is in the database. If it isn't, returns "valid" to the client.
app.post("/checkIfUsernameValid", function(req, res) {
  let query =
    'SELECT * FROM accounts WHERE username = "' + req.body.username + '"';
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in checkIfUsernameValid query");
    } else {
      if (result.length === 1) {
        // If this is true, then our query returned a row in the database
        res.send("invalid");
      } else {
        // Else username not in the database
        res.send("valid");
      }
    }
  });
});
```

For creating a new user account:



```
// Sign up for the platform. Check data and database so that a valid account can be created.
app.post("/signup", function(req, res) {
  let query =
    'INSERT INTO accounts (username, email, password) VALUES(' +
    req.body.username +
    ', ' +
    req.body.email +
    ', ' +
    passwordHash.generate(req.body.password) +
    ');';
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in signup query");
      console.log(error);
    } else {
      res.send("valid");
    }
  });
});
```

- Login

GUI - User can input username and password to login. The database is then queried to check if the username/password combination is correct, and logs them in if it is.

# Welcome to StudySets!

aaronsmith

....|

LOG IN

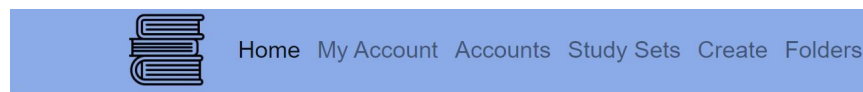
OR SIGN UP

NodeJS code - The code that handles this:

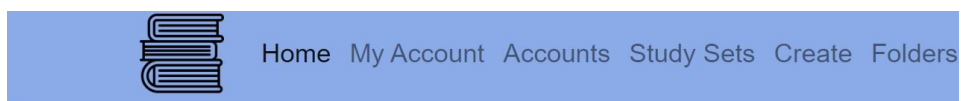
```
// Login to the platform. Check database for valid username and password.
app.post("/login", function(req, res) {
  let query =
    'SELECT * FROM accounts WHERE username = \'' + req.body.username + '\'';
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in login query");
    } else {
      if (result.length === 1 && passwordHash.verify(req.body.password, result[0].password)) {
        // If this is true, then our query returned a row in the database, i.e. the username/password combination is correct.
        console.log("Logging in as " + req.body.username);
        currentuser = req.body.username;
        res.send("valid");
      } else {
        // Else username/password combination not in the database.
        res.send("invalid");
      }
    }
  });
});
```

- View Folders

GUI - The folders page shows all the folders that exist in the database. After clicking on a folder, the page will show all the studysets contained in the folder.



folder: CS146  
folder: CS157A  
folder: CS151  
folder: CS152  
folder: CS166  
folder: CS154  
folder: CS149  
folder: CS158A  
folder: CS185C  
folder: CS174  
folder: math  
folder: science  
folder: logic  
folder: technology  
folder: literature



## Folder: CS146

Studyset: Interview Prep, owned by: aaronsmith

Studyset: Latin Terms, owned by: aaronsmith

[back to folders](#)

## NodeJS code -

```
app.get("/studyset", function(req, res) {
  database.query("SELECT * FROM StudySet", function(error, result) {
    if (error) {
      console.log("Error in query");
    } else {
      res.render("study-set", { rows: result });
    }
  });
});
```

```
// View folder details
app.post("/folderdata", function(req, res) {
  console.log("Fetching folder data for: " + req.body.foldername);
  let query = `SELECT title, owner FROM studyset, contains WHERE contains.setID = studyset.setID AND contains.folderid =
    + `(SELECT folderID FROM folders WHERE title = "` + req.body.foldername + `")`;
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in folder query");
      console.log(error);
    } else {
      res.send(JSON.stringify(result));
    }
  });
});
```

- View StudySets

GUI - The study sets page shows all the study sets registered in the database.

After clicking on one, the page will show more details about the study set, including who the owner is.



**studysset:** [Latin Terms](#)

**studysset:** [Inteview Prep](#)

**studysset:** [Algorithms](#)

**studysset:** [Data Structures](#)

**studysset:** [Star Trek](#)

**studysset:** [CS 157A](#)

**studysset:** [CS 185C](#)

**studysset:** [CS 147](#)

**studysset:** [CS 149](#)

**studysset:** [CS 152](#)

**studysset:** [test](#)

**studysset:** [test](#)

**studysset:** [test](#)

**studysset:** [hellooooo](#)

**studysset:** [Demo project](#)

**studysset:** [test](#)



# Studysset : Latin Terms

Owner : aaronsmith

[View Flashcards](#)

[back to studyssets](#)

NodeJS code -

```

app.get("/studyset", function(req, res) {
  database.query("SELECT * FROM StudySet", function(error, result) {
    if (error) {
      console.log("Error in query");
    } else {
      res.render("study-set", { rows: result });
    }
  });
});

```

```

// View studysetdata
app.post("/studysetdata", function(req, res) {
  console.log("Fetching study set data for: " + req.body.studysetName);
  let query = `SELECT * FROM studyset WHERE Title = "` + req.body.studysetName + `";`;
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in studyset query");
      console.log(error);
    } else {
      res.send(JSON.stringify(result));
    }
  });
});

```

- View Accounts

GUI - The accounts page shows all the accounts registered in the database.

After clicking on an account, the page will show the details of that account.



[Home](#) [My Account](#) [Accounts](#) [Study Sets](#) [Create](#)

User: [as](#)  
User: [bones](#)  
User: [data](#)  
User: [dudeman](#)  
User: [jadzia](#)  
User: [jimkirk](#)  
User: [picard](#)  
User: [redshirt](#)  
User: [spock](#)  
User: [test](#)  
User: [test1](#)  
User: [test12](#)  
User: [test123](#)  
User: [testing](#)  
User: [testingagain](#)  
User: [uhura](#)  
User: [user](#)  
User: [user1](#)  
User: [worf](#)



[Home](#) [My Account](#) [Accounts](#) [Study Sets](#) [Create](#)

## Account: [jimkirk](#)

Owns:

Contributes to:

[studyset: Interview Prep](#)

Stars:

[folder: CS166](#)  
[folder: CS157A](#)

Follows:

[user: spock](#)  
[user: picard](#)  
[user: bones](#)  
[user: aaronsmith](#)

[back to accounts](#)

[follow jimkirk](#)

### NodeJS code -

- Follow/Unfollow Account

GUI - User is given an option to follow/unfollow another user when on their profile page (the previous Account example shows the follow button towards the bottom). Here's an unfollow button example:



# Account: bones

Owns:

Contributes to:

studyset: Data Structures

Stars:

folder: CS151

folder: CS146

studyset: Algorithms

studyset: Interview Prep

Follows:

user: spock

user: jimkirk

user: aaronsmith

[back to accounts](#)

[unfollow bones](#)

NodeJS code -

```
// for following an account
app.post("/followaccount", function(req, res) {
  let query = `INSERT INTO follows (username1, username2) VALUES(` + currentuser + `,` + req.body.username + `)`;
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in followaccount query");
    } else {
      console.log(currentuser + " is now following " + req.body.username);
      res.send("done");
    }
  });
});

// for unfollowing an account
app.post("/unfollowaccount", function(req, res) {
  let query = `DELETE FROM follows WHERE username1 = ` + currentuser + ` AND username2 = ` + req.body.username + ``;
  database.query(query, function(error, result) {
    if (error) {
      console.log("Error in unfollowaccount query");
    } else {
      console.log(currentuser + " is no longer following " + req.body.username);
      res.send("done");
    }
  });
});
```

Database result -

Before clicking the follow jimkirk button, the follows table looks like:

username1	username2
aaronsmith	as
aaronsmith	jadzia
aaronsmith	picard
bones	aaronsmith
bones	jimkirk
bones	spock
jimkirk	aaronsmith
jimkirk	bones
jimkirk	picard
jimkirk	spock
spock	aaronsmith
spock	bones
spock	jimkirk
worf	data
worf	picard
worf	spock

After clicking follow, it looks like (notice the added row with 'aaronsmith'):

username1	username2
aaronsmith	as
aaronsmith	jadzia
aaronsmith	jimkirk
aaronsmith	picard
bones	aaronsmith
bones	jimkirk
bones	spock
jimkirk	aaronsmith
jimkirk	bones
jimkirk	picard
jimkirk	spock
spock	aaronsmith
spock	bones
spock	jimkirk
worf	data

After clicking unfollow, the database returns to its previous state.





- Create Set



GUI - A user can enter the set name as well as multiple flashcards

containing terms and definitions, saving the set as a whole. They can add or delete rows to enlarge or decrease the study set, and once complete can hit the “save” icon to write to the database.

## NEW STUDY SET

Name:

Term	Definition	Remove
CS 157A	Databases	
BCNF	strengthens 3NF	
A+	Our project	
		

NodeJS code -

```

42 app.post("/createStudySet", function(req, res) {
43   let checkQuery =
44     'SELECT * FROM studysset WHERE setID = "' + req.body.setID + '";';
45
46   database.query(checkQuery, function(error, result) {
47     if (error) {
48       console.log("Error in check studysset query");
49       console.log(error);
50     } else {
51       let query =
52         'INSERT INTO studysset (setID, Title, permissions, owner) VALUES("' +
53         req.body.setID +
54         '","' +
55         req.body.setName +
56         '","' +
57         "public" +
58         '","' +
59         currentUser +
60         '");';
61       database.query(query, function(error, result) {
62         if (error) {
63           console.log("Error in insert create query");
64           console.log(error);
65         } else {
66           res.send("valid");
67         }
68       });
69     }
70   });
71 });
72

```

Database Result -

Before add

```
mysql> SELECT * FROM cs157a.studyset;
```

setID	Title	permissions	owner
1	Latin Terms	public	aaronsmith
2	Inteview Prep	public	aaronsmith
3	Algorithms	public	aaronsmith
4	Data Structures	public	aaronsmith
5	Star Trek	publc	aaronsmith
6	CS 157A	private	aaronsmith
7	CS 185C	private	aaronsmith
8	CS 147	private	aaronsmith
9	CS 149	private	aaronsmith
10	CS 152	private	aaronsmith
359	12345	public	aaronsmith
409	ewww	public	aaronsmith
1022	[object Object]	public	aaronsmith
1209	sfjd	public	aaronsmith
1327	[object Object]	public	aaronsmith
1610	okkkk	public	aaronsmith
1778	ewww	public	aaronsmith
2520	sample	public	aaronsmith
3901	6789	public	aaronsmith
4243	erike	public	aaronsmith
4257	were	public	aaronsmith
4468	393993	public	aaronsmith
4573	sfjd	public	aaronsmith
6019	[object Object]	public	aaronsmith
6148	393993	public	aaronsmith
6405	test1	public	aaronsmith
7094	null	public	aaronsmith
7209	[object Object]	public	aaronsmith
7684	fwss	public	aaronsmith
7999	test1	public	aaronsmith
8301	were	public	aaronsmith
9618	fwss	public	aaronsmith
9686	[object Object]	public	aaronsmith

```
33 rows in set (0.00 sec)
```

After add



```
[mysql> SELECT * FROM cs157a.studyset;
```

setID	Title	permissions	owner
1	Latin Terms	public	aaronsmith
2	Intevieu Prep	public	aaronsmith
3	Algorithms	public	aaronsmith
4	Data Structures	public	aaronsmith
5	Star Trek	publc	aaronsmith
6	CS 157A	private	aaronsmith
7	CS 185C	private	aaronsmith
8	CS 147	private	aaronsmith
9	CS 149	private	aaronsmith
10	CS 152	private	aaronsmith
359	12345	public	aaronsmith
409	ewww	public	aaronsmith
1022	[object Object]	public	aaronsmith
1209	sfjd	public	aaronsmith
1327	[object Object]	public	aaronsmith
1610	okkkk	public	aaronsmith
1778	ewww	public	aaronsmith
2520	sample	public	aaronsmith
3901	6789	public	aaronsmith
4243	erike	public	aaronsmith
4257	were	public	aaronsmith
4468	393993	public	aaronsmith
4573	sfjd	public	aaronsmith
6019	[object Object]	public	aaronsmith
6148	393993	public	aaronsmith
6342	My new study set	public	aaronsmith
6405	test1	public	aaronsmith
7094	null	public	aaronsmith
7209	[object Object]	public	aaronsmith
7684	fwss	public	aaronsmith
7999	test1	public	aaronsmith
8301	were	public	aaronsmith
9618	fwss	public	aaronsmith
9686	[object Object]	public	aaronsmith

```
34 rows in set (0.00 sec)
```

- Create Flashcard

GUI - A user can enter a term and definition and press the “Add card”

button to submit the form elements and update the database.

## Add new card

Node JS code -

```
app.post("/add-card", function(req, res) {
  let query = `INSERT INTO flashcard (term, definitions) VALUES("` + req.body.cardTerm + `", "` + req.body.cardDefinition + `")`;
  database.query(query, function(error, result) {
    if (error) {
      console.log(error);
    } else {
      console.log("card ID was added");
      res.redirect("/edit-flashcards")
    }
  });
});
```

- Read Flashcard

GUI - The user can hover over the code to flip between the term and definition.



ante meridiem

Node JS code -

```
app.get("/flashcards", function(req, res) {  
  database.query("SELECT * FROM Flashcard", function(error, result) {  
    if (error) {  
      console.log("Error in query");  
    } else {  
      res.render("flashcards", { rows: result });  
      console.log(result);  
    }  
  });  
});
```

- Update Flashcard

GUI - Users can enter the card ID and a new term and definition to update the current existing card

## Edit existing card

Node JS code -

```
app.post("/edit-card", function(req, res) {  
  console.log(req.body.cardID);  
  let query = `UPDATE Flashcard SET term="" +req.body.cardTerm+"`,definitions= ""+req.body.cardDefinition+" WHERE cardID = "" + req.body.cardID + "`";  
  database.query(query, function(error, result) {  
    if (error) {  
      console.log(error);  
    } else {  
      console.log("card ID was updated");  
      res.redirect("/edit-flashcards")  
    }  
  });  
});
```

- Delete Flashcard

GUI - Users can enter in the card ID to delete the card from the database

## Delete card

Node JS code -

```
app.post("/del-card", function(req, res) {  
  console.log(req.body.cardID);  
  let query = `DELETE FROM Flashcard WHERE cardID = "` + req.body.cardID + `";`;   
  database.query(query, function(error, result) {  
    if (error) {  
      console.log(error);  
    } else {  
      console.log("card ID of: " + req.body.cardID + " was deleted");  
      res.redirect("/edit-flashcards")  
    }  
  });  
});
```

- Generate Practice test from flashcards

GUI - The navbar button on the top can be pressed to generate a practice test from the existing flash cards from the users

Practice Test

Node JS code -

```
app.get("/practice-test", function(req, res) {
  database.query("SELECT * FROM Flashcard ORDER BY RAND()", function(
    error,
    result
  ) {
    if (error) {
      console.log("Error in query");
    } else {
      res.render("practice", { rows: result });
      console.log(result);
    }
  });
});
```

## 7. Project conclusion

### Personal Statements

- Jan Rodriguez

“The main takeaways that I acquired from working on this project is mainly derived from the ability to effectively learn and apply new information. Especially with technical skills such as full stack development, it takes a lot of effort and energy and time to learn properly.”

- Rabia Mohiuddin

“Building this database web application allowed me to strengthen my knowledge of how databases are incorporated in the industry. You may know how to write SQL statements and manipulate a database, but to incorporate it with user requests using GET and POST requests is a real world application that is essential knowledge for our future careers in a booming tech industry. More than

enhancing my skills in full-stack development, adjusting and strengthening my knowledge in new libraries and languages built my confidence in creating database web applications on my own.”

- Aaron Smith

“Through this project, I learned more about web technologies by using HTML, CSS, Javascript, Express, NodeJS. I also learned more about creating a database with good tables and creating queries to access/modify it. I learned that database projects are difficult and time consuming, and that database design needs to be well thought out.”

### Further Improvements

With anything, improvements can always be made. For our database web application for study materials, StudySets, several enhancements and revisions can be made, including but not limited to the following items.

Currently, users can view folders with study sets within them, however to further enhance this feature, functionality to create and delete folders would be added. Additionally, a feature to add and remove study sets from folders would be another improvement to be made.