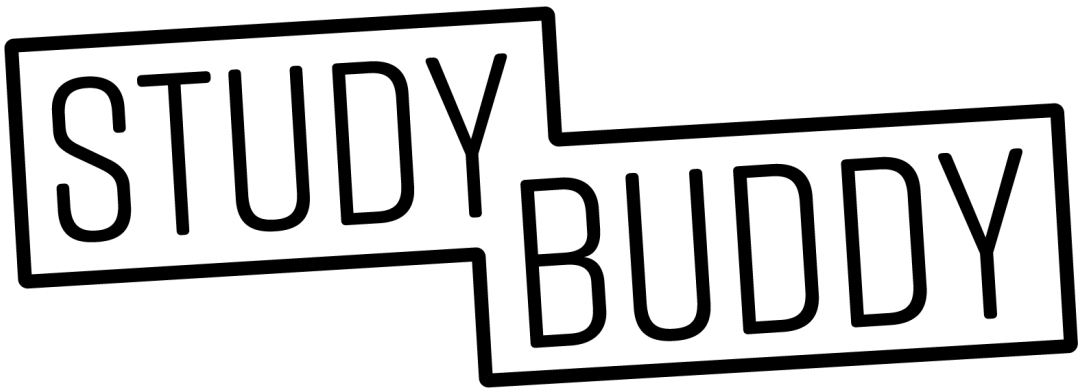

Pauline Fagerberg <fapa1696@student.ju.se>

Anna Darner <daan1696@student.ju.se>

A Project Work in Client-Server Communication

Jönköping University 2018

Table of Contents

Introduction

Studybuddy is an app which makes it easy for students to find study partners and makes them study more. If you get a notification that your friend is already preparing for the exam or finishing up the lab, would not you feel more inclined to start too? It will not only notify you that your friend is studying but also send a location to where so you can easily join them. It is also optional to add a message to your status so your friends can see what you are studying for. The status will then remain on a map for 2 hours for your friends to see before it is automatically removed.

To start up you will create an account, either through Facebook or mail. You can then add friends in multiple ways, 'add nearby' for example which will use bluetooth to connect with others who uses the function. Other alternatives for this is through Facebook friends or usernames.

When all of your friends get a notification that you are studying they will be able to answer if they are interested in joining. The figure below (figure 1) shows a mockup of how it might look when a user gets a notification that a friend is studying.

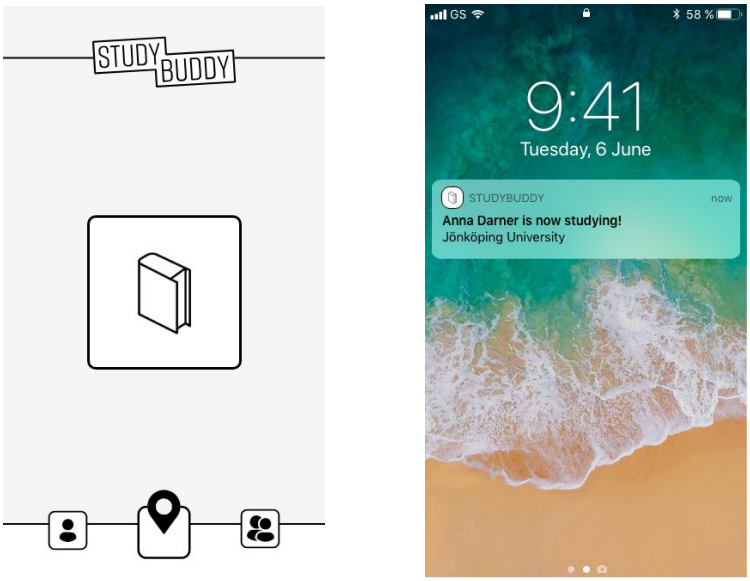

Figure 1. Mockups of the Studybuddy app.

1. Architecture

Study Buddy consist of a backend and frontend application which is shown in the figure below (figure 2.).

The frontend application is quite simple and will be developed for smartphones. The main page is primarily to notify your friends and a map to see who and where they are studying. Two more pages will be available, one for the account and one for friends.

The backend application will implement REST API so the client (the frontend application) can apply CRUD (Create, Read, Update, Delete) to the resources in the database to for example create a new status, create or update an account etc. This will be specified further down in this document.

An SQLite database will be used to store the resources on the same server.

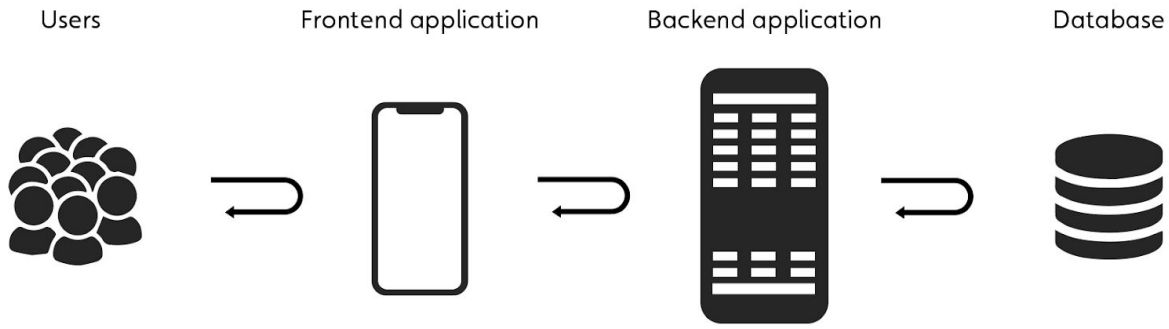

Figure 2. Architectural overview of the app.

2. Resources

The figure below (figure 3) shows an database scheme of the resources explained further down.

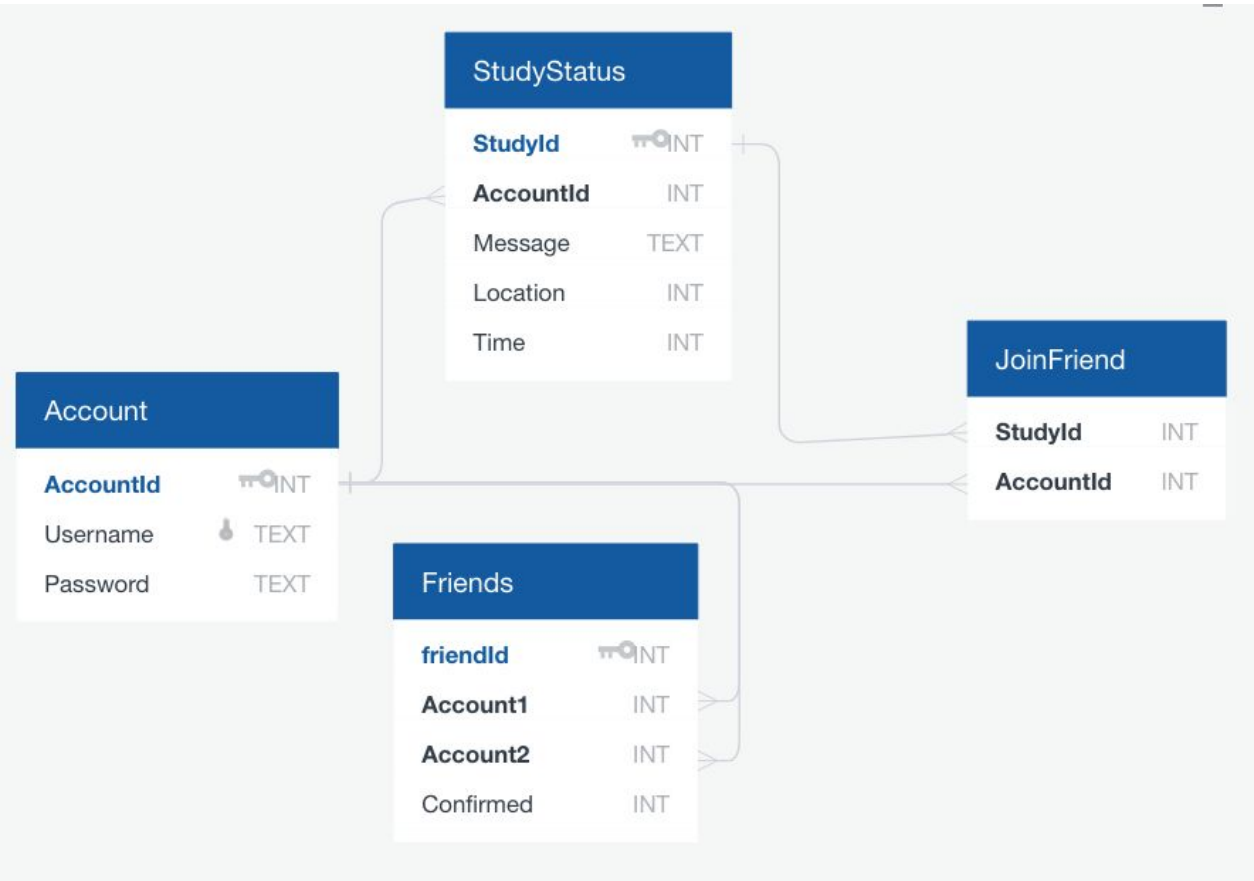

Figure 3. Database scheme of the resources.

2.1 StudyStatus

This resource is created when a user chooses to press the “study button”. The id, account and location is added automatically and the message is optional for the user to add. The resource is automatically removed after a certain amount of time (2 hours) since it is not relevant anymore.

- StudyId contains a unique identifier
- AccountId contains the Id of the account that the status belongs to
- Message contains the information about the StudyStatus. Can consist of up to 250 characters.
- Location contains the GPS coordinates of the location

- Time contains the time the status was created

2.2 Account

- AccountId is a unique identifier
- Username contains the chosen name of the user
- Password contains the password (hashed)

2.3 Friends

This resource connects different accounts to each other. It does not matter what account is account1 or account 2.

- FriendId is a unique identifier (used primarily when deleting a friendship)
- Account1 contains one of the accounts Ids
- Account2 contains the other accounts Id
- Confirmed contains an integer 1 or 0 (true or false) to see if the other account has confirmed the request

2.4 JoinFriend

This resource is for the function to “join” another user who is studying. The user who you are joining will then get a notification that you are on your way. When the join request is sent the user who posted the studyStatus will receive a notification with the message “ThisUser is joining!”.

- StudyId contains the Id of the chosen status
- AccountId contains the Id of the user who wants to join

3. REST API

Common details

In this chapter the specification for the REST API follows. The client can apply CRUD (Create, Read, Update, Delete) operations on the resources. Every operation is specified in detail in each subchapter below.

Status codes

Status Code	Description
`200 OK`	Standard response for successful HTTP requests.
`201 Created`	The request has been fulfilled, resulting in the creation of a new resource.
`400 Bad request`	If a request to update/create a new resource fails due to validation errors a response with the status code 400 (Bad Request) is sent back together with an array of error codes describing what is wrong.
`401 Unauthorized`	Similar to *403 Forbidden*, but specifically for use when authentication is required and has failed or has not yet been provided.
`404 Not found`	The requested resource could not be found but may be available in the future.
`422 Unprocessable Entity`	If a resource sent to the server is malformed, a response with the status code 422 will be sent back.
`500 Internal Server Error`	A generic error message, given when an unexpected condition was encountered and no more specific message is suitable.
`204 No Content`	The request was successful. A status used for example when a resource is updated or deleted.

3.1 Creating a new Account

Sends an HTTP request, displayed in Code Box 1.

```
POST /accounts HTTP/1.1
Host: localhost:6000
Accept: application/json
Content-Type: application/json
Content-Length: 68
{
     "username": "The username...",
     "password": "The password..."
}
```

Code Box 1, HTTP request to create a new Account

Status code 422 is sent back if the content of the request is malformed. Shown below in Code Box 2.

```
HTTP/1.1 422 Unprocessable Entity
```

Code Box 2, HTTP response if the new Account is malformed.

When the request is correct and an account is created, a response with the status code 201 is returned. Shown in Code Box 3.

```
HTTP/1.1 201 Created
Location: http://localhost:6000/accounts/1
```

Code Box 3, HTTP response when an Account is created.

If the content of the new Account request does not pass the validation. The status code 400 is returned along with the errors found, shown in Code Box 4. See Table 1 for the possible errors.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
content-Length: 33

["error1", "error2", ...]
```

Code Box 4, HTTP response if the content does not pass the validation.

Error	Description
usernameTooShort	**username** in the resource is too short
usernameTooLong	**username** in the resource is too long
usernameNotUnique	**username** is already used
usernameInvlChar	**username** contains invalid characters

Table 1, Errors for Account validation.

3.2 Sign in

To login to an account (to get an access token and an ID token), send an HTTP request like the one shown in Code Box 5 below.

```
POST /tokens HTTP/1.1
Host: localhost:6000
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Content-Length: 61

grant_type=password&username=theUsername&password=thePassword
```

Code Box 5, HTTP request to log in to an account

If the request was successful (the login succeeded) a response like the one in Code box 6 is sent back with the status code 200. . The id token should contain the claims sub (the user's account id) and preferred_username (the user's username).

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 92
```

```
{
      "access_token": "theAccessToken",
      "token_type": "Bearer",
      "id_token": "theIdToken"
}
```

Code Box 6, HTTP response if the Account is created

If the request was not successful and failed for some reason a response with the status code 400 will be sent back instead, along with an error code. This is shown in Code Box 7. Possible error codes is displayed in Table 2.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: 25
{
    "error": "theErrorCode"
}
```

Code Box 7, HTTP response if the login request fails.

Error code	Description
unsupportedGranType	grant_type has wrong value
invalidRequest	The request is malformed
invalidClient	Wrong password/username

Table 2, Error codes for failed login requests.

3.3 Post a StudyStatus

To post a studyStatus, send a HTTP request like the one in Code Box 8.

```
POST /studyStatus HTTP/1.1
```

```
Host: localhost:6000
Accept: application/json
Content-Type: application/json
Authorization: Bearer theAccessToken
Content-Length: 200

{
     "accountId": 1,
     "message": "Client-Server Communication
project",
     "time": 1234567898765,
     "location": 40.7127753,-74.0059728,

}
```

Code Box 8, HTTP request to post a StudyStatus.

If it is malformed, status code 422 is returned. Shown in Code Box 9.

```
HTTP/1.1 422 Unprocessable Entity
```

Code Box 9, HTTP response if the studyStatus is malformed.

When carried out correctly and the studyStatus is created and posted, a response with the status code 201 si returned. This is shown in Code Box 10 below.

```
HTTP/1.1 201 Created
Location: http://localhost:6000/studyStatus/321
```

Code Box 10, HTTP response when a studyStatus is created.

If the user is not authorized to create a studyStaus for the current account or if the access token is missing, the status code 401 is returned. Shown in Code Box 11.

```
HTTP/1.1 401 Unauthorized
```

Code Box 11, HTTP response if the user is not authorized to create a studyStatus.

If the post in the request does not pass validation, a response with the status code 400 is sent back including the error codes. Shown in Code Box 12 and Table 3 lists the possible errors.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
content-Length: 33

["error1", "error2", ...]
```

Code Box 12, HTTP response if the post does not pass the validation.

Error code	Description
accountNotFound	**accountID** does not contain the id of an existing account
messageTooLong	**message** in the resource is too long

Table 3, Errors for studyStatus validation.

3.4 Get all studyStatus

To retrieve all studyStatuses that are yours and your friends. Therefore you need too send a GET request for all your friends first (Code Box 31) after that your can send a HTTP request like the one in Code box 13. The '9' is the id of the users account so that the studyStatuses shown is only from friends.

```
GET /9/studyStatus HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code box 13, HTTP request to retrieve all studyStatuses.

A response is sent back with the status code 200 along with all studyStatus as shown in Code box 14 below, the response 404 will be sent back if the resource was not found.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 114
[{
     "studyId": 23,
     "accountId": 12,
     "message": "The message...",
     "location": 57.84528:12.96771,
     "time": 1538834664
}, {
   ...
},
...
]
```

Code box 14, HTTP response with all studyStatuses.

If no studyStatuses are found, the status code in Code Box 15 is sent back.

```
HTTP/1.1 404 Not found
```

Code Box 15, HTTP response if the studyStatus was not found.

3.5 Get a specific studyStatus

Send a request like shown in Code Box 16 to retrieve a studyStatus with a specific id. This retrieves the specific studyStatus from your friends studyStatuses. The '9' in this request stands for the users 'id'. The '123' stands for the 'studyId'.

```
GET /9/studyStatus/123 HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code box 16, HTTP request to retrieve studyStatus with specific id

A response with status code 200 will be sent back if the request was successful along with the studyStatus of that specific id. This is shown in Code Box 17.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 103
{
     "studyId": 23,
     "accountId": 12,
     "message": "The message...",
     "location": 57.84528:12.96771,
     "time": 1538834664
}
```

Code Box 17, HTTP response with a studyStatus of a specific id.

If the request was not successful and the specific studyStatus was not found the status code 404 will be sent back along with a error message as shown in Code Box 18 below.

```
HTTP/1.1 404 Not found
```

Code Box 18, HTTP response if the studyStatus was not found.

3.6 Delete studyStatus

Send a request like the one in Code Box 19 to delete a studyStatus.The '123' in this request stands for the statuses 'studyId'.

```
DELETE /studyStatus/123 HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code box 19, HTTP request to delete a studyStatus.

A response with the status code 204 (Code Box 20) will be sent back if the request was successful along with a status message like shown in Table 4.

```
HTTP/1.1 204 No content

Location:
http://localhost:6000/study-status/123
```

Code box 20, HTTP request to delete an account.

3.7 Get a specific User

To find a specific user, to possibly befriend, send a HTTP request like the one below in Code Box 21.

```
GET /accounts/username HTTP/1.1

Host: localhost:6000

Accept: application/json
```

Code box 21, HTTP request to retrieve an Account with specific username.

A response with status code 200 should be returned if the request was successful (Code Box 22) and the user was found, otherwise a 404 status code (Code Box 23) will be shown.

```
HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: 33

{

     "accountId": 23,

     "username": "The username…"

}
```

Code Box 22, HTTP response with a specific username.

```
HTTP/1.1 404 Not found
```

Code Box 23, HTTP response if the User was not found.

3.8 Delete account

Send a request like the one in Code Box 24 to delete an account. The '123' in this request stands for the users 'accountId'.

```
DELETE /accounts/123 HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code box 24, HTTP request to delete an account.

A response with the status code 204 will be sent back (Code Box 24) if the request was successful.

```
HTTP/1.1 204 No content
Location: http://localhost:6000/accounts/123
```

Code box 24, HTTP request to delete an account.

3.9 Update account

Send a request like the one in the Code Box below (Code Box 25) to update the information of an existing account. The '1' in this request stands for the users 'Id'.

```
PUT /accounts/1 HTTP/1.1
Host: localhost:6000
Accept: application/json
Content-Type: application/json
Content-Length: 98

{
     "Username": "The username",
     "Password": "The password.."
}
```

Code box 25, HTTP request to update an account.

If the request was successful the status code 204 will be returned (see Code box 26) , if not the status code 422 will be sent back as shown in Code Box 27.

```
HTTP/1.1 204 No content
```

Code box 26, HTTP response if the account was updated.

```
HTTP/1.1 422 Unprocessable Entity
```

Code box 27, HTTP response if the request was not successful.

3.10 Add friend

To add another account to the friend list and label that account as a friend send a post request as shown in Code Box 28. The '8' represents the users 'id'.

```
POST /8/friends HTTP/1.1
Host: localhost:6000
Accept: application/json
Content-Type: application/json
Content-Length: 98

{
     "AccountId": 27
}
```

Code box 28, HTTP request to add a friend.

If the request was successful a response will be sent back with the code 201 (see Code Box 29). If the request was unsuccessful status code 422 will be sent back instead (see Code Box 30).

```
HTTP/1.1 201 Created
Location: http://localhost:6000/friends/321
```

Code Box 29, HTTP response when a friend is added.

```
HTTP/1.1 422 Unprocessable Entity
```

Code box 30, HTTP response if the request was not successful.

3.11 Get all friends

View all friends by getting all friends where the friendship was confirmed/accepted. Send a request like the one in Code Box 31 to do this. The '1' in this request stands for the users 'Id'.

```
GET /1/friends HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code Box 31, HTTP request to get all friends.

If any friends where found the status code 200 is returned together with the id and username of friends. (Code Box 32)

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 213
[{
     "id": 12,
     "username": "The username"
}, {
   ...
},
...
]
```

Code Box 32, HTTP response with all friends

If the account does not have any friendships, status code 404 is sent back. (Code Box 33)

```
HTTP/1.1 404 Not found
```

Code Box 33, HTTP response when no friends were found.

3.12 Delete friendship

Send a request like the one in Code Box 34 with the friendId to remove a friend. The '321' in this request stands for the 'friendId'.

```
DELETE /friends/321 HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code box 34, HTTP request to delete the link between two accounts.

A response with the status code 204 will be sent back if the request was successful as shown in Code Box 35. Otherwise it has already been deleted and status code 404 will be sent back(see Code Box 36).

```
HTTP/1.1 204 No content
Location: http://localhost:6000/friends/321
```

Code Box 35, HTTP response when the table was updated.

```
HTTP/1.1 404 Not found
```

Code Box 36, HTTP response if the friendId was not found.

3.13 Confirm friendship

Send a request like the one in Code Box 37 to confirm a friend request, the integer is automatically set to 0 and changes to 1 when confirmed. The '1' in this request stands for the users 'Id. The 13 is the friendId.

```
PATCH /1/friends/13 HTTP/1.1
Host: localhost:6000
Accept: application/json
Content-Type: application/json
Content-Length: 28
{
     "confirmed": 1
}
```

Code 37, HTTP request to update the table Friends.

If the request was successful a response with the status code 204 will be sent back as shown in Code Box 38, otherwise response with status code 400 will be sent back (see Code Box 39).

```
HTTP/1.1 204 No content
```

```
Location: http://localhost:6000/friend/321
```

Code Box 38, HTTP response when the table was updated.

```
HTTP/1.1 422 Unprocessable Entity
```

Code Box 39, HTTP response when the request was unsuccessful.

3.14 Join friend

To join a friend (join the status the user has created) send a request as shown in Code Box 40.

```
POST /joinFriend HTTP/1.1
Host: localhost:6000
Accept: application/json
Content-Type: application/json
Content-Length: 200

{
     "studyId": 321,
     "accountId": 32
}
```

Code Box 40, HTTP request to join a friend.

If it is malformed, status code 422 is returned. Shown in Code Box 41.

```
HTTP/1.1 422 Unprocessable Entity
```

Code Box 41, HTTP response if the studyStatus is malformed.

When carried out correctly and the studyStatus is created and posted, a response with the status code 201 is returned. This is shown in Code Box 42 below.

```
HTTP/1.1 201 Created
Location: http://localhost:6000/join-friend/321
```

Code Box 42, HTTP response if the studyStatus is malformed.

3.15 Get all friend requests

View all friend requests where the friendship was not confirmed/accepted yet by sending a request like the one in Code Box 43 to do this. The '1' in this request stands for the users 'Id'.

```
GET /1/friend-requests HTTP/1.1
Host: localhost:6000
Accept: application/json
```

Code Box 43, HTTP request to get all studyStatus from friends.

If any friend requests where found the status code 200 is returned together with the accounts that is not confirmed as friends.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 213
[{
     "id": 12,
     "username": "The username"
}, {
   ...
},
...
]
```

Code Box 44, HTTP response with the friend requests

If the account does not have any requests, status code 404 is sent back. (Code Box 45)

```
HTTP/1.1 404 Not found
```

Code Box 45, HTTP response when no friend requests were found.