TheALLDictionary:

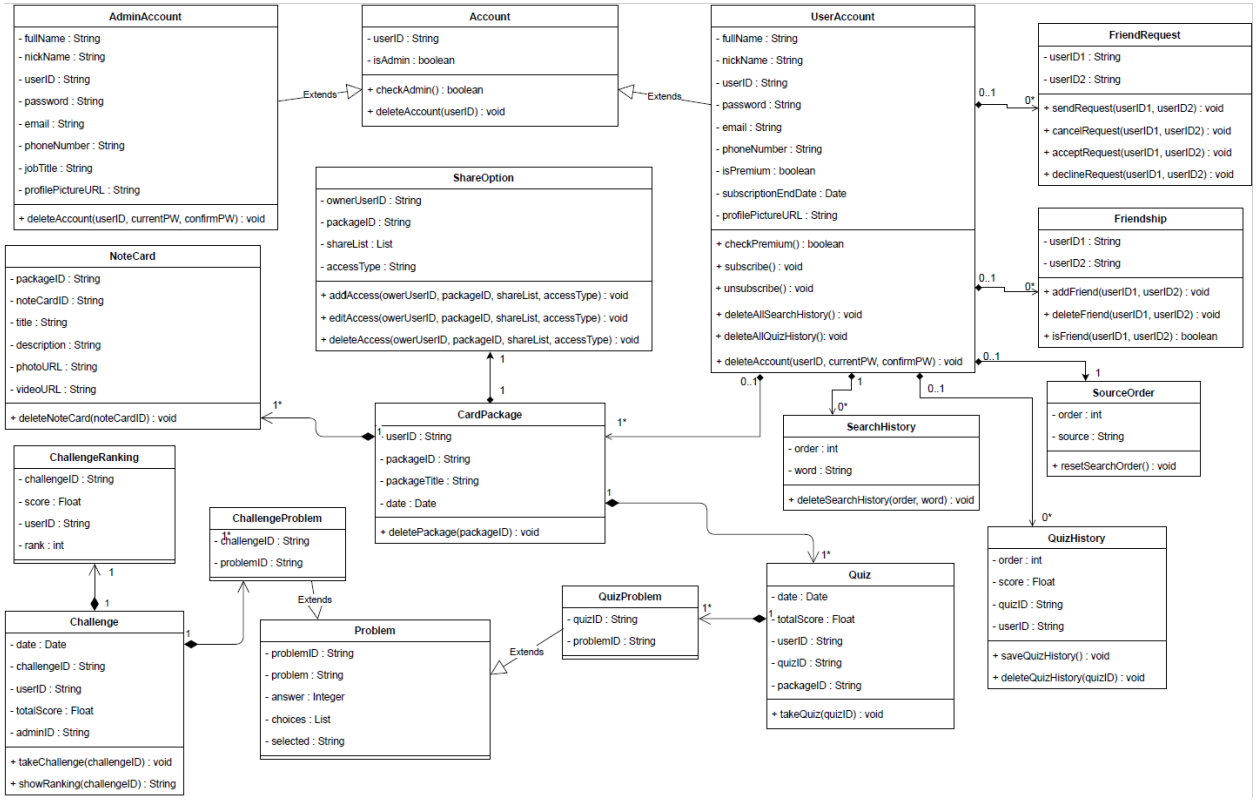
Data Design Document

Spring 2021 CSE 416 Final project

Team Members:

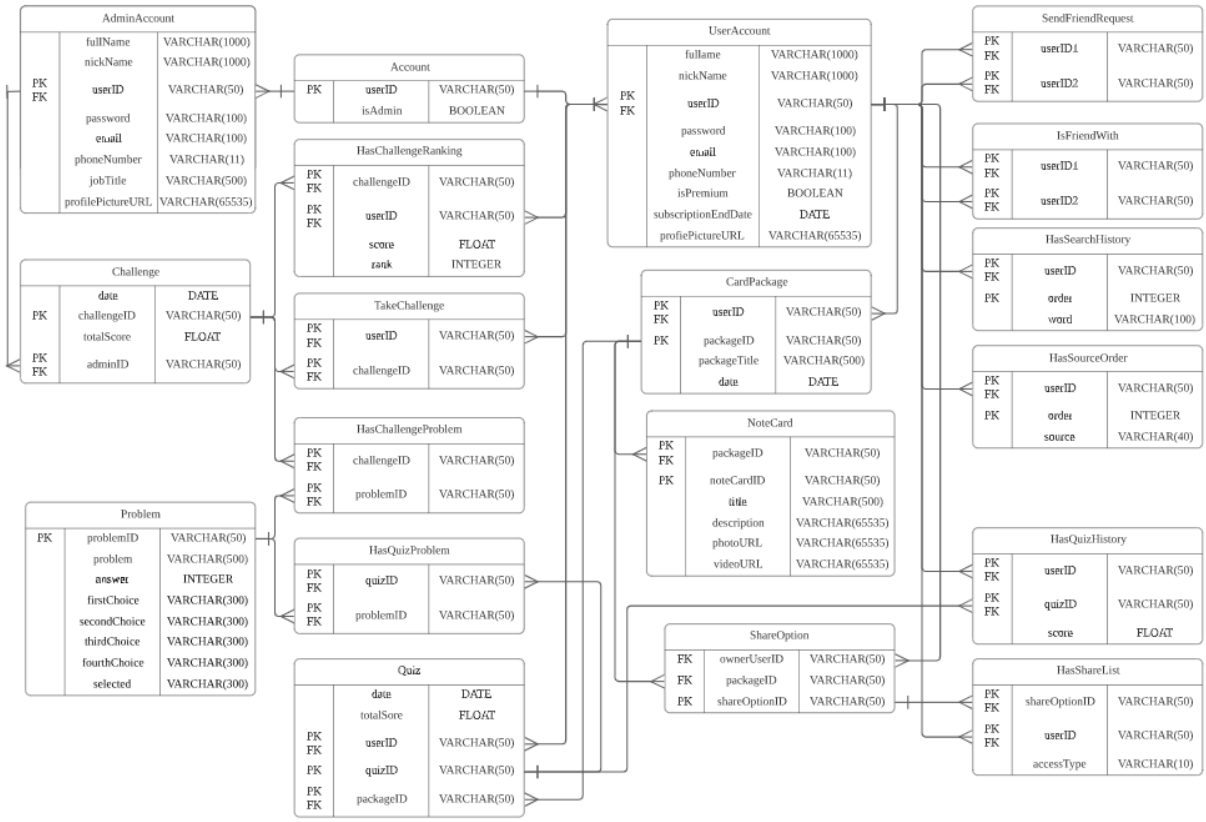
Young-jae Moon → Product owner	
Sub-team A	Sub-team B
Young-jae Moon → Project manager youngjae.moon@stonybrook.edu Yoo-ra Kim → Lead programmer yoora.kim@stonybrook.edu	Seung-jae Kang → Designer seungjae.kang@stonybrook.edu Seo-young Ko → Designer seoyoung.ko@stonybrook.edu

1. UML Class Diagram



Here is the Unified Modeling Language (UML) Class Diagram of TheALLDictionary.

2. Initial Entity-Relationship (ER) Diagram



Here is the ER diagram of TheALLDictionary initially created based upon the UML Class Diagram of TheALLDictionary. Therefore, note that this is not normalised to Boyce–Codd normal form (BCNF) yet. We have made this ER diagram because we decided to use Amazon Relational Database Service to implement our project.

Note that we decided to change the table representing the “SourceOrder” class into “HasSourceOrder,” the table representing the “SearchHistory” class into “HasSearchHistory,” and the table representing the “QuizHistory” class into “HasQuizHistory” for our initial ER diagram. This is because we have initially combined the “has” relationship and the “SearchHistory” entity into a single table.

Moreover, we decided to change the name for the relations created from the "FriendRequest" and "Friendship" classes in the UML Class Diagram to "SendFriendRequest" and "IsFriendWith" as they all represent a relationship between two UserAccount entities. Likewise, we decided to change the name of the table created from the "ChallengeProblem" class into "HasChallengeProblem," as it represents a relationship between Challenge and Problem entities. Similarly, we decided to change the name of the table created from the "QuizProblem" class into "HasQuizProblem," as it represents a relationship between Quiz and Problem entities.

Also, we decided to add the relationship "TakeChallenge" as it represents a relationship between Challenge and UserAccount entities.

3. Database design and Normalization

Table name: Account

Attributes: userID U, isAdmin A

Functional Dependencies:
There is no functional dependency.

Normalization:
The relation is BCNF.

Table name: AdminAccount

Attributes: fullName F, nickName K, userID U, password W, email E, phoneNumber N, jobTitle J, profilePictureURL P

Functional Dependencies:
U → FKUWENJP
N → FK
E → FK

Normalization:
The relation is BCNF.

Table name: UserAccount

Attributes: fullName F, nickName K, userID U, password W, email E, phoneNumber N, isPremium I, subscriptionEndDate D, profilePictureURL P

Functional Dependencies:
U → FKUWENIDP
N → FK
E → FK
D → I

Normalization:
The relation is BCNF.

Table name: SendFriendRequest

Attributes: userID1 F, userID2 S

Normalization:

There is no need to normalize this table as ‘SendFriendRequest’ represents a relationship between two entities as shown below.

UserAccount (Entity) - SendFriendRequest (Relationship) - UserAccount (Entity)

Table name: IsFriendWith

Attributes: userID1 F, userID2 S

Normalization:
There is no need to normalize this table as ‘IsFriendWith’ represents a relationship between two entities as shown below.

UserAccount (Entity) - IsFriendWith (Relationship) - UserAccount (Entity)

Table name: HasSearchHistory

Attributes: userID U, order O, word W

Functional Dependencies:
UO → UOW

Normalization:
The relation is in BCNF.

Table name: HasSourceOrder

Attributes: userID U, order O, source S

Functional Dependencies:
UO → UOS

Normalization:
The relation is in BCNF.

Table name: Quiz

Attributes: date D, totalScore S, userID U, quizID Q, packageID P

Functional Dependencies:
UP → UPQ
Q → QSD

Normalization:
The relation is not in BCNF.

The relation can be decomposed into two relations: GenerateQuiz and Quiz.
GenerateQuiz can have the following three attributes: userID U, packageID P, quizID Q.
Quiz can have the following three attributes: quizID Q, score S, date D.

Table name: HasQuizProblem

Attributes: quizID Q, problemID P

Normalization:
There is no need to normalize this table as 'HasQuizProblem' represents a relationship between two entities as shown below.

Quiz (Entity) - HasQuizProblem (Relation) - Problem (Entity)

Table name: HasQuizHistory

Attributes: userID U, quizID Q, score C

Functional Dependencies:
UQ → UQC

Normalization:
The relation is in BCNF.

Table name: Challenge

Attributes: date D, challengeID C, totalScore S, adminID A

Functional Dependencies:
CA → CAD
C → CS

Normalization:
The relation is not in BCNF.

The relation can be decomposed into two relations: GenerateChallenge and Challenge.
GenerateChallenge can have the following three attributes: adminID A, challengeID C, date D
Challenge can have the following three attributes: challengeID C, totalScore S.

Table name: TakeChallenge

Attributes: userID U, challengeID C

Normalization:
There is no need to normalize this table as 'TakeChallenge' represents a relationship between two entities as shown below.

User (Entity) - TakeChallenge (Relation) - Challenge (Entity)

Table name: ChallengeRanking

Attributes: challengeID C, userID U, rank R, score S

Functional Dependencies:
CU → CURS

Normalization:
The relation is in BCNF.

Table name: HasChallengeProblem

Attributes: challengeID C, problemID P

Normalization:
There is no need to normalize this table as 'HasChallengeProblem' represents a relationship between two entities as shown below.

Challenge (Entity) - HasChallengeProblem (Relation) - Problem (Entity)

Table name: Problem

Attributes: problemID P, problem B, answer A, firstChoice F, secondChoice S, thirdChoice T, fourthChoice F, selected E

Functional Dependencies:
P → PBAFSTFE
BFSTF → A

Normalization:
The relation is in BCNF.

Table name: CardPackage

Attributes: userID U, packageID P, packageTitle T, date D

Functional Dependencies:
P → PTD

Normalization:
The relation is not in BCNF, none of the attributes except userID is dependent on userID.

The relation can be decomposed into two relations: HasCardPackage and CardPackage.

HasCardPackage can have the following two attributes: userID U and packageID P.
CardPackage can have the following two attributes: packageID P, packageTitle T, date D.

Table name: NoteCard

Attributes: packageID P, noteCardID C, title T, description D, photoURL H, videoURL V

Functional Dependencies:
C → CTDHV

Normalization:
The relation is not in BCNF, none of the attributes except packageID is dependent on packageID.

The relation can be decomposed into two relations: HasNoteCard and NoteCard.
HasNoteCard can have the following two attributes: packageID P and noteCardID C.
NoteCard can have the following two attributes: noteCardID C, title T, date D.

Table name: ShareOption

Attributes: ownerUserID U, packageID P, shareOptionID O

Functional Dependencies:
O → UPO

Normalization:
The relation is in BCNF.

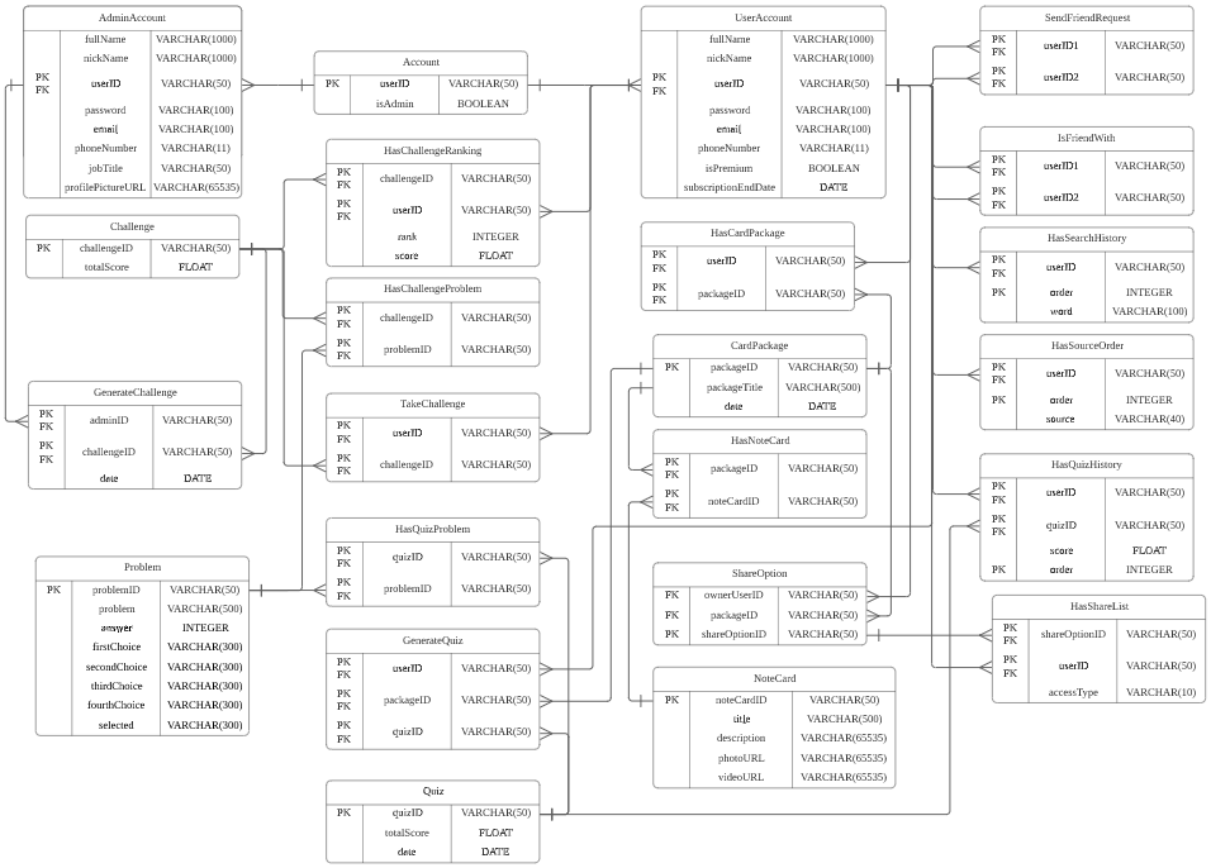
Table name: HasShareList

Attributes: shareOptionID O, userID U, assessType T

Functional Dependencies:
O → OUT

Normalization:
The relation is in BCNF.

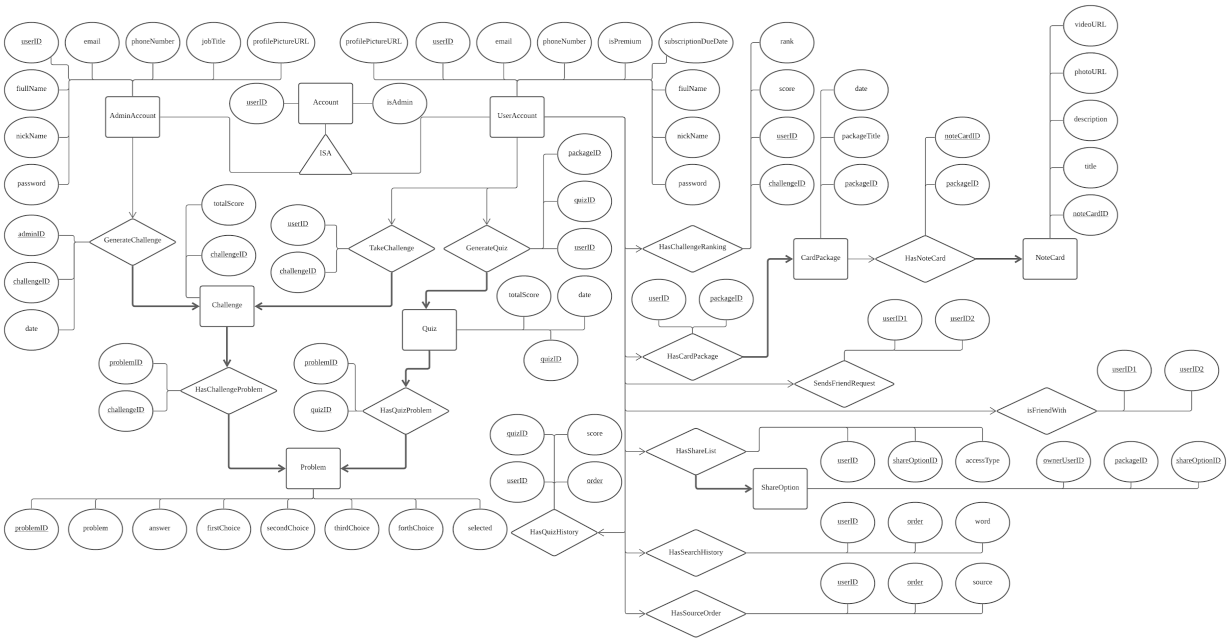
Here is the ER diagram in BCNF drawn by using Lucidchart:



4. Denormalisation and Final ER Diagram

Denormalisation of a normalised database is often needed to increase the speed of reading the database by adding redundant data or grouping data. It, in turn, decreases the speed of writing the database.

In this project, denormalisation is not required as all the functional dependencies have been preserved while normalising the tables. Thus, since it takes a lot of time to draw a final ER diagram with proper symbols and notations in Lucidchart, we decided to recreate a proper ER diagram at the last moment. Here is our final ER diagram which is drawn with proper symbols and notations:



5. Contributions

Name	Contributions & Date Completed (DD/MM/YY)
Young-jae Moon	● I made ‘UML Class Diagram (1st draft).pdf’ with Yoo-ra Kim. (25/03/21) ● I have created and formatted ‘TheALLDictionary-DataDesign.docx’ in the ‘Data Design + Web views’ subfolder of our Google Drive folder. (25/03/21) ● I made ‘UML Class Diagram (2nd draft).pdf’ by myself. (26/03/21) ○ Since userID will be all primary keys for Account, AdminAccount, PremiumAccount, and UserAccount; I decided to change the Account class so that it only includes userID and isAdmin attributes and getUserID, checkAdmin and deleteAccount methods. ○ Thus, I have added userID and password attributes, and getUserID, getPassword, setPassword methods to AdminAccount, PremiumAccount, and UserAccount. ○ Thus, I have added a subscriptionEndDate attribute to PremiumAccount class. ● I made ‘UML Class Diagram (3rd draft).pdf’ by myself. (26/03/21) ○ Combined UserAccount and PremiumAccount into a single class called ‘UserAccount’ for simplification. ○ Added IsPremium and subscriptionEndDate attributes to UserAccount ○ Added CheckPremium method to User Account ○ Edited the relationship between UserAccount and other classes. ● I made ‘UML Class Diagram (4th draft).pdf’ with Yoo-ra Kim. (26/03/21) ○ Added ShareOption class which has four attributes and three methods. ○ Added the return type of getFriendList method in UserAccount. ○ Reorganised the diagram to look better. ● I made ‘ERD (before Normalization).pdf’ with Yoo-ra Kim based upon the 4th draft of our UML Class Diagram using LucidChart. (26/03/21) ● I have uploaded a screenshot of the UML Class Diagram in Chapter 1. (26/03/21) ● I have formatted Chapter 3 and added all the names of each table in the ERD into the table in Chapter 3. (26/03/21)

	● I have completed the first row of the table in Chapter 3. (26/03/21) ● I have completed the remaining rows of the table in Chapter 3 by myself. (27/03/21) ● I have changed the class name for SearchOrder to SourceOrder in our 4th draft of UML Class Diagram, and then reuploaded the screenshot of the UML Class Diagram in Chapter 1 by myself. (27/03/21) ● I have edited some of the table names in the 'ERD (before Normalization).pdf' and reuploaded to our Google Drive Folder by myself. Thus, I have reuploaded the screenshot of the ER diagram before normalization in Chapter 2 by myself. (27/03/21) ○ SourceOrder → HasSourceOrder ○ SearchHistory → HasSearchHistory ○ QuizHistory → HasQuizHistory ○ FriendRequest → SendFriendRequest ○ Friendship → IsFriendWith ○ QuizProblem → HasQuizProblem ○ ChallengeProblem → HasChallengeProblem ● I wrote the textual description for Chapter 1 and Chapter 2 by myself. (27/03/21) ● I made "ERD BCNF.pdf" together with Yoo-ra Kim. (27/03/21) ● I wrote the textual description for Chapter 3 and Chapter 4 by myself. (27/03/21) ○ Yoo-ra will draw the final ER diagram by her hands and will upload it later. ● I decided to use Flask framework, Heroku, MySQL for back-end programming in addition to Amazon RDS and Amazon S3. (29/03/21) ○ Thus decided to use REST API to connect React and Flask. ● I made 'UML Class Diagram (5th draft).pdf' with Yoo-ra Kim. (29/03/21) ○ Contains more attributes for challenge class. ○ Changed score → totalScore for challenge and quiz class. ● I edited the ER diagram before & after normalisation (29/03/21) ○ Score is not FK anymore for HasQuizHistory and HasChallengeRanking tables ○ Reuploaded the pdf file into our Google Drive folder. ○ Updated relevant materials in Chapter 2 and Chapter 3. ● I edited some wordings for Chapter 3 as Yoora told me that it is hard to draw it by hand due to too many attributes, entities and the relationships between them and therefore drew it by LucidChart in a different manner. (29/03/21) ● Added textual description for Chapter 3. (31/03/21)

	● Deleted the chapter for NoSQL. (31/03/21) ● Created 'UML Class Diagram (6th draft).pdf.' (31/03/21) ○ Removed "all the basic getters and setters from the diagram that just access or set an attribute on the class. They add little extra information and can clutter the screen to make it harder to take in the diagram." ○ Deleted 'firstName' and 'lastName' attributes for UserAccount and AdminAccount classes. ○ Added 'fullName' and 'nickName' attributes for User Account and AdminAccount classes. ○ Updated the UML Class Diagram for Chapter 1. ● Created 'UML Class Diagram (7th draft).pdf.' (31/03/21) ○ Changed the data type for the answer attribute in Problem class from String to Integer. This is because all the problems are currently in multiple choice format. ○ Added profilePictureURL attribute to AdminAccount and UserAccount class. ○ Added photoURL and videoURL attributes to NoteCard class. ○ Updated the UML Class Diagram for Chapter 1. ● Revised the initial ER diagram for Chapter 2. (31/03/21) ○ Removed 'firstname' and 'lastName' attributes for UserAccount and AdminAccount tables. ○ Added 'fullName' and 'nickName' attributes for User Account and AdminAccount tables. ○ Added profilePictureURL attribute to AdminAccount and UserAccount tables. ○ Added photoURL and videoURL attributes to NoteCard tables. ○ Edited the size of VARCHAR for various attributes. ● Updated the table for the normalization due to the changes in the attributes. (31/03/21) ● Updated the normalised ER diagram drawn by LucidChart for Chapter 3. (31/03/21). ● Uploaded the MySQL code for creating tables in the database to our GitHub repository. (05/04/21). ○ Changed the attribute name 'date' to 'dateCreated' as 'date' is a datatype in MySQL ○ Added delete options for each foreign key. ○ Labelled attributes which shall be 'NOT NULL' ○ Set Engine as 'InnoDB' ○ Default character set → 'utf-8'
Yoo-ra Kim	● I made 'UML Class Diagram (1st draft).pdf' with Young-jae Moon. (25/03/21) ● I made 'UML Class Diagram (4th draft).pdf' with Yoo-ra Kim.

	(26/03/21) ○ Added ShareOption class which has four attributes and three methods. ○ Added the return type of getFriendList method in UserAccount. ○ Reorganised the diagram to look better. ● I made 'ERD (before Normalization).pdf' with Young-jae Moon based upon the 4th draft of our UML Class Diagram using LucidChart. (26/03/21) ● I have uploaded a screenshot of the ER diagram before normalization in Chapter 2. (26/03/21) ● I wrote all the attributes for each table in Chapter 3. (26/03/21) ● I made "ERD BCNF.pdf" together with Young-jae Moon. (27/03/21) ● I added one more TakeChallenge for the ER diagrams in Chapters 2 and 3, and added two rows for showing the normalisation processes of these new tables. (28/03/21) ● I made 'UML Class Diagram (5th draft).pdf' with Young-jae Moon. (29/03/21) ○ Contains more attributes for challenge class. ○ Changed score → totalScore for challenge and quiz class. ● I made the final ER diagram in Chapter 4. (29/03/21) ● I updated the final ER diagram in Chapter 4. (31/03/21) ● I made a sign in page front-end (01/04/21) ● I made a sign up page front-end (01/04/21) ● I made a search page front-end (02/04/21) ● I made a search result page front-end (02/04/21) ● I made a change dictionary order page front-end (03/04/21) ● I made a my page front-end (03/04/21) ● I made a payment front-end (04/04/21) ● I made a friend front-end (04/04/21)
Seo-young Ko	● Made Card Show page front-end (05/04/21) ● Made Card Show2 page front-end (05/04/21) ● Made Card Making page front-end (03/04/21) ● Made Card homepage front-end (04/04/21)
Seung-jae Kang	● I made a quiz start page front-end. (02/04/21) ● I made a quiz page front-end. (02/04/21) ● I made a quiz result page front-end. (03/04/21) ● I made a quiz review page front-end. (03/04/21) ● I made a quiz history page front-end. (03/04/21) ● I made a rank page front-ent. (03/04/21) ● I made a word challenge page front-end. (04/04/21) ● I made a word challenge quiz page front-end. (04/04/21) ● I made a word challenge result page front-end. (04/04/21)

	• I made a temporary brand logo. (04/04/21)

Also note that Yoo-ra and Seo-young have changed their roles. Yoo-ra will work on the front-end, while Seo-young will work on the back-end. Thus, Yoo-ra will focus more on the word searching and displaying functionalities, while Seo-young will focus on the note card and other features related to note card functionalities.

Yoo-ra Kim will be the only lead programmer in our team, and Seo-young will focus on the designer role.

Thus, Young-jae worked on writing MySQL code for creating tables in the database and worked on Chapter 1, Chapter 2.1, Chapter 2.2, Chapter 2.4, Chapter 3.5 of the Software Design Document, as Young-jae and Yoora has completed the data design part much faster than the web-views part, and it becomes less productive to work on the web-views with four people.