[DANDELiON] Report

Table of Contents

Team Members and Roles

UID	Name	Role
u6706220	Zhuxuan Yan	Structure and UI
u7516976	Zixuan Wang	UI design
u7560434	Ethan Yifan Zhu	Logic and tests
u7518626	Pin-Shen Chang	Code implementation
u7532738	Jinhan Tan	Data and Logic and tests

Summary of Individual Contributions

u7532738, Jinhan Tan, I contribute 20% of the code. Here are my contributions:

- [Comment.class]
- [CommentDao.class]
- [CommentDaoInterface.interface]
- [Course.class]
- [CourseDao.class]
- [CourseDaoInterface.interface]
- [MyUser.class]
- [User.class]
- [UserDao.class]
- [UserDaoInterface.class]
- [Message.class]
- [ChatActivity.class]
- [ChatAdapter.class]
- [ContactListActivity.class]
- [ContactListAdapter.class]

- [CourseHomeActivity.class]
- [CourseJoinActivity.class]
- [DataGenerator.class]
- [LocalFileToFirebase.class]
- [RBTree.class]
- [SearchActivity.class]
- [Global.class]
- [comment.xml]
- [user.json]
- [course.json]
- [content.csv]

For my contribution, I designed the database of comments, courses, user and message. I randomly generated 2500 comments and write them into comment.xml file using DataGenerator.createData. Also, I created user.json and some other local files to store data and created Dao to read data from local files. But later I found that I can not modify data since we cannot write to assets folder in Android. So I moved the data from local file to firebase storage using LocalFileToFirebase.class.

Additionally, I was responsible for creating UI for search page and chat page. For search page, I used the tokenised and parser my teammate had created to tokenise the user input. And then use ListView to display the result. For chat page, I use firebase realtime database to store and retrieve messages sent by users, and use RecyclerView and ChatAdapter.class to implement the UI which shows the left and right chat bubbles.

I also write the implementation of Red Black Tree, I read the basic code from homework and write the function insert, delete, rotateLeft, rotateRight, and I use the Rad Black Tree to store and sort the comment of each course.

u6706220, Zhuxuan Yan, I contribute 20% of the code. Here are my contributions:

- CourseHomeActivity.class:[L98-L104]
- LoginActivityclass:[L217-L263]
- RegisterActivity.class[L63-L114]
- RegisterActivity.class[L223-L246]
- [TermOfUseActivity.class]
- [PrivacyPolicyActivity.class]
- [HelpActivity.java]
- CourseActivity.xml
- CourseHomeActivity.xml
- CourseJoinActivity.xml
- Login.xml
- deadline.xml
- Settings.xml
- TermOfUseActivity.xml
- PrivacyPolicyActivity.xml
- RegisterActivity.xml

For my contribution, I was mainly responsible for the front UI design. At the beginning of the project, I was responsible for building and designing the necessary pages of the software, such as login, registration, settings, privacy and so on. Considering the usefulness of the software, I set up the creation of buttons on multiple pages and created page jump listeners for them. I built the basic framework for the software. At the later stage, I communicated with the group mates in charge of the database to configure and manage the page navigation together. Set up multiple master pages to complete the functional implementation.

u7560434, Ethan Yifan Zhu, I contribute 20% of the code. Here are my contributions:

- [CourseParser.java]
- [CourseToken.java]
- [CourseTokenizer.java]
- [Tokenizer.java]
- LoginActivity.java: [L80-L117], [private Location getLocation()], [private String getAddress(Location location)], [private void setLanguage(String country)]
- RegisterActivity.java: [L120-L221], [L247-L313]
- SettingsActivity.java: [L58-L210], [public List getAllLocale()]
- [ParserTest.java]
- [LoginTest.java]
- [RegisterTest.java]

For my contribution, I was mainly responsible for back-end logic and writing tests. After front-end pages are designed, I wrote the basic logic for those UI, i.e. provide error message when inputting wrong texts, checking consistent input details, add functions to buttons, get GPS location functions and etc.

Besides, I wrote Tokenizer and Parser and their tests for searching method. Firstly I designed a grammer for our searching method, as well as design tokens and related operations in CourseToken.java. Then wrote an abstract Tokenizer class with interfaces to get current token, check whether there is a token next, and get next token. Then wrote a class called CourseToken.java to implement this abstract method. Finally in CourseParser.java, we can seperate searching string according to our gramma into different tokens, then we can get our searching keywords based on different type.

On the other hand, I was mainly responsible for UI tests using espresso. I wrote different tests on different pages, like LoginActivity, RegisterActivity and etc, to detect whether our page worked successfully.

u7516976, Zixuan Wang, I contribute 20% of the code. Here are my contributions:

- [MycourseAdapter.class]
- [CourselistFragment.class]
- [myCourseFragment.class]
- myCourseViewModel.class:[L17-L31]
- SettingsViewModel.class:[L10-L22]
- SettingsFragment.class:[L22-L42]
- activity_contact_list.xml
- activity_mycourse.xml
- course_list.xml
- fragment_home.xml
- fragment_transform.xml
- item_transform.xml
- list_item.xml
- activity_navigation_drawer.xml

- mobile_navigation.xml

For my contribution I designed the navigation bar, the course listings and the UI design and access data for my courses. For the home page (HomeFragment) I used the adapter design pattern by creating a HomeAdapter which I used to assign product images and details to the container of the CardView. I read the JSON file and pass it to the HomeFragment.class, and then I can use this data to create my adapter. In the beginning, I used FireBase to get the course data and passed it to MyCourseAdapter which contains the Course_list and data, and I created an adapter for the RecyclerView.

u7518626, Pin-Shen Chang, I contribute 20% of the code. Here are my contributions:

- LoginActivity.java: [private boolean checkLocationPermission()], [public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)]
- NavigationActivityjava [L58-L61], [L64-L91], [public boolean onMenuItemClick(MenuItem item)], [L103-L115], [public boolean onCreateOptionsMenu(Menu menu)]
- SettingsActivityjava [L213-L220], [private void showLogoutConfirmationDialog()], [public boolean onOptionsItemSelected(MenuItem item)]
- RegisterActivityjava [L66-L101]
- [DeadlineshowFragmentjava]
- activity_navigation.xml

For my contribution, My primary responsibility is to write correct and appropriate codes to implement various functions and design user interfaces so that users can smoothly use the rich features of our developed app. To be more specific, for example, asking the user if they grant location permission when they first open the app, allowing users to click hyperlinks on the registration page to view detailed information, enabling users to search for desired keywords by clicking the search icon on the top right of the main screen, and providing three different buttons on the bottom of the sidebar to jump to the corresponding pages. When users want to log out of the app, they only need to click the "Settings" icon to enter the Settings page, and then scroll to the bottom of the app to find the "Log out" button. To prevent users from accidentally clicking this button, we have implemented a pop-up window to confirm if the user wants to log out. Only when the user confirms will they be logged out, otherwise, not. *you should ALSO provide links to the specified classes and/or functions*

Conflict Resolution Protocol

In the first meeting after completing the team formation, we reached the following consensus on the conflict resolution protocol:

1. `Task Delegation in Case of Illness:` If a team member is unwell, the individual who completes their assigned task first will assist in completing the unwell member's task. Once recovered, the unwell member will assist with the helper's other assigned tasks, ensuring a balanced workload. Priority will be given to key requirements of the assignment.

2. `Conflict Resolution:` In the event of team issues or disagreements, resolution will be achieved through a majority vote, ensuring fair and democratic decision-making.

3. `Task Difficulty:` If a team member encounters difficulties with their assigned task, they are expected to flag the issue via the group chat. The team will then collaboratively work through the issue, promoting a supportive and cooperative work environment.

4. `Code Documentation:` All team members are required to maintain clear and comprehensive code documentation. This ensures that others can easily understand and troubleshoot the code, or assist in completing the task if necessary.

5. `Communication of Progress and Issues:` Updates on task progress and any issues should be communicated promptly in the group chat and during team meetings. This ensures transparency and allows for timely assistance or adjustments as needed.

Application Description

What is DANDELiON：

DANDELiON is a comprehensive learning aid app designed to enhance and streamline the educational process. First and foremost, it encompasses most features of a traditional course learning software, such as search courses，joining courses, watching lessons, commenting on courses, and opting out of courses. This grants users the flexibility and engagement they need in an online learning environment. Secondly, DANDELiON integrates a calendar functionality, setting reminders for upcoming deadlines and study times. This feature ensures that users stay organized and up-to-date, providing convenient prompts to optimize their learning schedule. Thirdly, the app supports peer-to-peer real-time chat, enabling users to communicate with classmates enrolled in the same course. This fosters a sense of community and enhances collaborative learning. In summary, DANDELiON is not just an app, but a comprehensive learning companion, blending the traditional learning software capabilities, scheduling aids, and real-time communication to create an enriching and interactive educational experience.

Application Use Guide:

For the Application Use Guide, please see the following external links to the documentation.Application Use Guide

Target Users

DANDELiON is designed to cater to a wide range of users, from high school students to college students, from adult learners to professionals looking to upskill. Here are a few use cases illustrating how different people might utilize the app:

1. `High School Students:` Sarah, a high school student, uses DANDELiON to keep track of her various online courses. She appreciates the integrated calendar feature that reminds her of upcoming deadlines and study times, ensuring she never misses an assignment. She also uses the real-time chat function to discuss homework problems with her classmates.

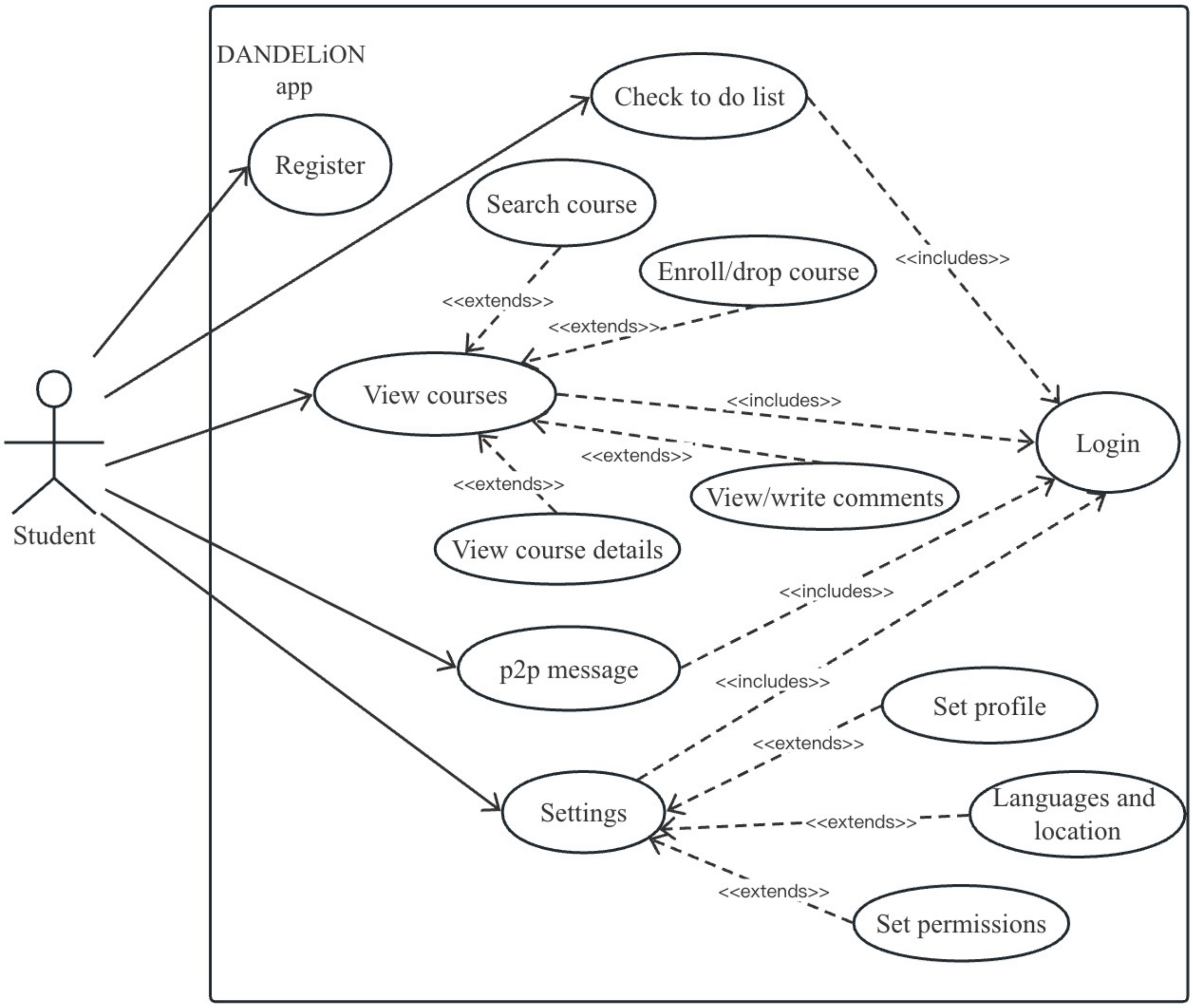
2. `College Students:` John, a college student, leverages DANDELiON to take additional online courses beyond his university curriculum. He enjoys the flexibility of joining and opting out of courses as his interests and needs evolve. The peer-to-peer chat feature also allows him to connect with other learners across the globe, fostering a sense of global community.

3. `Adult Learners:` Linda, an adult learner, uses DANDELiON to learn new skills in her free time. She appreciates the convenience of watching lessons at her own pace and the ability to comment on the courses. The reminders from the calendar function help her balance learning with her busy work schedule.

4. `Professionals:` Mike, a professional looking to upskill, uses DANDELiON to take specialized courses relevant to his field. The app allows him to engage with the material whenever he finds time amidst his demanding professional life. He also values the real-time chat function to network with peers and exchange ideas.

In summary, DANDELiON is a versatile and user-friendly app that can be adapted to the diverse learning needs and schedules of its users, whether they are school students, college learners, adult self-learners, or busy professionals.

Application Use Cases and Example



- Bernardo wants to register an account for the DANDELiON app.

  1. He accesses the DANDELiON app and goes to the Login Page.
  2. He clicks the Register Button and jump into Register page.
  3. He types name, email and password, and check two boxes.
  4. He clicks the JOIN NOW button and register seccessful.

- Bernardo wants to Login into the DANDELiON app.

  1. He accesses the DANDELiON app and goes to the Login Page.

2. He types email and matched password in the text box.
3. He clicks the Login button and login seccessful, goes to the main page.

- Bernardo wants to check his to-do list for the week.

  1. He goes to the DANDELiON app and logs into his account
  2. He clicks on "Deadline" in the navigation bar and goes to the deadline page.
  3. He clicks on the date of the calendar and checks the to-do list for the corresponding date.

- Bernardo wants to change his name in his personal information.

  1. He goes to the DANDELiON app and logs into his account
  2. He clicks on the "Settings" icon in the navigation bar and goes to the Settings page.
  3. He clicks on the name to get a popup with a text box.
  4. He enters the modified name in the text box and clicks "OK".

- Bernardo would like to view a course on java programming.

  1. He logs into his account and goes to the DANDELiON app
  2. He clicks on the navigation bar icon in the top left corner of the page, the navigation bar expands and he clicks on the "Course list" option under personal information.
  3. He goes to the "Course list" page, which shows all the courses offered.
  4. Click on the search button at the top of this page and the page will jump to the search page.
  5. Search for the course name or course number in the text box on the search page to view the course in question.

- Bernardo was a little concerned about the difficulty of the Comp6442 and was going to check the reviews to decide whether to register for the course.

  1. He logs into his account and goes to the DANDELiON app
  2. He clicks on the navigation bar icon in the top left corner of the page, the navigation bar expands and he clicks on the Course list option under Personal Information. 3.
  3. Once he is on the Course list page, browse the comments of people who have enrolled in the course on this page.

- Bernardo wants to register for Comp6442

  1. He logs into his account and goes to the DANDELiON app
  2. He clicks on the navigation bar icon in the top left corner of the page, the navigation bar expands and he clicks on the course listing option under personal information.
  3. On the course list page, he finds and clicks on the course
  4. He find the "Join Now" button on the course description page and clicks on it.

- Bernardo has registered for the Comp6442 course but finds that he does not have the time to complete the course on time, so he wants to withdraw from it.

  1. He logs into his account and goes to the DANDELiON app
  2. He clicks on the navigation bar icon in the top left corner of the page, the navigation bar expands and he clicks on the My Courses option under Personal Information.
  3. Once on the My Courses page, he will browse through the courses he has registered for and click on the "Drop Course" button.

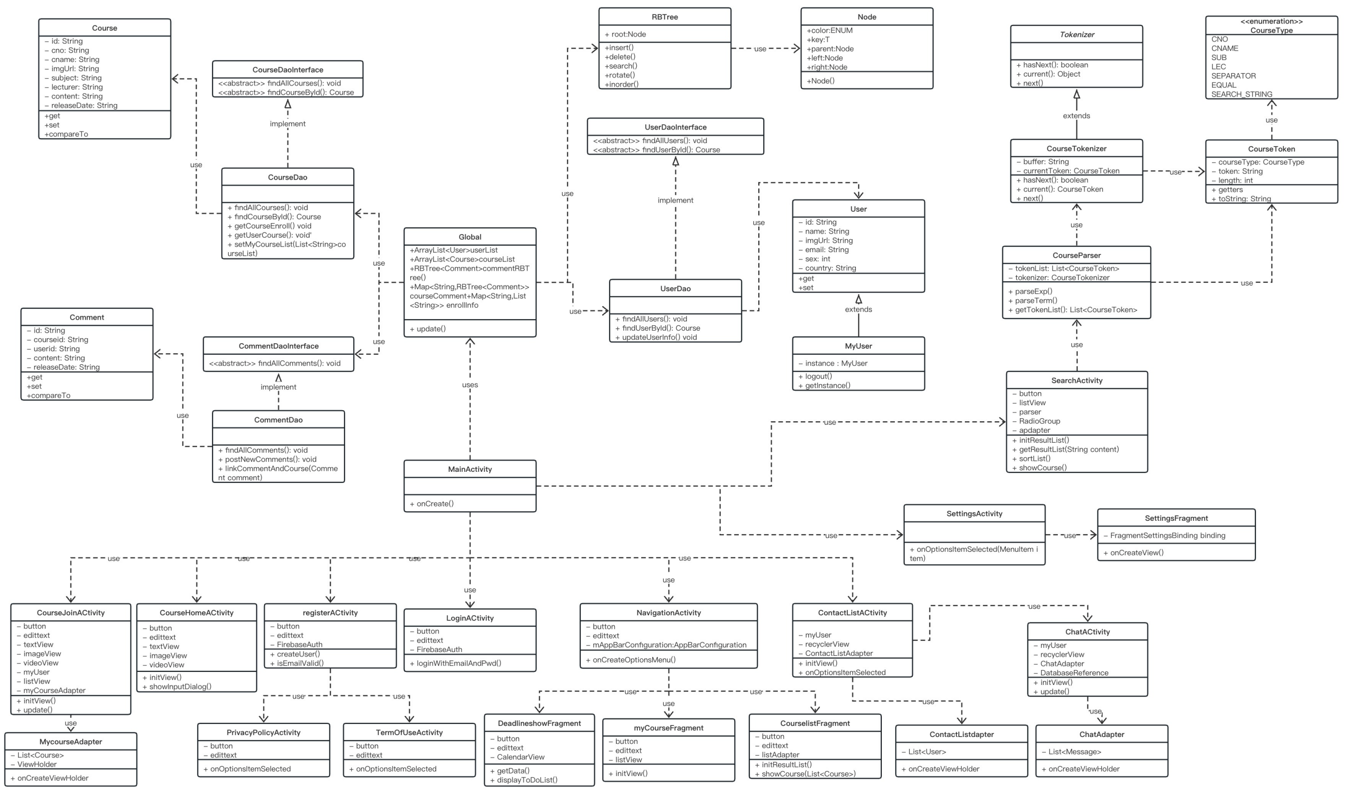
- Bernardo has completed the Comp6442 course and would like to review it and get more people to sign up for the course.

    1. He logs into his account and goes to the DANDELiON app
    2. He clicks on the navigation bar icon in the top left corner of the page, the navigation bar expands and he clicks on the My course option under Personal Information.
    3. Once on the My course page, he will browse through the courses he has registered for and click on the course he wants to evaluate.
    4. He will see the write a comment button below the course description page.
    5. He write his comments and thoughts in the pop-up window, and press the send button to send his comments.

- Bernardo is looking for students who have enrolled in Comp6442 together to discuss problems they have encountered on the course.

    1. He goes to the DANDELiON app and logs into his account
    2. Once he is in the app, click on the information icon at the bottom of the page to see the list of contacts.
    3. He can select the person in the contact list and go to the chat page. In the chat page, enter the message he want to communicate with and click on the send button.

Application UML



Application Design and Decisions

Data Structures

- *We used the following data structures in our project:*

1. *Red Black Tree*
    - *Objective: It is used for storing and sorting Comments for storage and interactivity feature.*
    - *Locations:*
        - [Line 50-52 in Global.java]
    - *Reasons:*
        - *It is more efficient than Arraylist for insertion and sorting*
2. *ArrayList*
    - *Objective: It is used for storing courses, users, and messages, and is assigned to the adatpter of RecyclerView for message feature and data visualisation feature.*
    - *Locations:*
        - [Line 46-47 in Global.java]
        - [Line 42 in ChatActivity.java]
    - *Reasons:*
        - *It is more efficient to access data in position i with a time complexity O(1)*
        - *We don't need to insert and delete data for this feature*
        - *We need to use arrayList to fit the adapter of RecyclerView*
3. *HashMap*
    - *Objective: It is used for storing the pair of courses and comment information, and course enroll information, it is for user Interactivity feature.*
    - *Locations:*
        - [Line 52-57]
    - *Reasons:*
        - *The course is key, and the comments related to the course is value, it is more efficient to access the comment list by key*

Design Patterns

- *We used the following design patterns in our project:*

1. *DAO*
    - *classes:*
        - [CommentDao.class]
        - [CourseDao.class]
        - [UserDao.class]
    - *Explaination:*
        - Use DAO design pattern when retrieving course, user and comment data from firebase. Data Access Object (DAO) pattern is used to separate low-level data access APIs from high-level logical services.
        - We create *User.java*, *UserDAO.java* and *UserDaoInterface.java* for getting user data. The same with Course data and Comment data.
        - The interface defines the standard operations to be performed on a model object. The Dao class implements the interface and is responsible for obtaining data from firebase. The object class is a simple Java Bean that contains the get & set method to store the data retrieved using the Dao class.
2. *Adapter*

- *classes:*
    - [ChatAdapter.class]
    - [ContactListAdapter.class]
    - [MycourseAdapter.class]
- *Explaination:*
    - We use ChatAdapter to fit the message into corresponding recyyclerView according to whether the message is sent by the user and received by the user, e.g. the message received by the user will be displayed in the left and the message sent by the user will be displayed in the right.
    - We use ContactListAdapter to fit the contact into a list and display them in the recyclerView, where we will display the name of the contact and the message.
    - We use MycourseAdapter to display the card view of the course which user has enrolled.

3. *Singleton*
    - *classes:*
        - [MyUser.class]
    - *Explaination:*
        - The singleton design pattern used is lazy singleton, which means we intialize the instance when we call the getInstance() for the first time.
        - In our project, there is only one current user at a time, so the class should create its own unique instance when the user login or register. And the instance should be reset to null when the user log out.
        - We use singleton design pattern to design the MyUser class, which ensure the class only has one instance and provide a global point of access to it.

Grammar

This grammar is designed for search page. The main purpose of it is to tokenizer the well-formed input string into different kinds of tokens we need to pursue searching operation.

Production Rules:

```
  <exp> ::= <term> | <term>;<exp>
  <term> ::= <CNO> = <SEARCH_STRING> | <CNAME> = <SEARCH_STRING> | <SUB> =
<SEARCH_STRING>| <LEC> = <SEARCH_STRING>
```

This grammar consists of two rules as `<exp>` and `<term>`. The `<exp>` is the start symbol, while `<term>` and `<exp>` are non-terminal symbols ,and `<CNO>`, `<CNAME>`, `<SUB>`, `<LEC>`, `<SEARCHING_STRING>`, `";"` and `"="` are terminal symbols.

- For the first rule,

  ```
  <exp> ::= <term> | <term>;<exp>
  ```

  `<exp>` is used to combine one or more `<term>` rules using seperation symbol `";"`.

- For the second rule,

```
<term> ::= <CNO> = <SEARCH_STRING> | <CNAME> = <SEARCH_STRING> | <SUB>
= <SEARCH_STRING> | <LEC> = <SEARCH_STRING>
```

`<term>` is used to define a single searching keyword format, i.e. the part of well-formed format used for searching. The key word should be `<CNO>`, `<CNAME>`, `<SUB>`, `<LEC>`, followed by a terminal symbol `"="`, and search string `<SEARCH_STRING>`.

One single `<term>` is one inquire. Multiple `<term>`s will be combined in first rule and seperated by `";"`.

The advantage of this design is that it is quite simple and easy to understand, as well as it is the most familiar way to do complex search who is not in this area.

Additionally, the use of semicolons as a separator enables users to combine several search parameters into complicated searches without having to be familiar with Boolean logic or other complex query languages.

Tokenizer and Parsers

- classes:
  - [CourseParser.java]
  - [CourseToken.java]
  - [CourseTokenizer.java]
  - [Tokenizer.java]
- explainations:

After we established grammar, now we can use Tokenizer and Parsers to analyse our input strings for searching purpose.

First we design a class for tokens, named in `CourseToken.java` in our file. Different kinds of tokens are listed in an enum called `CourseType`. Despite some getters for private elements, we have a function `public CourseToken(String res)`. It takes a string as a parameter then transfered it into the corresponding token.

Then we design an abstract class called `Tokenizer`, which provides abstract method `public abstract boolean hasNext();`, which is to detect whether there exists next token, `public abstract Object current();`, which is to get current token, and `public abstract void next();`, which is to jump to next token. The purpose of designing this class is to be impleted by different kinds of parsers.

According to our requirement, `CourseTokenizer.java` is designed and it extends our abstract class `Tokenizer`. There are two private fields called `private String buffer;` and `private CourseToken currentToken;`. The former restores remaining string while the latter saves current token. All methods in the abstract class are implemented in this class.

Finally, we design `CourseParser.java` to parse our input string, as it is indicated as a parameter in the construction method. Then remaining methods are designed according to our grammar discussed previously.

The advantage of using tokenizer and parser are listed as below:

- Input search queries can be efficiently parsed using tokenizer and parser. It separates the input query into tokens or discrete components like search keywords, operators, and search strings.
- It is possible to utilise more complex quiries, i.e. using multiple keywords, differnt order of keywords and etc, when using a tokenizer and parser. This would not be possible with straightforward keyword or phrase searches.

Summary of Known Errors and Bugs

[Where are the known errors and bugs? What consequences might they lead to?]

1. *GPS Issue:*

- *The switcher for users to control the permission of GPS location. However, it doesn't work successfully. It results from our Settings Activity is static. We try to change it to non-static, but it happens to cause other problems.*
- *Our app relies on the Google API to obtain user location permissions. This results in our app being unable to retrieve the user's location information if the user has not granted Google Maps location permissions.*

2. *Firebase Issue:*

- The firebase may disconnect when the request for data is too much within one period of time. This is because our premium is the free one so it has some limitation in data retrieve and storage.

Testing Summary

We mainly tested functional classes including Red-Black tree and Tokenizer and Parser. where the corresponding features are Data loading and searching

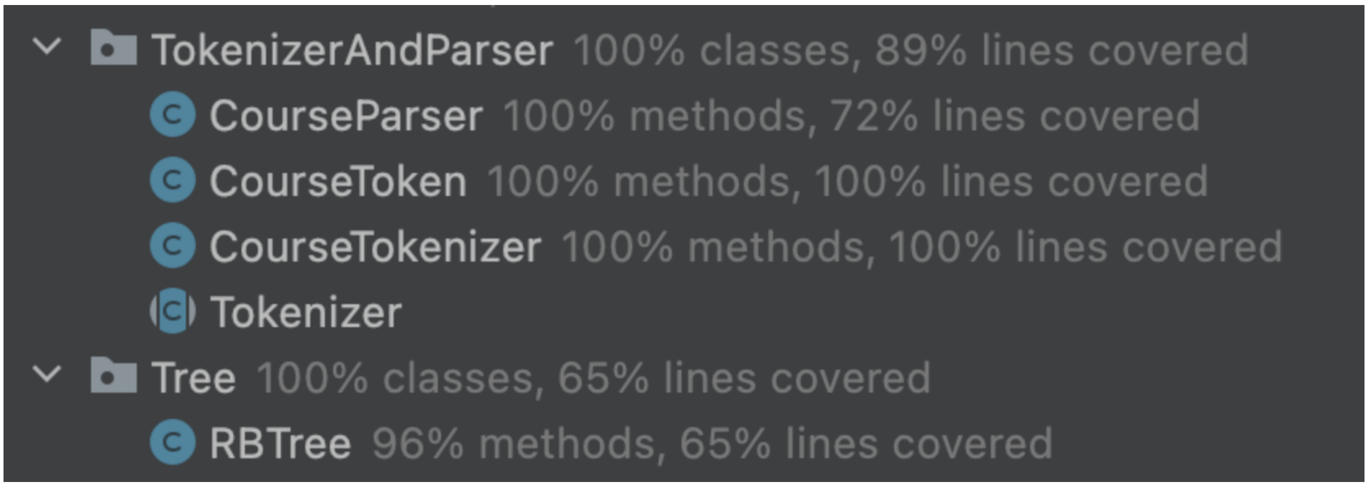
1. RBTree:

- *Number of test cases:* 15
- *Code coverage:* 96%

2. Tokenizer and Parser:

- *Number of test cases:* 5
- *Code coverage:* 100%

This is a screenshot of test coverage:



```
TokenizerAndParser 100% classes, 89% lines covered
    CourseParser 100% methods, 72% lines covered
    CourseToken 100% methods, 100% lines covered
    CourseTokenizer 100% methods, 100% lines covered
    Tokenizer
Tree 100% classes, 65% lines covered
    RBTree 96% methods, 65% lines covered
```

Implemented Features

Basic App

1. [Login]. Description of the feature and your implementation (easy)

    - Class: [MainActivity.java]
    - Class: [LoginActivity.java]
    - Class: [RegisterActivity.java]

    Additional description: The login page requires students to enter their registered email address and username to login. A pop-up window will appear for emails that are not registered. Students can use the registration page to gain login access and add personal details.

2. [Data instances]. There must be data file(s) with at least 2,500 valid data instances. The data files must be used to feed your app, simulating a data stream. For example, every x seconds, a new item is read from a file. An item can be an action (e.g., a new set of lecture notes has been posted by a lecturer; students are added to a course; an assessment submission; a new user signed up; etc). (easy)

    - Class:[DataGenerator]
    - Class:[LocalFileToFirebase]
    - Class:[Comment.class]
    - Class:[CommentDao.class]
    - File: [comment.xml]

    Additional description: 2500 comment data is generated randomly and be writen to firebase. All comments can be loaded from firebase and feeded to our app. Users can see comments in course detail page.

3. [Search]. Users must be able to search for information on your app. (medium)

    - Class:[SearchActivity.java]
    - Class:[search_course.xml]

    Additional description: This page provides user to seach courses based on different keywords, i.e. course number(`CNO`), course name(`CNAME`), subject(`SUB`) and lecturer(`LEC`). This function is implecated by tokenizer and parser by analyse the input string into different tokens then perform search operation.

4. [Data visualization].Users must be able to load data/information (from the data file(s) or Firebase) and visualise it (e.g., a list of courses and/or a list of lecturers and students as users). (medium)

    - Class:[CourselistFragment]
    - Class:[CourseDao line 31-62]
    - Class:[CourselistFragment]
    - Class:[CourseDao line 31-62]

    Additional description: In our app, we randomly generate some data instances, including course, user, and comment. Those data firstly be writen to local files, then we migrate them to firebase. Course data is able to be loaded from firebase into a RecycleView to display, and when a course is clicked, a detailed page with more information will be displayed, users can see comments be posted in the course detail page. In our app, all courses information and comments can be loaded from firebase and can be visualized using recyclerView.

General Features

Feature Category: Search-related

1. [Search-Invalid] Search functionality can handle partially valid and invalid search queries. (medium)

    - Class CourseParser, methods `public static class IllegalParserException extends IllegalArgumentException`, `public CourseParser(String searchText)`, `public void parseExp()`, `public void parseTerm()`, `public boolean isMatched(Course course)`,Lines of code: [20-57]
    - Class SearchActivity, methods `public boolean isMatched(Course course)`, Lines of code: [158-196]

    Additional description: When user need to search courses, there is a hint in our text box showing `eg. CNO=;CNAME=;LEC=;SUB=;`. When user obey the search rules, right results will be presented below the search box. When user input invalid strings, our `CourseParse.java` class will throw an error, and then there will be a Toast appearing on the screen. Also, we provide fuzzy search, i.e. partially invalid search. This is implemented by substring match using inbuild `contains()` function.

2. [Search-Filter] Sort and/or filter a list of items returned from a search, with the help of suitable UI components. For instance, when searching for assignments, include checkboxes for users to select the target course(s); include drop-down field for the selection of sorting methods, etc. (easy)

    - Class:[Course.java Line 201-226]
    - Class:[SearchActivity.java Line 100-123]

    Additional description: User can sort a list of courses returned from a search, with the help of buttons and radios, For example, when user check "sort by cno" radio, the result list will be sorted according to the course number, and when user check "sort by time" radio, the result will be sorted according to the date of course. Similarly, when user click "^" button, the result will be in ascend order. We implement this by implement comparabl interface and override compareTo() method for Course.class.

Feature Category: UI Design and Testing

3. [UI-Layout] UI must have portrait and landscape layout variants as well as support for different screen sizes. Simply using Android studio's automated support for orientation and screen sizes and or creating support without effort to make them look reasonable will net you zero marks. (easy)

   - class: [portrait] and [landscape]

   Additional description: This app is compatible with multiple models ranging from Pixel 2 to Pixel 6, accommodating various screen sizes. A landscape layout has been designed for each page in addition to the standard layout. Consequently, every page layout file includes both portrait and landscape designs.

4. [UI-Test] UI tests using espresso or similar. Please note that your tests must be of reasonable quality. (hard)

   - Class: [LoginTest.java], lines of code: whole file
   - Class: [RegisterTest.java], lines of code: whole file
   - Class: [SearchTest.java], lines of code: whole file

   Additional description: We tested 3 main functional pages in our UI test, which are *Login page*, *Register page* and *Search page*.

   - For login page, we tested following features:
     - Whether all elements are displayed properly
     - Logic of enabling or disabling Login button and Register button
     - Check inputs, i.e. empty input, well-formed email or not, right and wrong password
   - For register page, we tested following features:
     - Whether all elements are displayed properly
     - Logic of checking checkboxs and enabling of join button in default situation
     - Check inputs, i.e. empty input, well-formed email or not, inconsistent of email and password
   - For register page, we tested following features:
     - Whether all elements are displayed properly
     - Check whether all buttons are clickable
     - Check whether `ListView` is clickable
     - Check input string is valid, i.e. with right tokens, and display properly
     - Check input string is invalid, i.e. with wrong tokens, and display wrong message
     - Check fuzzy search and display results

Feature Category: Greater Data Usage, Handling and Sophistication

5. [Data-Formats] Read data instances from multiple local files in at least 2 different formats (JSON, XML or Bespoken). (easy)

   - Class:[DataGenerator.java]
   - Class: [LocalFileToFirebase.java]

Additional description: We used .xml, .json and .csv formats. The local files are in assets folder, including user.json, course.json, comment.xml, content.csv, subject.csv.

6. [Data-Profile] User profile or Course material activity containing a media file (image, animation (e.g. gif), video). (easy)

    - Class: [Global.java Line 67-124]
    - Class: [MycourseAdapter.java Line 66-70]
    - Class: [CourseHomeActivity.java]

    Additional description: The video shows the course content and assigns a default avatar to each student and course.

7. [Data-GPS] Use GPS information (see the demo presented by our tutors. For example, your app may use the latitude/longitude to show information relevant to your app. An example is to display localised based on the location of the user app. (e.g., English materials if the user is located in an English speaking country)). (easy)

    - Class [LoginActivity]:
        - [private boolean checkLocationPermission()]
        - [public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)]
        - [private Location getLocation()]
        - [private String getAddress(Location location)]
        - [private void setLanguage(String country)]

    Additional description: Check and request location permissions, retrieve the device's last known location, obtain the address corresponding to a location, set the language based on the country. Overall, the code aims to enable location-based functionality and language customization in the application.

8. [Data-Graphical] Graphical report viewer. Provide users with the ability to see a report of interactions with your app (e.g., summary of assessment results for a course or an individual student, etc), in a graphical manner. (medium)

    - Class [DeadlineshowFragment]
    - Class [DeadlineshowViewModel]

    Additional description: 'DeadlineshowFragment' for displaying a calendar and associated to-do lists. The 'DeadlineshowViewModel' class is used to manage the text data related to the calendar.

Feature Category: User Interactivity

9. [Interact-Micro] The ability to micro-interact with items/users (e.g. add to todo-list, like/follow a post in the forum, connect to another user, etc.) [stored in-memory]. (easy)

    - Class:[Comment.java]
    - Class:[CourseHomeActivity.class Line 101-122]

Additional description: User can interact with course by sending comment to the course and reviewing comments of the course. When comment is sent by a user, it will be updated to the firebase, and all other user can see this comment.

10. [Interact-Follow] The ability to ‘follow’ a course or any specific items. There must be a section specifically dedicated to 'things' followed (e.g., showing all updates from all courses followed in chronological order). [stored in-memory] (medium)

    - Class:[CourseHomeActivity.java Line 83-91]
    - Class:[CourseJoinActivity.java Line 93-100]
    - Class:[myCourseFragment.java]

    Additional description: User can enroll and drop a course. When user enrolled a course, the user can send comments about the course and send peer to peer message with users who enrolled the same course. Also, the user can check what courses he has enrolled in "MyCourse" page.

Feature Category: Privacy

11. [Privacy-Visibility] A student can only see a course’s profile that is set to Public (consider that there are at least two types of profiles: public and private) or after being added to the course as a student (easy).

    - Class:[Global.java Line 194-239]

    Additional description: Access to course content, evaluations and chatting with students on the same course is only available to students who are registered for this course. Students can gain these permissions by registering for a course.

Feature Category: Creating Processes

12. [Process-visualise] Process visualisation. Your app may implement a graphical element to visualise the progress of a process (e.g., stages of Group Projects or course completion). (medium)

    - Class [DeadlineshowFragment]
    - Class [DeadlineshowViewModel]

    Additional description: 'DeadlineshowFragment' for displaying a calendar and associated to-do lists. The 'DeadlineshowViewModel' class is used to manage the text data related to the calendar.

Feature Category: Peer to Peer Messaging

13. [P2P-DM] Provide users with the ability to message each other directly in private. (hard)

    - Class:[Message.java]
    - Class:[ChatActivity.java]
    - Class:[ChatAdapter.java]

    Additional description: 'DeadlineshowFragment' is responsible for presenting a calendar and the corresponding to-do lists. The 'DeadlineshowViewModel' class is utilized to handle the text data associated with the calendar.

14. [P2P-Restriction] Enable users to restrict who can message them by some association (e.g. setting: can message me only if we are currently enrolled in the same course). (hard)

    - Class:[ContactListActivity.java]
    - Class:[ContactListApdapter.java]
    - Class:[CourseDao.java Line 75-145]

    Additional description: We display those users who enrolled in the same courses with the login user in the contactList using recyclerView, we get the enroll information and get the courses the login user enrolled and judge which users are in the same courses. When user click on the contact, the app will jump to "chat" page and they can send peer to peer messages.

Feature Category: Firebase Integration

15. [FB-Auth] Use Firebase to implement User Authentication/Authorisation. (easy)

    - Class:[LoginActivity.java Line 363-412]
    - Class:[RegisterActivity.java Line 341-380]

    Additional description: when user sign up or login, their email and password will be stored in firebase authentication.

16. [FB-Persist] Use Firebase to persist all data used in your app. (medium)

    - Class:[LocalFileToFirebase]
    - Class:[UserDao.java Line 93-121]
    - Class:[CourseDao.java Line 31-62]
    - Class:[CommentDao.java Line 36-72]
    - Firebase

    Additional description: All the data are firstly stored in local file and then we migrate them to firebase manually. We use DAO to write and retrieve data from firebase and then feed them into our app. We store all those data except message in firebase storage, and store message data in firebase realtime database. Because we need to update message and notify user in real time, so we store message in realtime database. However, for the other data, we don't need to much update, what we need is querying and sorting, so we used firebase storage to store those data.

17. [FB-Syn] Using Firebase or another remote database to store user information and having the app updated as the remote database is updated without restarting the application. e.g. User A (a lecturer) posts an announcement, user B on a separate instance of the application sees the announcement appear on their app instance without restarting their application. (hard)

    - Class:[ChatActivity.java Line 85-98]

    Additional description: We stored message data in firebase realtime database, since we need those data to be updated and notify users in real time. Using firebase realtime database, we can keep the message updated and notify user in real time. when another user send a message, the user whom the message sent to can receive it without restarting the app.

Feature Category: Others

18. [Multilingual assistive features] To enhance the user experience and expand the audience, multilingualism is set up.
    - Class: [strings.xml(English)]
    - Class: [strings.xml(Japanese)]
    - Class: [strings.xml(French)]
    - Class: [strings.xml(Simplified_Chinese)]
    - Class: [strings.xml(Traditional_Chinese)]

Additional description: We have set up five languages for all the basic functions of the app: English, Japanese, French, Simplified Chinese, and Traditional Chinese. This multilingual setup not only shows our commitment to inclusivity and user experience, but also positions the app for greater reach in international markets.

Team Meeting

Here is an example (you could start numbering your meetings from 1):

- *Team Meeting 1*
- *Team Meeting 2*
- *Team Meeting 3*
- *Team Meeting 4*
- *Team Meeting 5*