

DOKUZ EYLÜL UNIVERSITY
ENGINEERING FACULTY
DEPARTMENT OF COMPUTER ENGINEERING
CME3202 - CONCEPTS OF PROGRAMMING LANGUAGES
FINAL ASSIGNMENT

CENGOnline

by

2400000145 – Mehmet Hamdi Uslu

2016510089 – Ezgi Berfin Şahin

June, 2020

İZMİR

1.CENGOnline

The requirements section contains information that the project has no restrictions when it comes to the preferences of the database and development environment. The choice of which development environment will be used can differ for each group and there are no restrictions on IDE preferences. The project created by this group, that meets the specific requirements expected is created in the form of an android application by using the development environment Android Studio.

In Figure 1.1 it can be seen the activity diagram our app which is a multilayered realtime application. In this paragraph details of those activities will be explained with their workflow on our adapters and activities. On start, user logs in or registers which are setup in Login and Register classes. Those classes use Person abstract, Teacher, and Student classes as models to create objects add database those objects as their respective fields. After login, we get to the Main Activity. This class decides if the user is a Teacher or Student and calls StudentDrawer or Teacher Drawer activity.

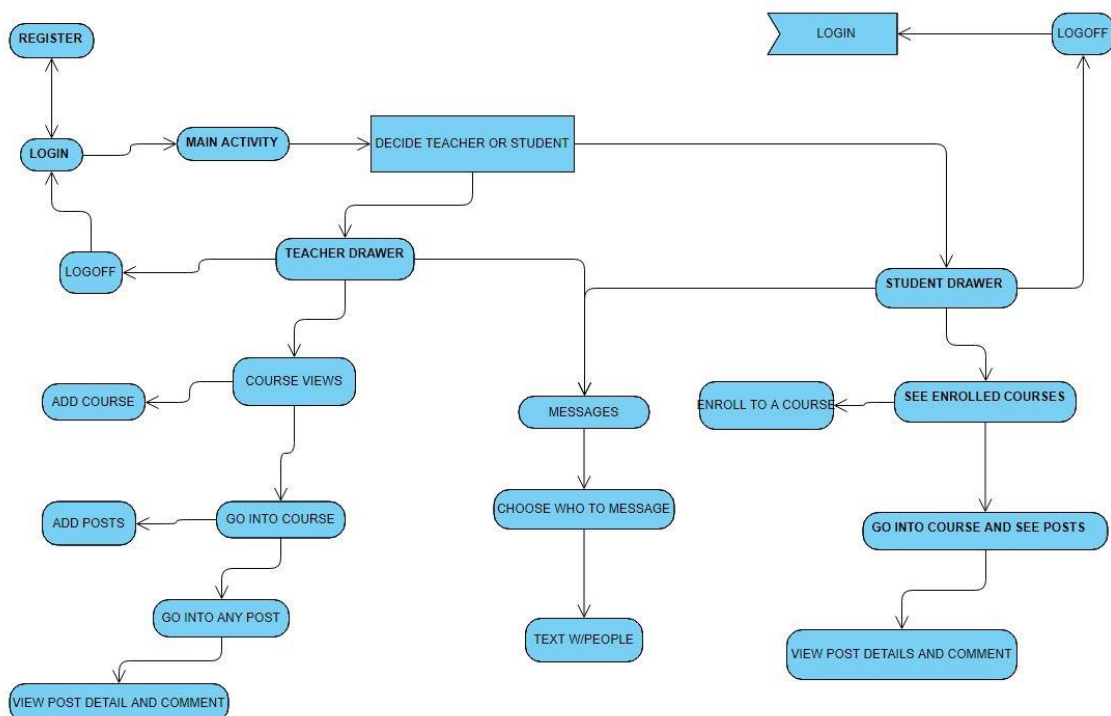


Figure 1.1 Work Flow of the Application

These two activity shares the same chatting interface on the drawer which is contained on MessageFragment. This fragment creates a list of users that do not include the current user that logged in and call ChatAdapter for recycler view. This view showcases all users that are registered as clickable objects. If any person is clicked activity changes to ChatDetailActivity which is the activity that gets messages between two individuals which are logged in user and clicked user. This activity takes all the messages between two and sends it to MessageAdapter. This adapter decides whether the user received the message or sent it and creates a view on that information.

On TeacherDrawer we create navigation drawer according to user credentials. Also we have a floating action button that lets the teacher create courses. Courses are displayed thanks to on HomeFragment_Teacher fragment. This display is a recycler view that can be held any number of created courses. This view of courses created by CourseAdapter that takes the course list and creates views for every course with respective information of that course. If a course is clicked on this fragment will change to PostFragment_Teacher. This fragment also contains a floating action button that gives the opportunity to the teacher to add post to course. Also, all the posts created under this course will be shown on again a recycler view. The adapter of this view is PostAdapter takes a post list of the course and creates views for every post with respective information of that post. In this view if a post clicked app will call PostDetailActivity, which showcases data of that post and attaches a recycler view at the bottom that lets everyone comment or read comments. The adapter of that is the CommentAdapter.

On StudentDrawer things are nearly the same but the functionality of the floating action button is different. This button lets students join courses with course codes like on google classroom. In this activity courses are displayed with HomeFragment fragment. Which checks all the classes student is enrolled to and displays those classes. This view again calls CourseAdapter that takes the course list and creates views for every course with respective information of that course. If a course is clicked on this fragment will change to PostFragment_Teacher but in this case students can't add posts. Only sees them and their details. The adapter of this view is PostAdapter that takes a post list of the course and creates views for every post with respective information of that post. In this view if a post clicked app will call PostDetailActivity, which showcases data of that post and attaches a recycler view at

the bottom that lets everyone comment or read comments. The Adapter of that is the CommentAdapter.

2.Data Repository

Several databases can be used to store the mobile application's content. However, after comparisons of the databases based on their features and seeing each database's cons and pros the database Firebase is picked because of the fact of its convenience.

2.1Firebase

Firebase, is a platform by Google that provides functionalities and helps with the backend development of Android, iOS, web, and Unity3D based applications. Firebase manages its own infrastructure with a variety of tools and services to simplify the workflow of the developer by providing them with development kits and an online dashboard. Firebase is categorized as a NoSQL database program, which stores data in JSON-like documents. In Figure 2.1 the comparison between Firebase and the traditional databases can be seen.

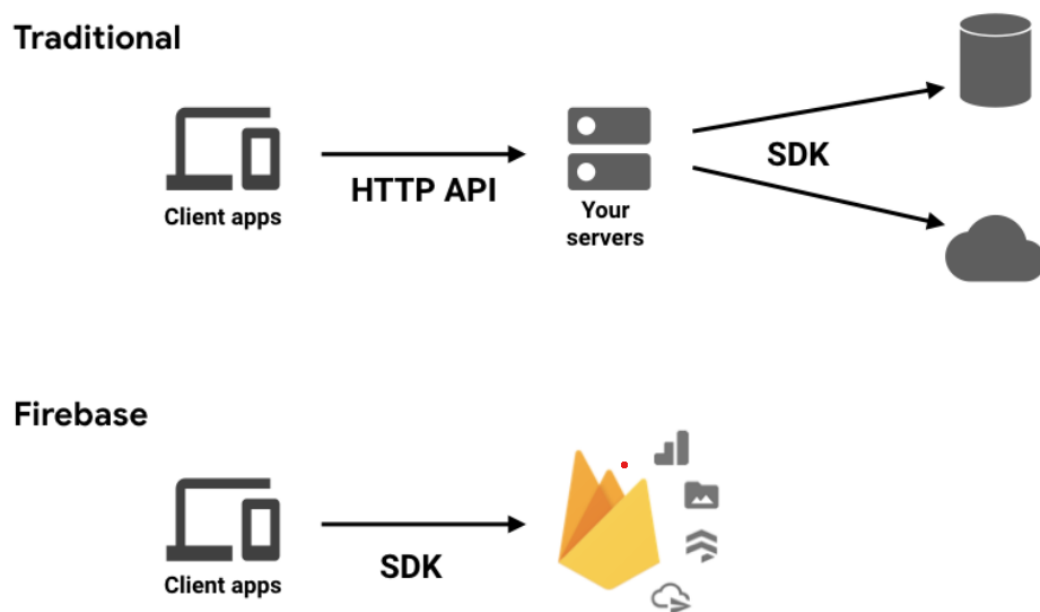


Figure 2.1 Comparison of the Databases

Applications that use Firebase can use and control data without thinking about how data is stored and synchronized across different instances of the application in real-time. This behavior is the main reason for Firebase to be database of the project. Since, Firebase is a realtime -

database that feature gives developers the opportunity to synchronize all the data across all clients in realtime and remains available even when an app goes offline.

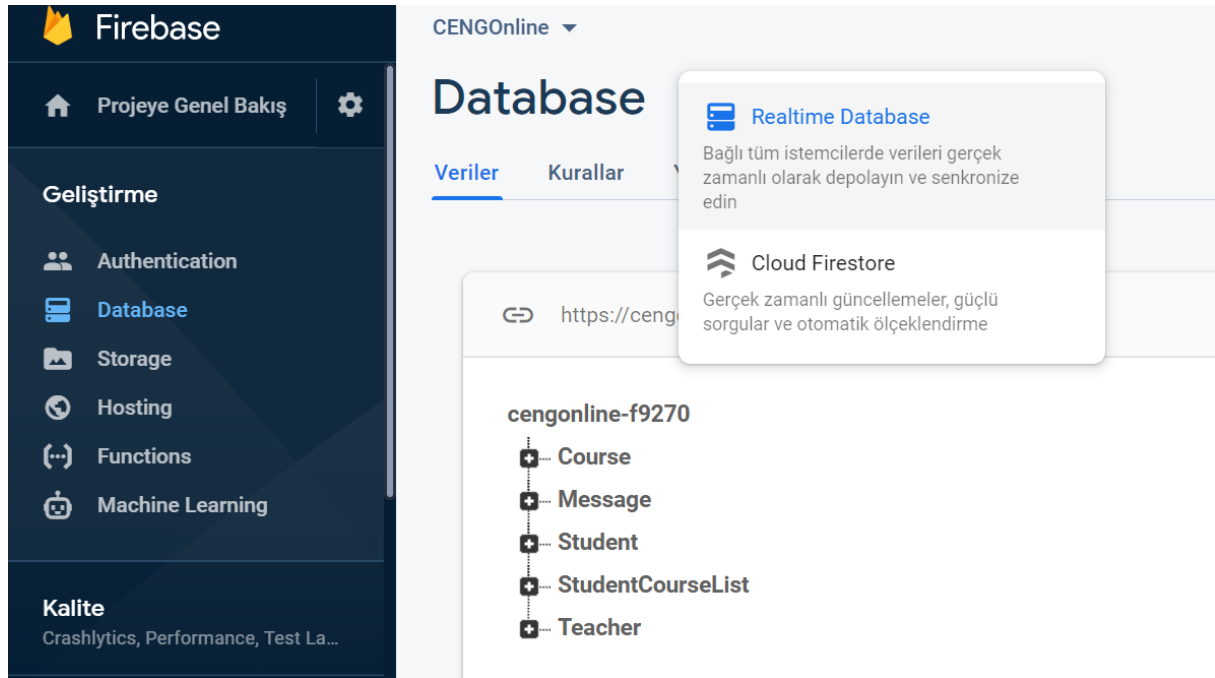


Figure 2.2 Database Structure of the Project “CENGOnline”

The figures Figure 2.3, Figure 2.4, Figure 2.5, Figure 2.6, and Figure 2.7 are some examples from the database after users, post and course created and comments sent.

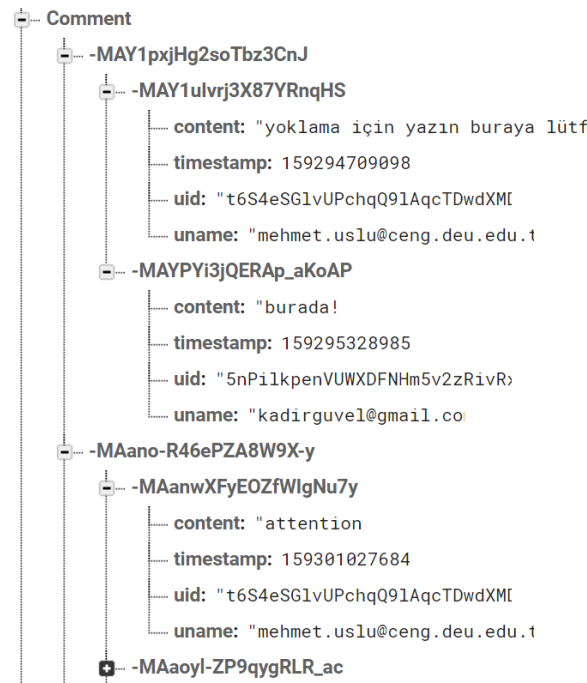


Figure 2.3 Comment Branch



Figure 2.4 Course Branch

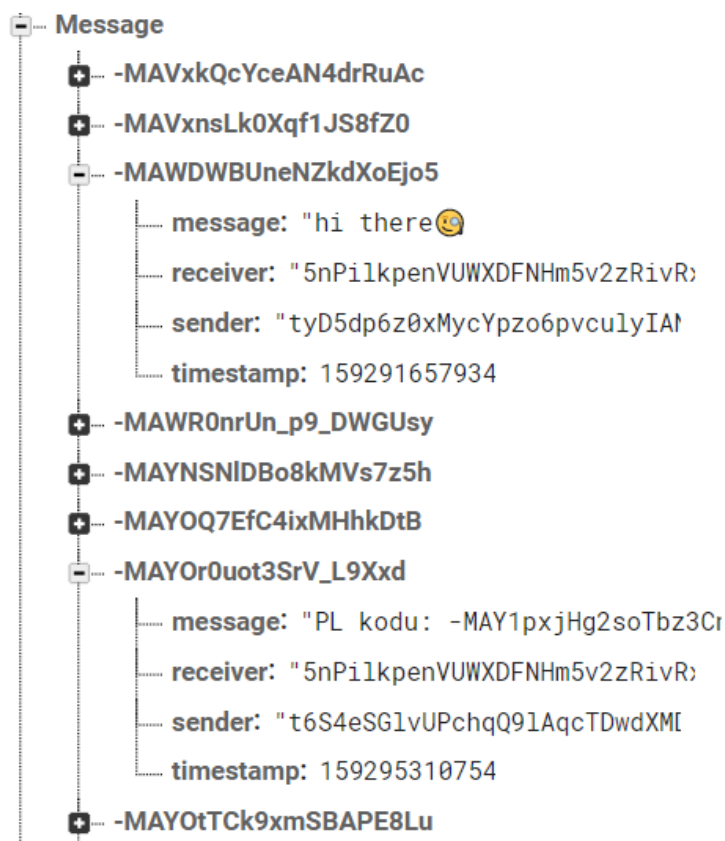


Figure 2.5 Message Branch

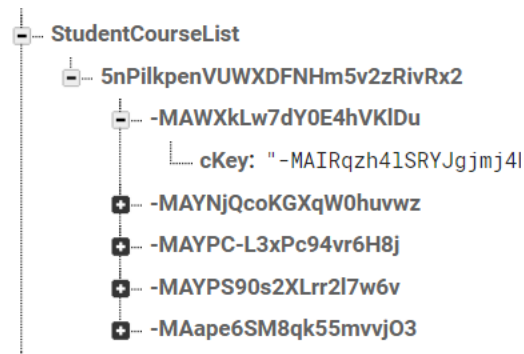


Figure 2.6 StudentCourse Branch



Figure 2.7 Teacher Branch

3. IDE

Both members of the project group are more familiar with the two development environments Android Studio and Eclipse comparing to the other IDEs. This fact narrowed down the numbers of the IDE which can be picked, to two. However, the user interface and the

design of the application can be developed in many various ways and UI may be seen a lot more attractive if the IDE picked is Android Studio. In addition to these, the database operations are a lot easier between the pair of Firebase and Android Studio. Thus, Android Studio is picked based on these two main features.

3.1 Android Studio

Android Studio is the integrated development environment for Google's Android platform. Versions of Android Studio are compatible with some Apple, Windows and Linux operating systems. The project is expected to form in the OOP approach and in Android Studio the classes and other layers can be created and found in future use easily. The Android SDK version used in this project is android sdk 29.0.3.

The application tested in each step on two devices simultaneously for creating a more bug-free application. The emulator used for running the application is NEXUS 5X API R and the physical device used is Xiaomi Mi A3. In Figure 3.2 details about the emulator can be seen. Xiaomi Mi A3 runs on the Android version "X" which is the latest android version on the market.



Type	Name	Play Store	Resolution	API	Target	CPU/ABI
	Nexus 5X API R		1080 × 1920: 420dpi	R	Android API 30 (Google Play)	x86

Figure 3.2 NEXUS 5X API R Details

4.Libraries

Several libraries useful for different functions from design to operational purposes are added. The frequency of the libraries' usage can differ for each of the individual libraries. Some of the libraries are used almost in every class while some of them only used in one class.

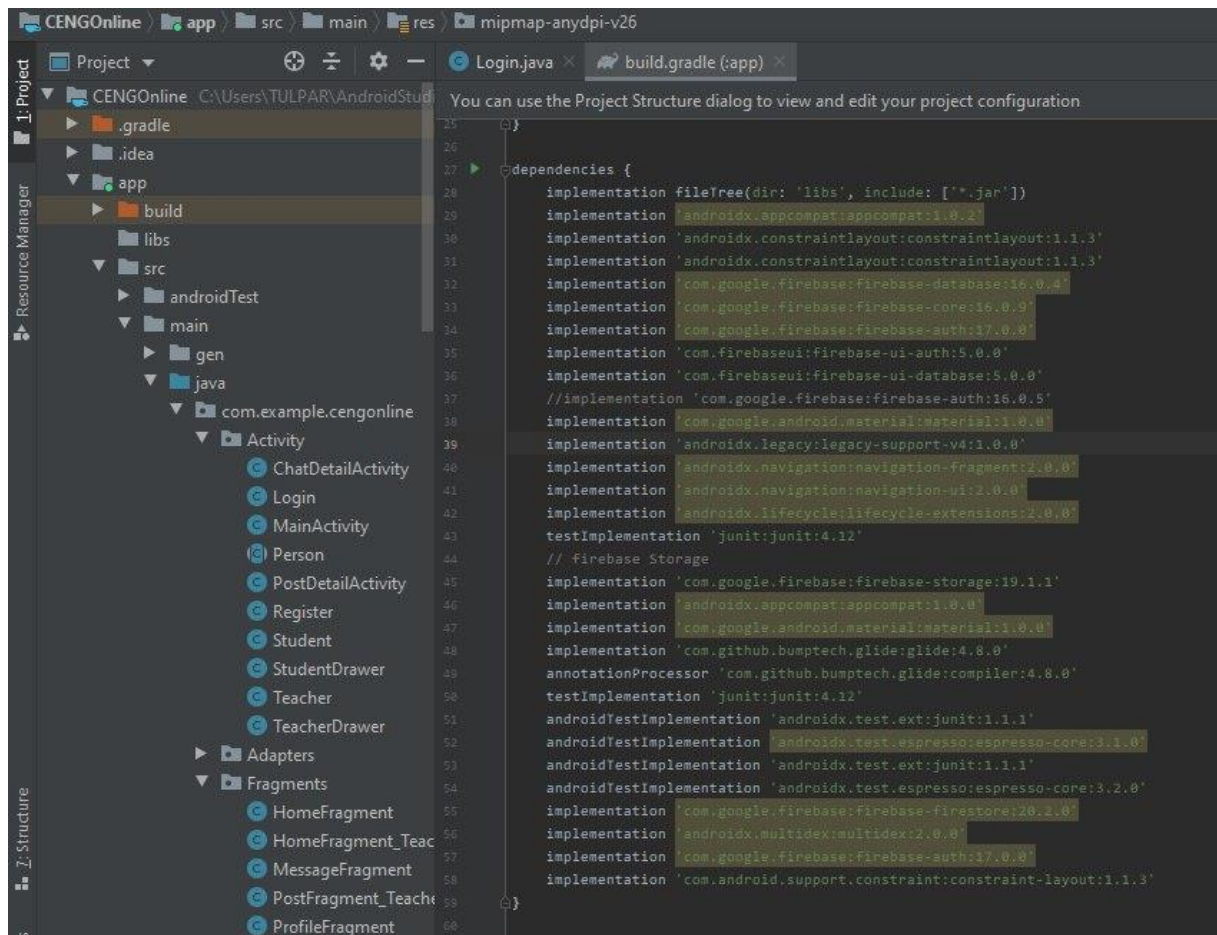


Figure 4.1 Libraries of the Project

FirebaseAuth method registers a listener to changes in the user authentication state. There can be more than one listener registered at the same time for one or more FirebaseAuth instances. This library contains two functions "createUserWithEmailAndPassword" and "signInWithEmailAndPassword". The method FirebaseAuth represents a user's profile information in the Firebase project's user database. It also contains helper methods to change or retrieve profile information, as well as to manage that user's authentication state. These two libraries are the most used functions in general. These two methods are imported to a hugely significant part of the project. In Figure 4.1 all the methods for database operations are listed. Most used database operations in the project are retrieving data from the database, writing data to the database, and managing the connection between the application and the database.

```
18 import com.google.firebase.auth.FirebaseAuth;
19 import com.google.firebase.auth.FirebaseUser;
20 import com.google.firebase.database.DataSnapshot;
21 import com.google.firebase.database.DatabaseError;
22 import com.google.firebase.database.DatabaseReference;
23 import com.google.firebase.database.FirebaseDatabase;
24 import com.google.firebase.database.ValueEventListener;
```

Figure 4.2 Database Operations Methods

In Figure 4.3 the methods and that are needed for XML file Layout attributes are listed. TextUtils is mostly used for the function “isEmpty” in the project. View is class represents the basic building block for user interface components. The view is used to create user interface components like buttons and text fields.

```
import android.text.TextUtils;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
```

Figure 4.3 XML File Layout Attributes

5. Class

The application contains way too many classes and at some point it gets complicated and messy to understand the connection between each other. In order to prevent this confusion, several packages are created and each class is divided into these packages based on each classes' purpose. In Figure 5.1 the packages named "Activity", "Adapters", "Fragments", and "Models" can be seen.

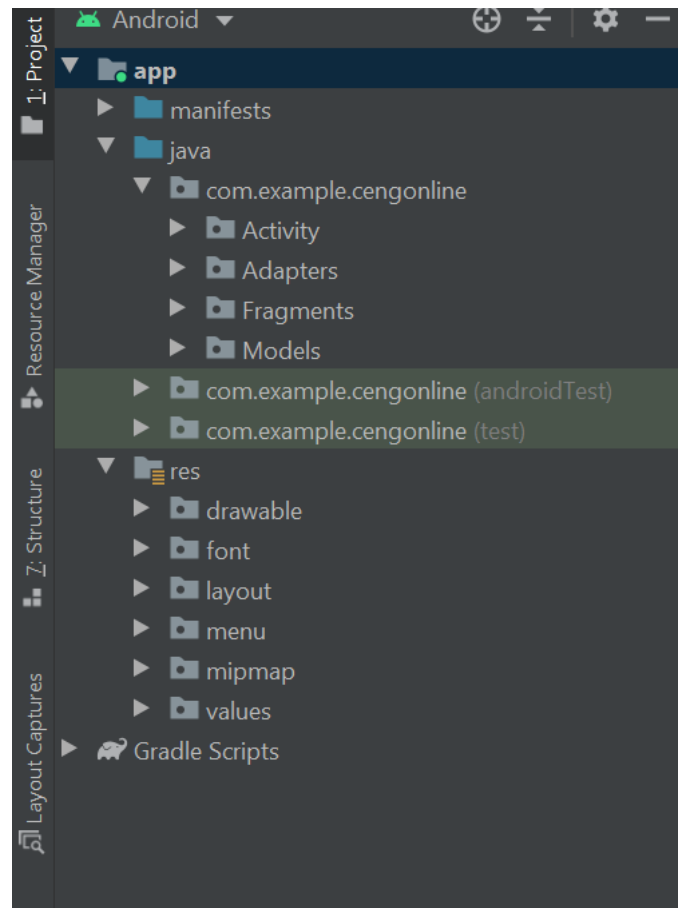


Figure 5.1 Class Packages

In Figure 5.2 overall look of the classes and the XML layout files can be seen side by side. The activity package contains 8 classes. Alongside these activity classes, there are 5 Adapters classes, 5 Fragment classes, and 6 Models classes in the application.

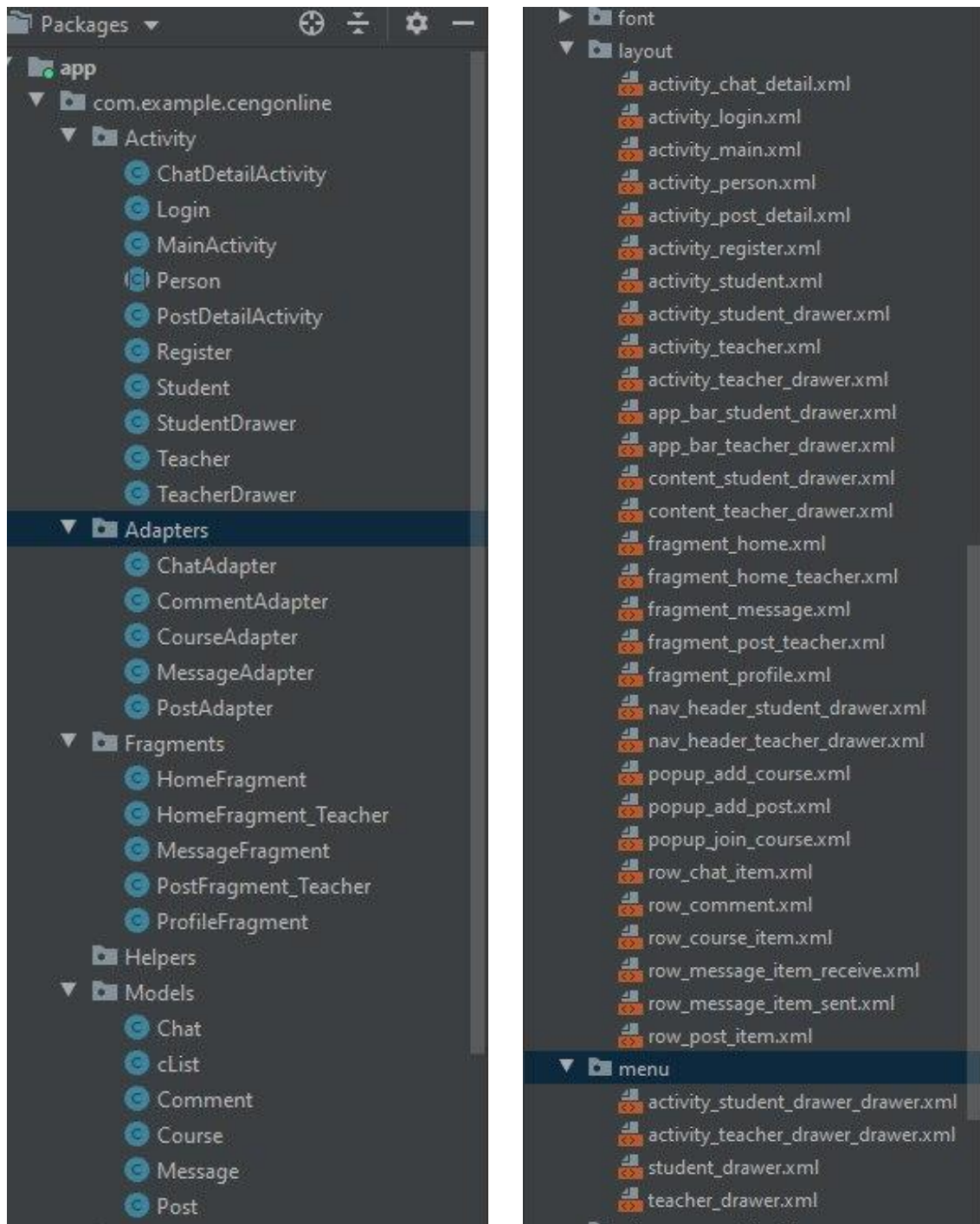


Figure 5.2 Classes and XML Layout Files

5.1 Main Activity

This class is the initial class when a user logs into the application. The "userType " attribute filled while the user creates an account is used in this MainActivity class. The user is lead to either home page of the teacher or home page of a student by retrieving the current user's type from the database. This class acts as a bridge from login to several other classes. The XML layout file to match this class is named "activity_main.xml".

This class only contains two features to display because of the fact that this class mainly contains backend activities. This layout class contains an image view to display the logo generated from the combination of the computer engineering department logo and Google Classroom logo. A text view that says "Welcome to CENGOnline" is also at this layout class.



Figure 5.3 Creation of the Logo

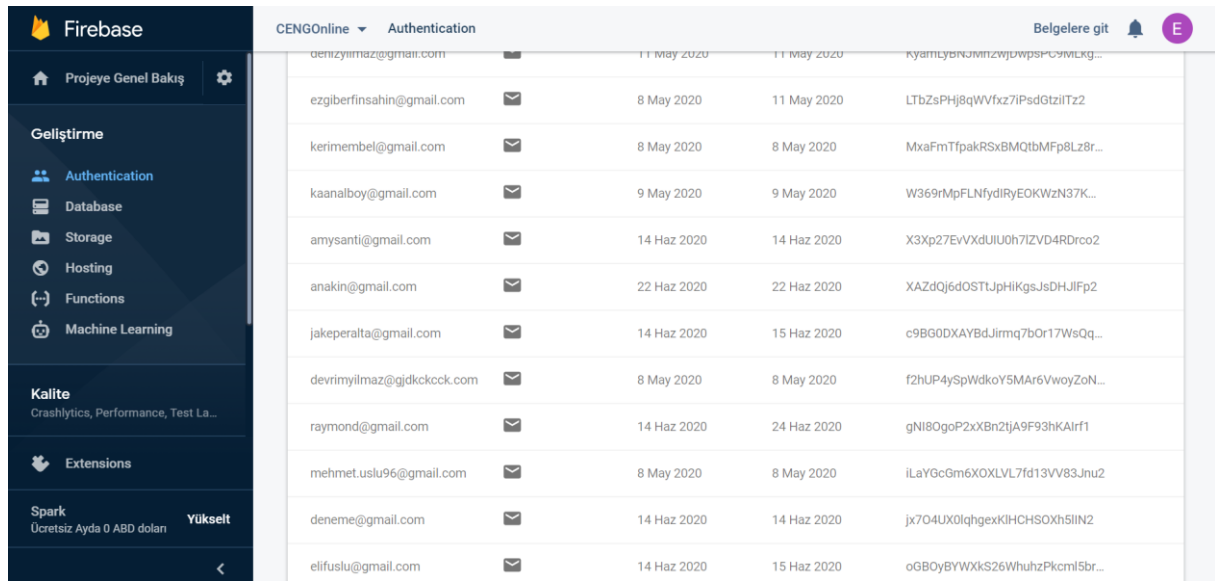
5.2 Login

The initial class when the application launched is the login page. The login page also showed after the user sign-outs from the application. This class contains only one crucial named "loginFunction". Based on the user type attribute retrieved from the database after checking if all the conditions correct the user leads to the specific home page of user's own.

The XML layout file of this class named "activity_login.xml" contains the logo and the name of this application. Alongside these two text views and image view, 2 buttons and 2 text bars are exist in this class. One of the buttons leads to the registration page of the application. This page is needed and used only if the user does not have an account in the application. The other button named login leads to specific home page based on the user type. The two text bars are editable. Thus, the error message in the form of Toast is showed at the screen even one of the edit text fields is empty.

5.3 Register

This class is created for the usage of creating an account if the user does not have an account already in the application. The register button should be pressed after filling the areas about the information of the user. Unless specified areas for both email and password are not empty, by using a switch case condition student or teacher is created based on user type of current user. In Figure 5.4 some of the authenticated users can be seen from the Firebase.



The screenshot shows the Firebase Authentication console. On the left is a sidebar with navigation options: 'Projeye Genel Bakış', 'Geliştirme' (with sub-options: Authentication, Database, Storage, Hosting, Functions, Machine Learning), 'Kalite' (with sub-options: Crashlytics, Performance, Test Lab), and 'Extensions'. The main area displays a table of authenticated users under the 'Authentication' tab. The table has columns for email, a status icon, and two dates. The data is as follows:

Email	Status	Created At	Last Signed In At
denizyilmaz@gmail.com	✓	11 May 2020	11 May 2020
ezgiberfinsahin@gmail.com	✉	8 May 2020	11 May 2020
kerimembel@gmail.com	✉	8 May 2020	8 May 2020
kaanailboy@gmail.com	✉	9 May 2020	9 May 2020
amysanti@gmail.com	✉	14 Haz 2020	14 Haz 2020
anakin@gmail.com	✉	22 Haz 2020	22 Haz 2020
jakeperalta@gmail.com	✉	14 Haz 2020	15 Haz 2020
devrimyilmaz@gjdkckock.com	✉	8 May 2020	8 May 2020
raymond@gmail.com	✉	14 Haz 2020	24 Haz 2020
mehmet.uslu96@gmail.com	✉	8 May 2020	8 May 2020
deneme@gmail.com	✉	14 Haz 2020	14 Haz 2020
elifuslu@gmail.com	✉	14 Haz 2020	15 Haz 2020

Figure 5.4 Authenticated Users

The XML layout file of this class named "activity_register.xml" contains the logo and the name of this application. Alongside these two text views and image view, 2 buttons and 5 text bars are exist in this class. One of the buttons leads to the login page of the application. This page is needed and used only for leading the user back to the initial page which is login. Users should login after the registration by filling email and password as usual. the if the user does not have an account in the application. Also, the other button named go back to login leads to the login page. However, the difference between these two buttons is that if the user clicks on the button "go back to login" users can go back to login by not filling the information about themselves while the register button requires filling the blank text areas. The five text bars are editable. However, the error message in the form of Toast is showed at the screen if the email and password fields are empty.

5.4 Person, Student & Teacher

The class Person is an abstract class. The two classes "Student" and "Teacher" are extended from this Person class. The attributes and methods in the class "Person" are overridden in both classes. Based on the type of the user a student or a teacher is created in the form of these two classes in the register part of the application.

```
case "student":
    String emailStudent = FirebaseAuth.getCurrentUser().getUid();
    DatabaseReference currentStudent = referenceStudent.child(emailStudent);
    currentStudent.child("UserKey").setValue(emailStudent);
    currentStudent.child("Name").setValue(name);
    currentStudent.child("Surname").setValue(surname);
    currentStudent.child("Email").setValue(email);
    currentStudent.child("Password").setValue(password);
    currentStudent.child("UserType").setValue(studentOrTeacher);
    startActivity(new Intent(getApplicationContext(), Login.class));
    Toast.makeText(context, Register.this, text: "Student Inserted To The Database.", Toast.LENGTH_LONG).show()
    updateUI();
    break;
case "teacher":
    String emailTeacher = FirebaseAuth.getCurrentUser().getUid();
    DatabaseReference currentTeacher = referenceTeacher.child(emailTeacher);
    currentTeacher.child("UserKey").setValue(emailTeacher);
    currentTeacher.child("Name").setValue(name);
    currentTeacher.child("Surname").setValue(surname);
    currentTeacher.child("Email").setValue(email);
    currentTeacher.child("Password").setValue(password);
    currentTeacher.child("UserType").setValue(studentOrTeacher);
    startActivity(new Intent(getApplicationContext(), Login.class));
    Toast.makeText(context, Register.this, text: "Teacher Inserted To The Database.", Toast.LENGTH_LONG).show()
    updateUI();
    break;
```

Figure 5.5 Create Student / Teacher

5. Teacher Drawer

Teacher Drawer class is based on the drawer that pop-ups on the left side of the screen. On top of the drawer, name of the teacher, email address, and a stock image used for declaring that the user of the account is a teacher are placed. The floating action button for adding a course is placed on the right-below corner of the home page of the teacher. A signout function is also declared at this navigation bar. Teachers can log out by clicking on the text "Sign Out" on the drawer and lead to the login page.

Courses are displayed thanks to on HomeFragment_Teacher fragment. This display is a recycler view that can be held any number of created courses. This view of courses created by CourseAdapter that takes the course list and creates views for every course with respective information of that course. If a course is clicked on this fragment will change to PostFragment_Teacher. This fragment also contains a floating action button that gives the opportunity to the teacher to add post to course. Also, all the posts created under this course will be shown on again a recycler view. The adapter of this view is PostAdapter takes a post list of the course and creates views for every post with respective information of that post. In this view if a post clicked app will call PostDetailActivity, which showcases data of that post and attaches a recycler view at the bottom that lets everyone comment or read comments. The adapter of that is the CommentAdapter.

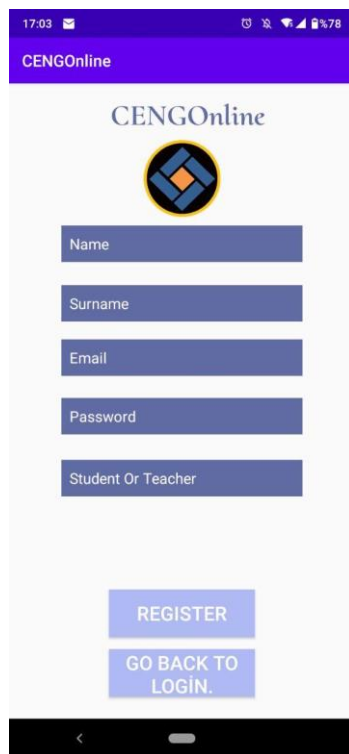
5.6.Student Drawer

The student Drawer class is based on the drawer that pop-ups on the left side of the screen. On top of the drawer, name of the student, email address, and a stock image used for declaring that the user of the account is a student are placed. The floating action button for adding a course to the encountered classes is placed on the right-below corner of the home page of the student. A signout function is also declared at this navigation bar. Students can log out by clicking on the text "Sign Out" on the drawer and lead to the login page.

On StudentDrawer things are nearly the same but the functionality of the floating action button is different. This button lets students join courses with course codes like on google classroom. In this activity courses are displayed with HomeFragment fragment. Which checks all the classes student is enrolled to and displays those classes. This view again calls CourseAdapter that takes the course list and creates views for every course with respective information of that course. If a course is clicked on this fragment will change to PostFragment_Teacher but in this case students can't add posts. Only sees them and their details. The adapter of this view is PostAdapter that takes a post list of the course and creates views for every post with respective information of that post. In this view if a post clicked app will call PostDetailActivity, which showcases data of that post and attaches a recycler view at the bottom that lets everyone comment or read comments. The Adapter of that is the CommentAdapter.

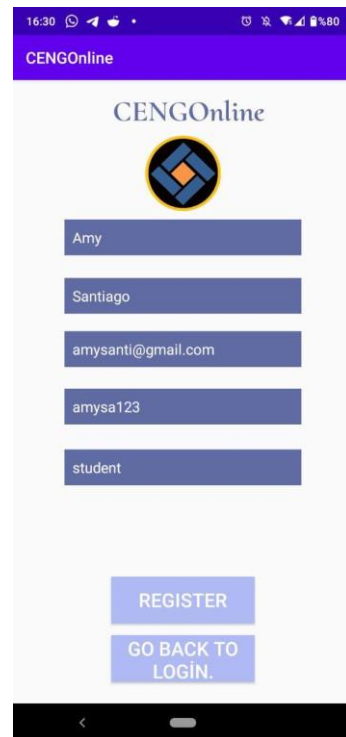
6.UI

6.1 Register



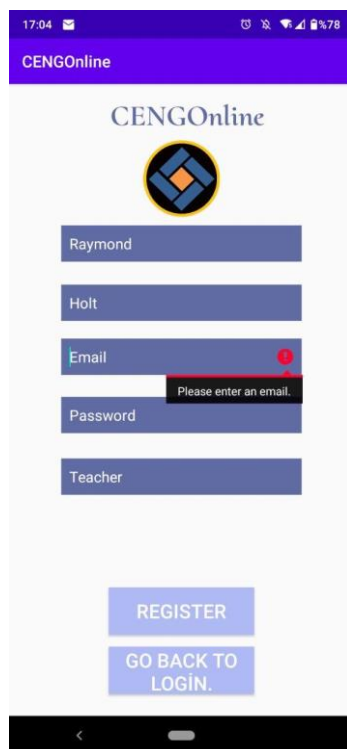
A mobile app interface for the 'CENGOnline' registration page. The header is a purple bar with the time '17:03' and status icons. Below the header, the 'CENGOnline' logo is centered. The form consists of five stacked input fields: 'Name', 'Surname', 'Email', 'Password', and 'Student Or Teacher'. At the bottom, there are two buttons: 'REGISTER' and 'GO BACK TO LOGIN.'.

Figure 6.1 Blank Register Page

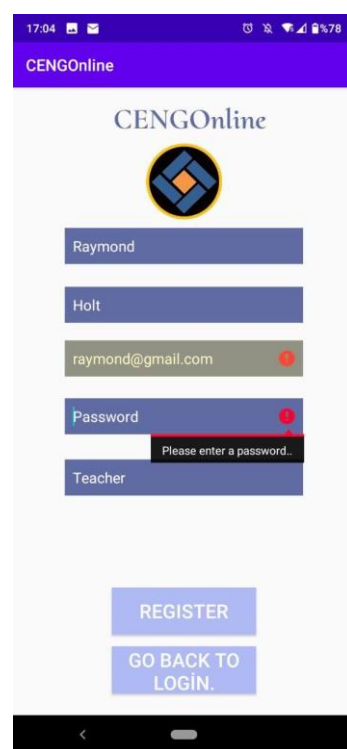


A mobile app interface for the 'CENGOnline' registration page, showing the form filled with sample data. The header is a purple bar with the time '16:30' and status icons. Below the header, the 'CENGOnline' logo is centered. The form consists of five stacked input fields: 'Name' (Amy), 'Surname' (Santiago), 'Email' (amysanti@gmail.com), 'Password' (amysa123), and 'Student Or Teacher' (student). At the bottom, there are two buttons: 'REGISTER' and 'GO BACK TO LOGIN.'.

Figure 6.2 Filled Register Page



A mobile app interface for the 'CENGOnline' registration page, showing an error message. The header is a purple bar with the time '17:04' and status icons. Below the header, the 'CENGOnline' logo is centered. The form consists of five stacked input fields: 'Name' (Raymond), 'Surname' (Holt), 'Email' (raymond@gmail.com), 'Password' (Please enter an email.), and 'Teacher'. A red error message 'Please enter an email.' is displayed below the 'Email' field. At the bottom, there are two buttons: 'REGISTER' and 'GO BACK TO LOGIN.'.



A mobile app interface for the 'CENGOnline' registration page, showing an error message. The header is a purple bar with the time '17:04' and status icons. Below the header, the 'CENGOnline' logo is centered. The form consists of five stacked input fields: 'Name' (Raymond), 'Surname' (Holt), 'Email' (raymond@gmail.com), 'Password' (Please enter a password.), and 'Teacher'. A red error message 'Please enter a password.' is displayed below the 'Password' field. At the bottom, there are two buttons: 'REGISTER' and 'GO BACK TO LOGIN.'.

Figure 6.3 Error Messages of the Register Page

6.2 Login

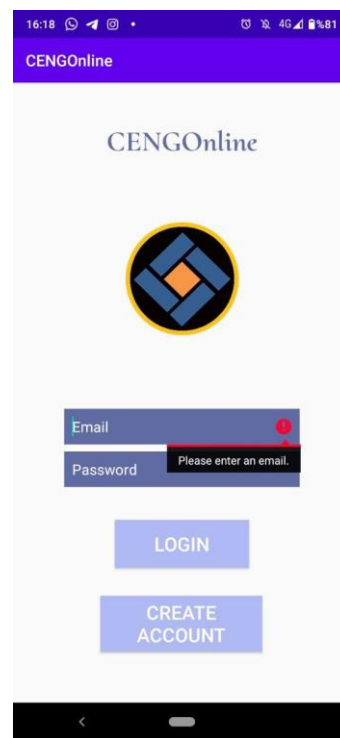
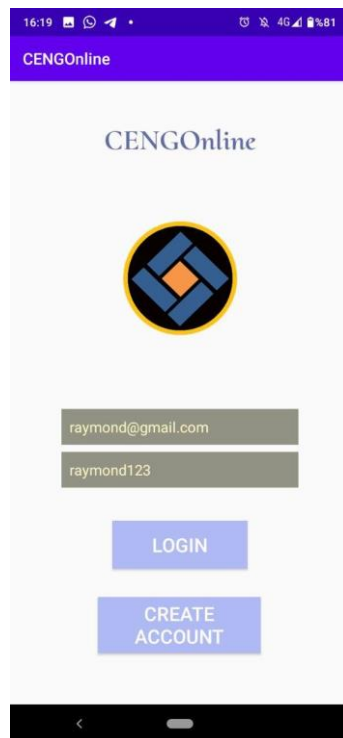


Figure 6.4 Login Page of the Application

Figure 6.5 Error Messages of the Login Page

6.3 Welcome Page

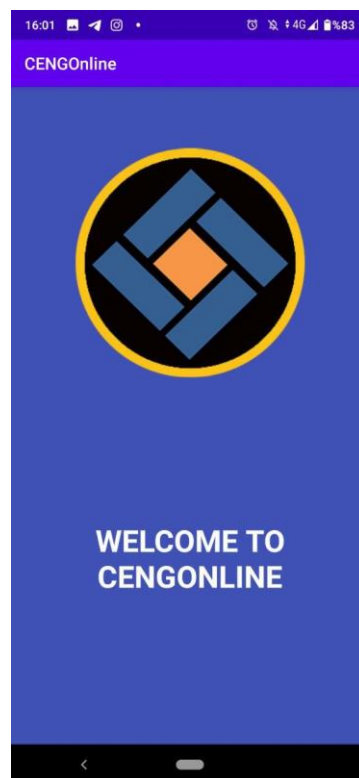


Figure 6.6 Welcome Page

6.4 Student Drawer / Teacher Drawer

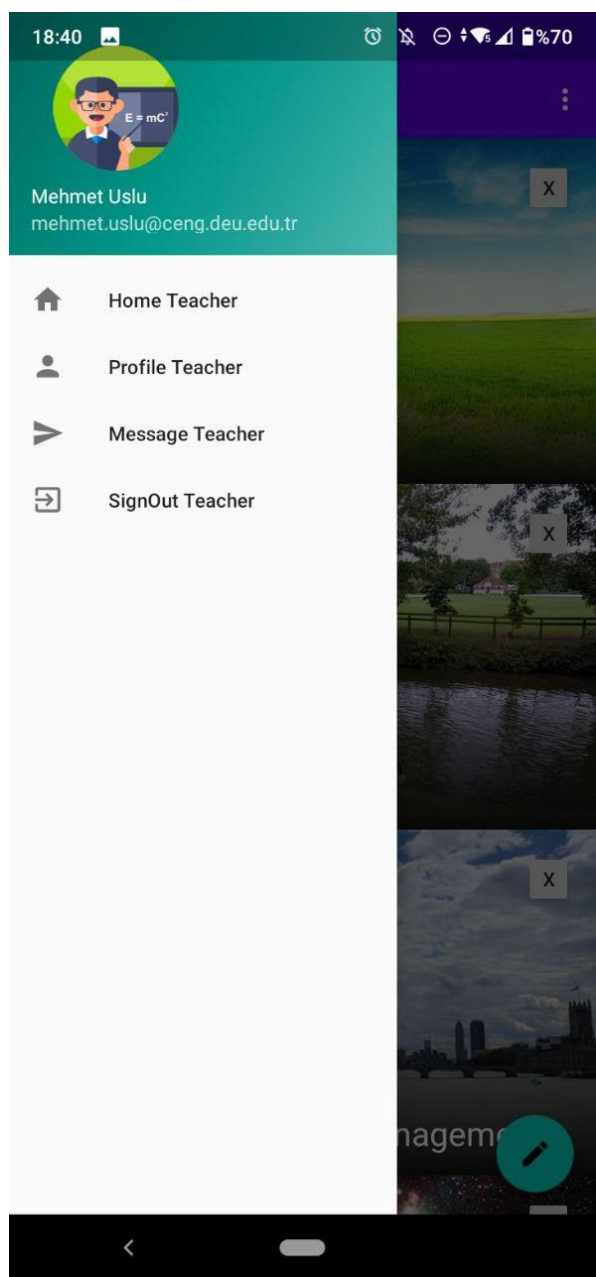
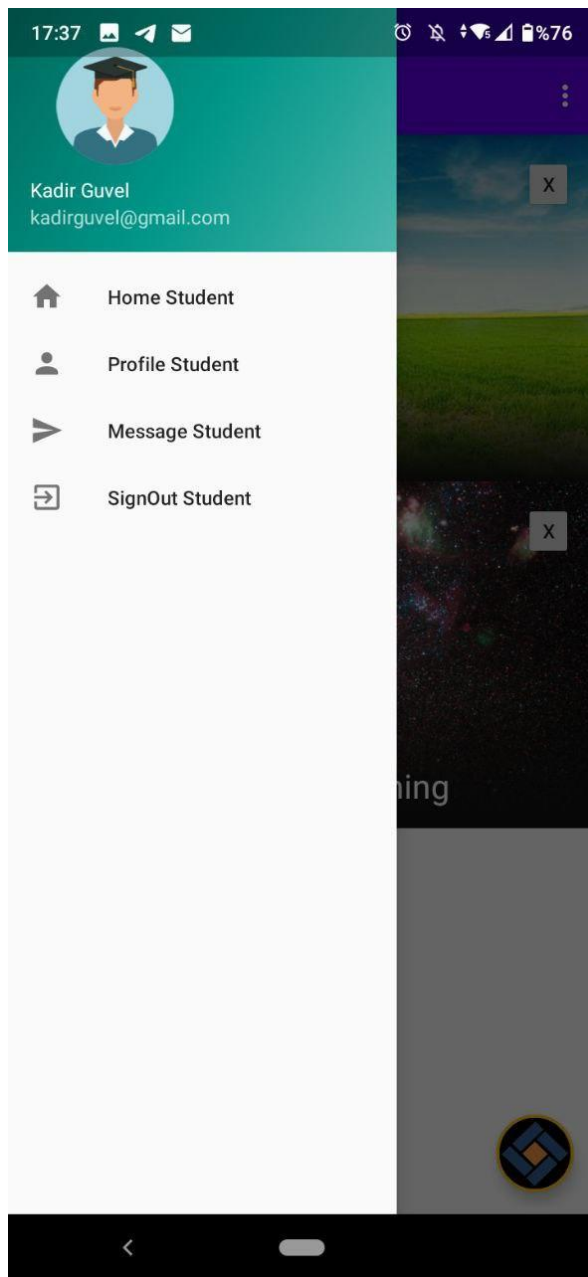


Figure 6.7 Student Navigation Bar Figure 6.8 Teacher Navigation Bar

6.5 Teacher Home Page

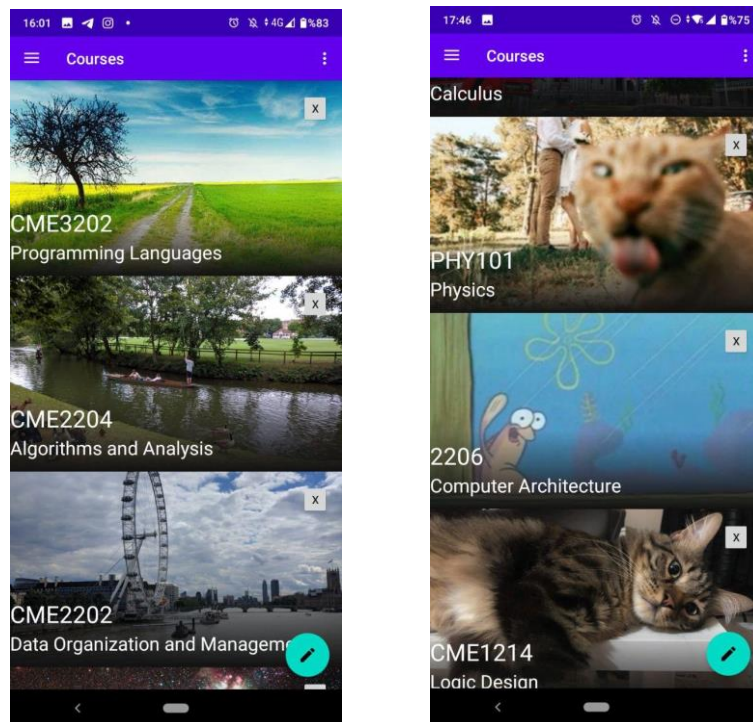


Figure 6.9 Home Page of the Teacher

6.6 Posts & Streaming

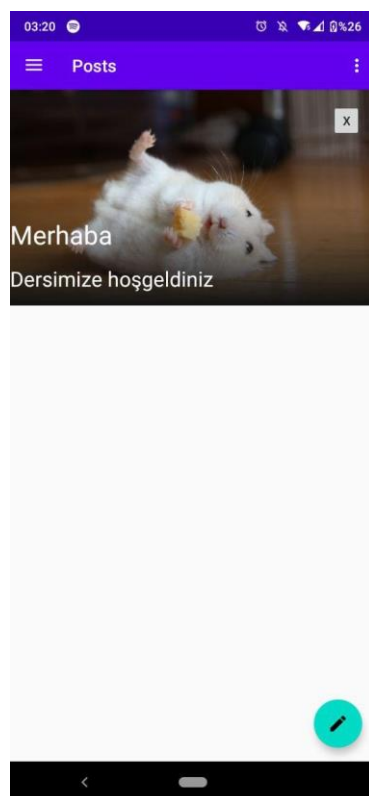


Figure 6.10 Post Example

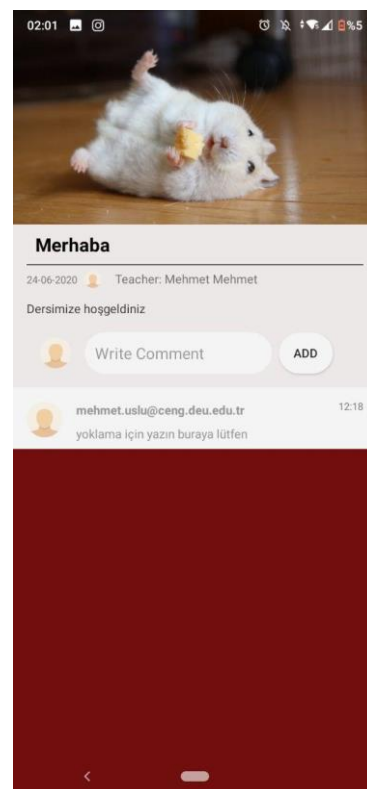


Figure 6.11 Teacher Commenting to a Post

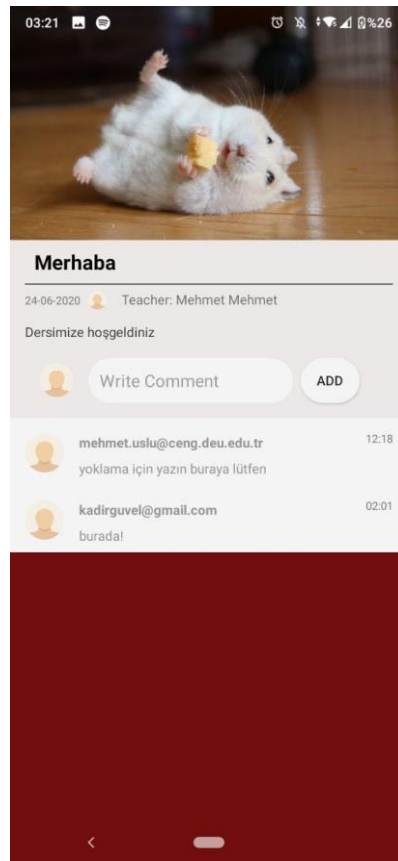


Figure 6.11 Stundent Sending a Comment to a Post

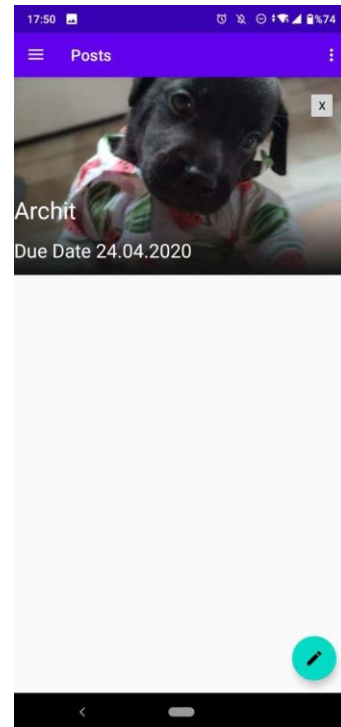
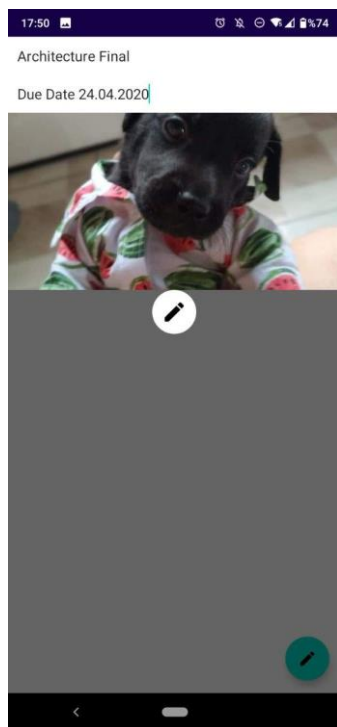
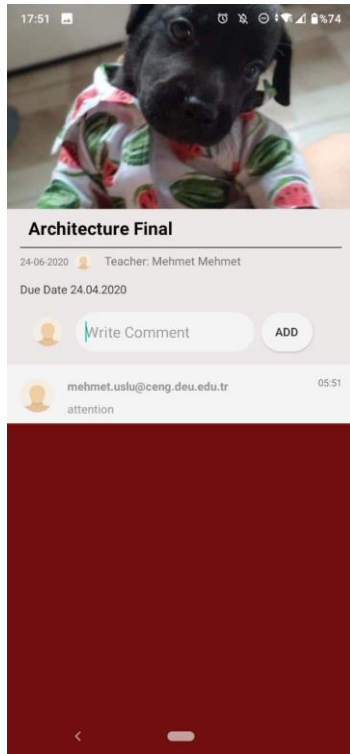
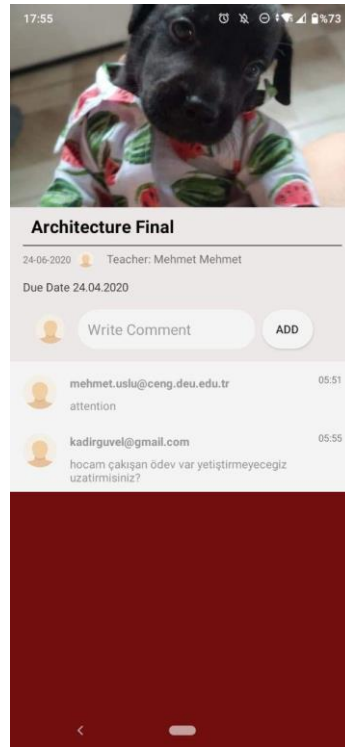


Figure 6.12 Teacher Adding a Post to a Course

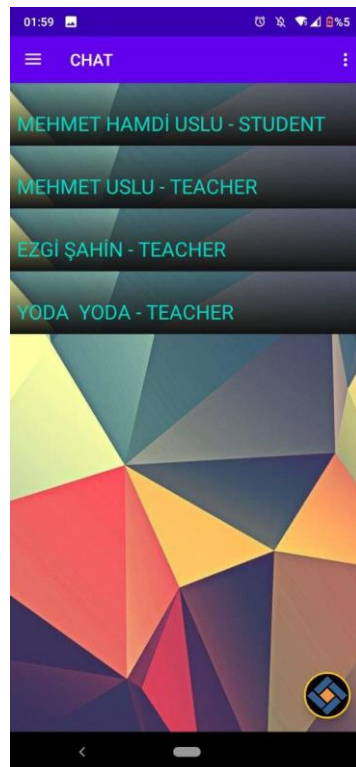


**Figure 6.13 Teacher Adding
Comment to a Post**



**Figure 6.14 Student Adding
Comment to a Post**

6.7 Message Page



6.15 List of People to Message With

6.8 Studing Add Course

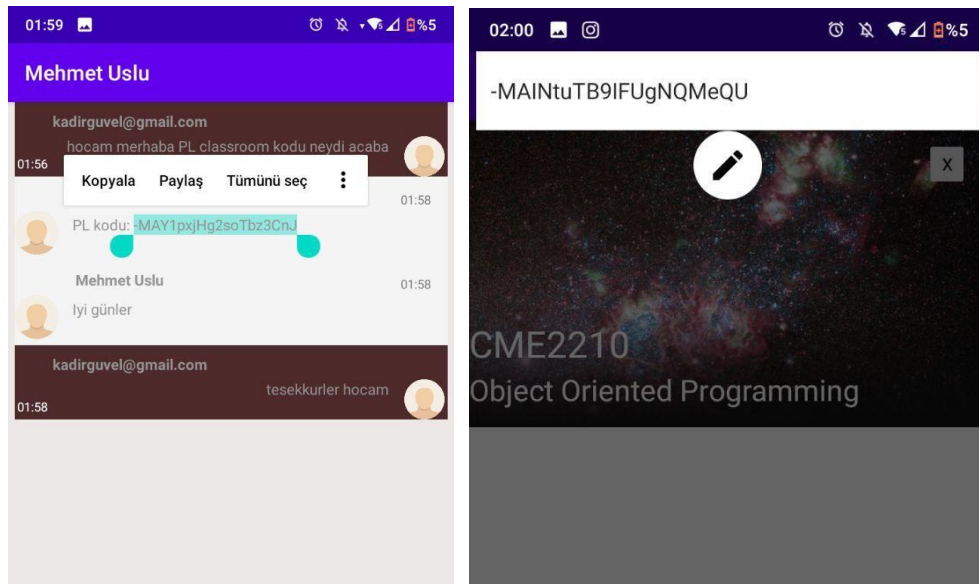


Figure 6.16 Adding Course With a Course Key

6.9 Teacher Add Course

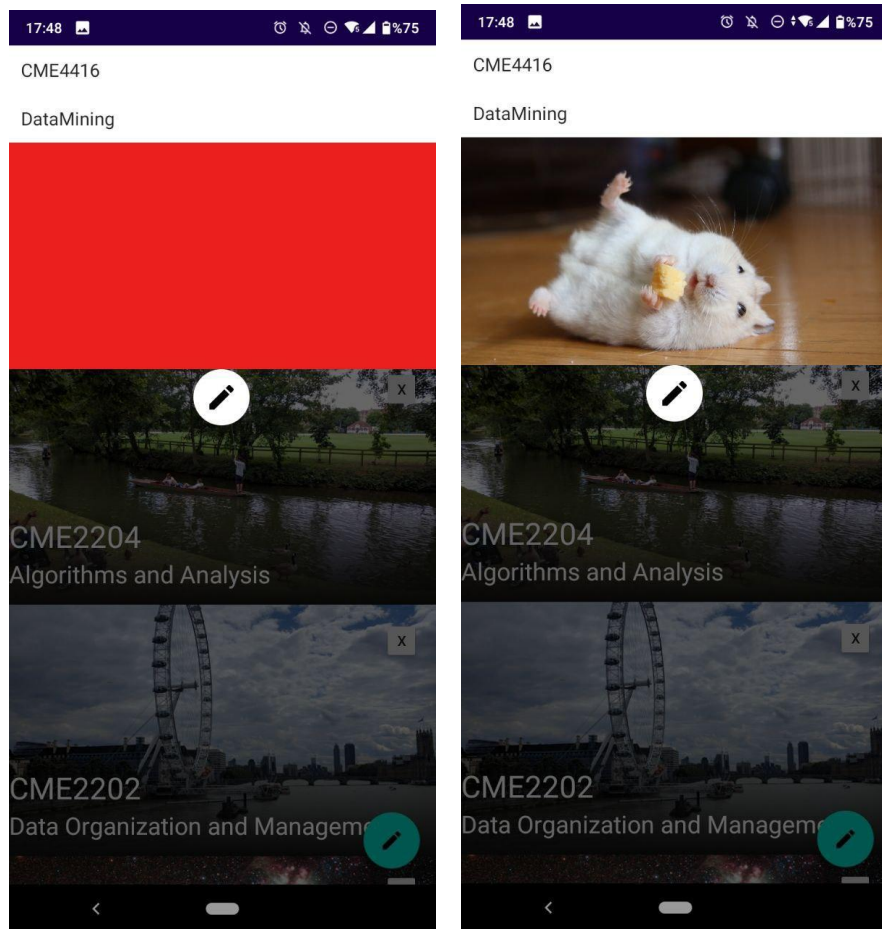


Figure 6.17 Adding Course Figure 6.18 Picking Course Image from Phone Gallery

6.10 Chat Screen Between Student & Teacher

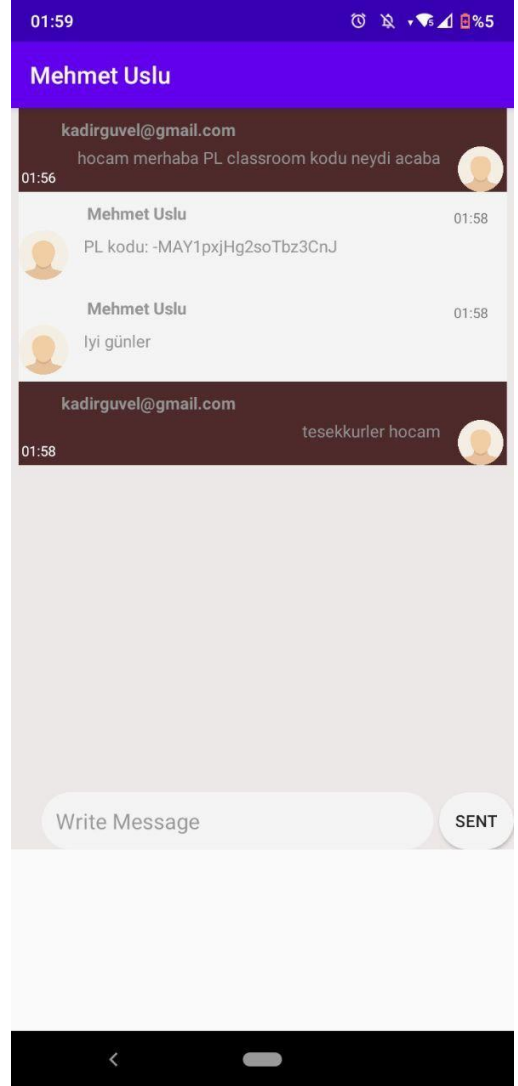
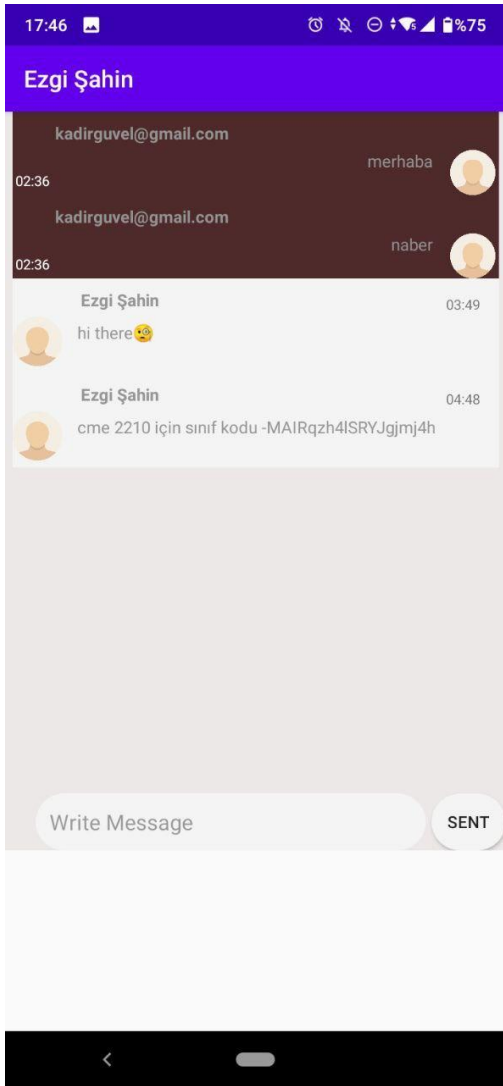


Figure 6.19 Chat Example

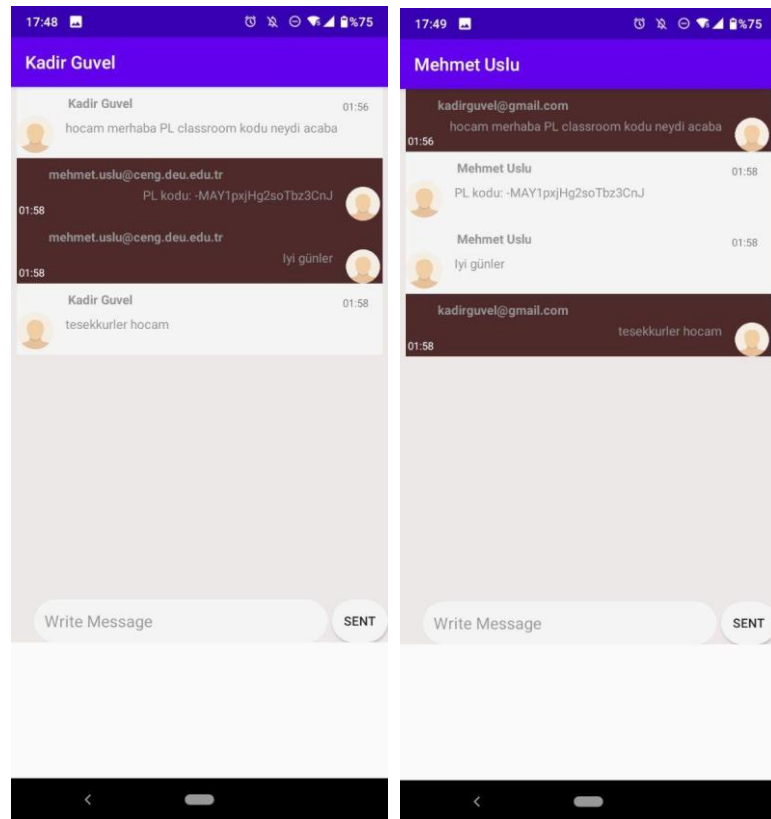


Figure 6.20 Chat Example II

6.11 Home Page of the Student

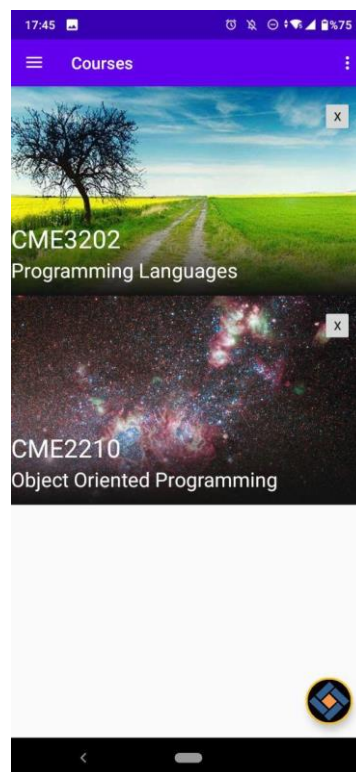


Figure 6.21 Home Page of the Student

7. Requirements

7.1 Inheritance

The user types “Student” and “Teacher” are extended from the abstract class “Person”. The variables and methods containing in the Person class are overridden in the other two classes as well.

```
public abstract class Person extends AppCompatActivity {
```

```
public class Student extends Person {
```

```
public class Teacher extends Person {
```

Figure 7.1 Inheritance Example

7.2 Abstract Data Types

ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations. It is called “abstract” because it gives an implementation-independent view. For providing the requirement of a project having an abstract data type list type is used in multiple parts of the project. In figures Figure 7.2, Figure 7.3, Figure 7.4, Figure 7.5, and Figure 7.6 usage of the list in various forms to deal with several different objects can be seen for an easy and an efficient coding experience.

```
List<Comment> listComment;
```

Figure 7.2 Example List Declaration From the Project

```
private Context mContext;  
private List<Comment> mData;  
comment adapter to show it off on post detail activity according to row_comment.xml  
  
public CommentAdapter(Context mContext, List<Comment> mData) {  
    this.mContext = mContext;  
    this.mData = mData;
```

Figure 7.3 List Usage From the Project

```

public class CourseAdapter extends RecyclerView.Adapter<CourseAdapter.MyViewHolder> {

    Context mContext;
    List<Course> mData ;
    public String courseID;
    FirebaseAuth fAuth;
    FirebaseFirestore fStore;
    DatabaseReference referenceStudent, referenceTeacher;
    FirebaseUser currentUser;
    String name;
    String te;
    private static final String TAG = "MainActivity";

    public CourseAdapter(Context mContext, List<Course> mData) {
        this.mContext = mContext;
        this.mData = mData;
    }
}

```

Figure 7.4 List Usage From the Project II

```

public class PostAdapter extends RecyclerView.Adapter<PostAdapter.MyViewHolder> {

    Context mContext;
    List<Post> mData;

    // this post adapter gets posts when called and projects it to fragment post fragment
    public PostAdapter(Context mContext, List<Post> mData) {
        this.mContext = mContext;
        this.mData = mData;
    }
}

```

Figure 7.5 List Usage From the Project III

```

public class ChatAdapter extends RecyclerView.Adapter<ChatAdapter.MyViewHolder>{

    // showing all people on chat list
    Context mContext;
    List<Chat> mData ;
    public String courseID;

    RecyclerView chatRecyclerView ;
    MessageAdapter messageAdapter ;
    FirebaseDatabase firebaseDatabase;
    DatabaseReference databaseReference ;
    FirebaseAuth mAuth;
    FirebaseUser currentUser ;

    public ChatAdapter(Context mContext, List<Chat> mData) {
        this.mContext = mContext;
        this.mData = mData;
    }
}

```

Figure 7.6 List Usage From the Project IV

```

// corresponding xml schemas which are row_message_item_(receive or send).xml
commentRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        listMessage = new ArrayList<>();
        for (DataSnapshot snap : dataSnapshot.getChildren()) {
            Message message = snap.getValue(Message.class);

            String userNow = firebaseUser.getUid();
            String userClicked = peopleKey;

            // getting every user but current user to be listed
            if (!userNow.equals(userClicked)) {
                if (userNow.equals(message.getSender())) {
                    if (userClicked.equals(message.getReceiver())) {
                        listMessage.add(message);
                    }
                } else {
                    if (userNow.equals(message.getReceiver())) {
                        if (userClicked.equals(message.getSender())) {
                            listMessage.add(message);
                        }
                    }
                }
            }
        }
    }
});

```

Figure 7.7 List Usage From the Project V

7.3 Switch Case

In the requirements there exists a statement that there must be two types of users as teacher and student. This requirement mostly divides all functions to be divided into two. One for student actions and the other for teacher actions. In the figures Figure 7.7 and Figure 7.8 two examples of usage of the switch case condition statements can be seen.

In Figure 7.7 the cases are user being student or teacher while in Figure 7.8 cases are the who belongs to the messages from the left or right side of the screen. The left side messages belong to the person the current user is talking to, while the right side of the chat belongs to the current user.

```

public MessageAdapter.MessageViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(parent.getContext());
    View viewChatMine = null;
    // deciding messages are sent or received and calling respective layouts to put on recycler view
    switch (viewType) {
        case VIEW_TYPE_ME:
            viewChatMine = inflater.inflate(R.layout.row_message_item_sent, parent, attachToRoot: false);
            break;
        case VIEW_TYPE_OTHER:
            viewChatMine = inflater.inflate(R.layout.row_message_item_receive, parent, attachToRoot: false);
            break;
    }

    // View row = LayoutInflater.from(mContext).inflate(R.layout.row_message_item_receive, parent, false);

    return new MessageViewHolder(viewChatMine);
}

```

Figure 7.8 Switch Case Example For Chat Part of the Project

```

if (!TextUtils.isEmpty(email) && !TextUtils.isEmpty(password)) //buralar eklencek
{
    FirebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener(task) → {
        // if clicked create student or teacher on chosen kind
        switch (studentOrTeacher.toLowerCase()) {
            case "student":
                String emailStudent = FirebaseAuth.getCurrentUser().getUid();
                DatabaseReference currentStudent = referenceStudent.child(emailStudent);

                currentStudent.child("UserKey").setValue(emailStudent);
                currentStudent.child("Name").setValue(name);
                currentStudent.child("Surname").setValue(surname);
                currentStudent.child("Email").setValue(email);
                currentStudent.child("Password").setValue(password);
                currentStudent.child("UserType").setValue(studentOrTeacher);
                startActivity(new Intent(getApplicationContext(), Login.class));
                Toast.makeText(context, Register.this, text: "Student Inserted To The Database.", Toast.LENGTH_LONG).show();
                updateUI();
                break;

            case "teacher":

                String emailTeacher = FirebaseAuth.getCurrentUser().getUid();
                DatabaseReference currentTeacher = referenceTeacher.child(emailTeacher);

                currentTeacher.child("UserKey").setValue(emailTeacher);
                currentTeacher.child("Name").setValue(name);
                currentTeacher.child("Surname").setValue(surname);
                currentTeacher.child("Email").setValue(email);
                currentTeacher.child("Password").setValue(password);
                currentTeacher.child("UserType").setValue(studentOrTeacher);
                startActivity(new Intent(getApplicationContext(), Login.class));
                Toast.makeText(context, Register.this, text: "Teacher Inserted To The Database.", Toast.LENGTH_LONG).show();
                updateUI();
                break;
        }
    }
}

```

Figure 7.8 Switch Case Example For Register Part of the Project

7.4 Foreach Loop

For each loop is used when we need data reference from firebase and we have at least two and maximum of five data reference across all adapters, fragments and activities.

```
DatabaseReference commentRef = firebaseDatabase.getReference(COMMENT_KEY).child(PostKey);
commentRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        listComment = new ArrayList<>();
        for (DataSnapshot snap:dataSnapshot.getChildren()) {

            Comment comment = snap.getValue(Comment.class);
            listComment.add(comment) ;

        }

        commentAdapter = new CommentAdapter(getApplicationContext(),listComment);
        RvComment.setAdapter(commentAdapter);
    }
}
```

Figure 7.9 Foreach Loop

7.5 Named Constants

Named constants that are used for permanent variables are oftenly used in the project. In figures Figure7.10, Figure 7.11, and Figure 7.12 some of those usages can be seen.

```
private static final String TAG = "StudentDrawer";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    mAuth = FirebaseAuth.getInstance();
    currentUser = mAuth.getCurrentUser();

    final String UID = currentUser.getUid();
}
```

Figure 7.10 Named Constants Example I

```
private static final String TAG = "MainActivity";
```

Figure 7.11 Named Constants Example II

```
Register.setOnClickListener((v) -> {
    listen register button and make sure its all filled

    final String email = Email.getText().toString().trim();
    final String password = Password.getText().toString().trim();
    final String name = Name.getText().toString();
    final String surname = Surname.getText().toString();
    final String studentOrTeacher = StudentOrTeacher.getText().toString();

    if (TextUtils.isEmpty(email)) {
        Email.setError("Please enter an email.");
        return;
    }
    if (TextUtils.isEmpty(password)) {
        Password.setError("Please enter a password..");
        return;
    }
    if (password.length() < 8) {
        Password.setError("Incorrect Form Of A Password.");
        return;
    }

    if (TextUtils.isEmpty(studentOrTeacher))// tüm attributelar için buralar eklenebilir.
    {
        Email.setError("Please fill in the blanks.");
        return;
    }
    if (!TextUtils.isEmpty(email) && !TextUtils.isEmpty(password))//buralar eklencek
    {
        FirebaseAuth.createUserWithEmailAndPassword(email, password).addOnCompleteListener((task) -> {
```

Figure 7.12 Named Constants Example III

7.6 Method Overloading

On multiple database references we had to overload the method with another on database change references to listen couple values on different parts of real time database. When we pull reference of need we create two types of one class that is creation argument lists are different.

```
databaseReference2.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

        for (DataSnapshot postsnap : dataSnapshot.getChildren()) {

            Chat chat = postsnap.getValue(Chat.class);
            if (!chat.getUserKey().equals(FirebaseAuth.getInstance().getCurrentUser().getUid())) {

                chatList.add(chat);
                System.out.println(currentUser.getUid() + " OFFER" + chat.getUserKey());

            }

        }

        chatAdapter = new ChatAdapter(getActivity(), chatList);
        chatRecyclerView.setAdapter(chatAdapter);
    }
}
```

Figure 7.13 Overloaded Method

7.7 Associative Arrays

For associative array example we need to pass arguments to next activity putExtra function creates another part that connects and maps variables to corresponding variable and we can access those extras on other activity with their data needed from this activity other activity.

```
public MyViewHolder(View itemView) {
    super(itemView);

    tvTitle = itemView.findViewById(R.id.row_post_title);
    tvDesc = itemView.findViewById(R.id.row_post_description);
    imgPost = itemView.findViewById(R.id.row_post_img);
    imgPostProfile = itemView.findViewById(R.id.post_detail_user_img);

    itemView.setOnClickListener((view) -> {
        Intent postDetailActivity = new Intent(mContext, PostDetailActivity.class);
        int position = getAdapterPosition();

        postDetailActivity.putExtra( name: "title", mData.get(position).getTitle());
        postDetailActivity.putExtra( name: "postImage", mData.get(position).getPicture());
        postDetailActivity.putExtra( name: "description", mData.get(position).getDescription());
        postDetailActivity.putExtra( name: "postKey", mData.get(position).getPostKey());
        // will fix this later i forgot to add user name to post object
        postDetailActivity.putExtra( name: "userId", mData.get(position).getUserId());
        long timestamp = (long) mData.get(position).getTimeStamp();
        postDetailActivity.putExtra( name: "postDate", timestamp);
        mContext.startActivity(postDetailActivity);

        if any post clicked we call post detail activity to show of post details and comments of it

    });
}
```

Figure 7.14 Associative Array


```

public MyViewHolder(View itemView) {
    super(itemView);

    tvTitle = itemView.findViewById(R.id.row_chat_person);

    itemView.setOnClickListener((view) -> {
        Intent chatDetailActivity = new Intent(mContext, ChatDetailActivity.class);
        int position = getAdapterPosition();

        chatDetailActivity.putExtra( name: "sent", mData.get(position).getUserKey());

        chatDetailActivity.putExtra( name: "namesur", value: mData.get(position).getName() + " " + mData.get(position).getSurname());
        mContext.startActivity(chatDetailActivity);

    });
}

```

Figure 7.14 Associative Array II

8. Project Parts for Each

Although everyone concentrated different parts of the project, the project is completed together with helping or continuing the job the other one is doing when someone cannot complete or get the problem or task. Concentrated parts for each are divided like this:

Ezgi : Login, Register and Messaging

Mehmet : Announcements, Assignments and Courses