Bachelor’s Final Project

Subject: Desktop application for managing projects and tasks

Professor: Dr. Hooman Tahayori

Mina Nayernia

About the project

The main idea of this project is to manage tasks and projects individually or in a team.
In this documentation, I will explain each feature of this application in code and also how it works.
Also because it’s a local application I will explain how to run each part of it on your own device.

1. Frameworks

I used the flutter framework for my front end because I was aiming to expand my project to mobile applications and the web. In flutter, I don't have to write multiple code bases for multiple platforms and it reduces the effort needed to produce versions of an application. The other reason was the high number of packages available in this framework.
And I used Django rest framework for my back-end and database because I could define my database models in python which makes the work much easier and I didn’t have to struggle with raw SQL anymore due to simple queries in this framework.

2. Database design

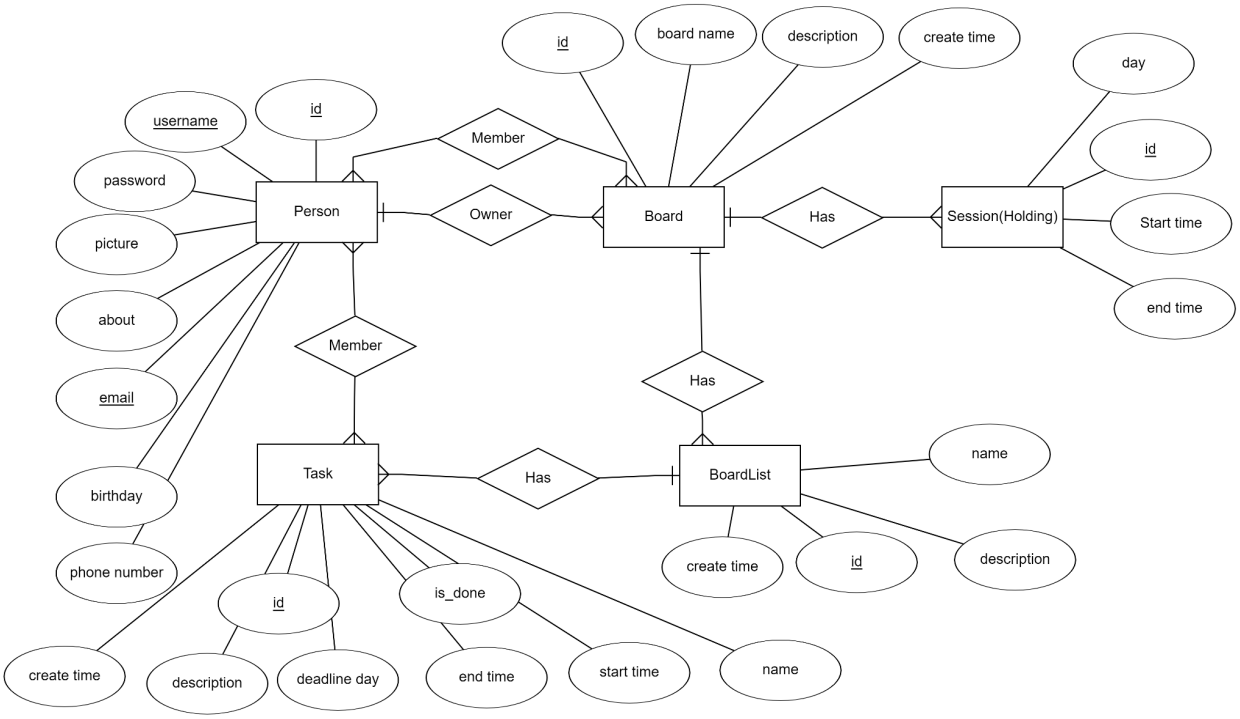


Diagram 1.

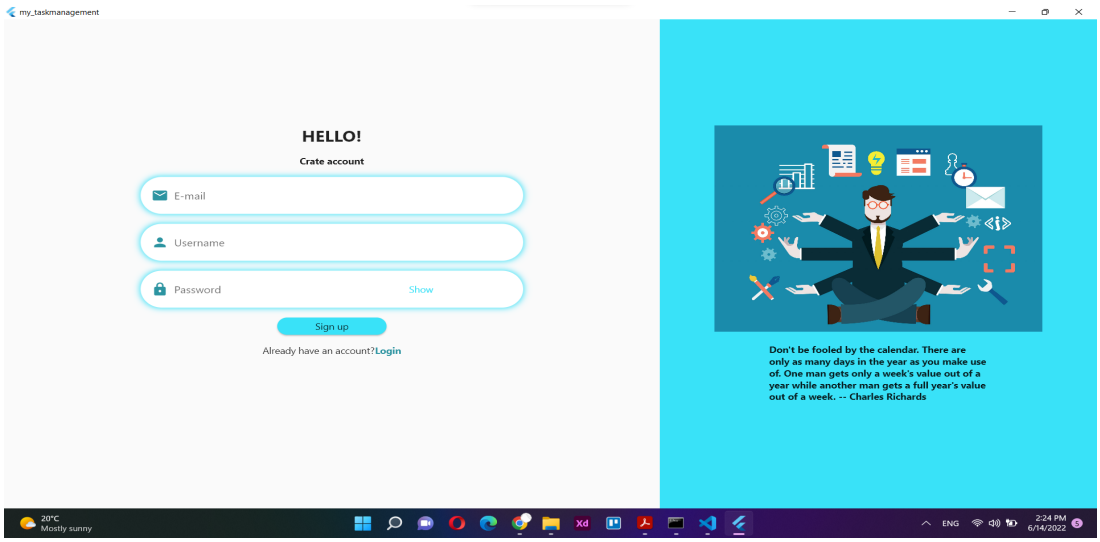
In the above UML diagram, you can see the main design of this project. It consists of 5 different entities: Person, Board, Session, Board List, and Task and their relations and attributes. You can see how I implement these entities in the file “models.py. the following is an example:

```
class Task(models.Model):
    boardList   = models.ForeignKey(BoardList , on_delete=models.CASCADE , null=True , blank=True)
    name        = models.CharField(max_length=30)
    description = models.CharField(max_length=256, null=True , blank=True)
    members     = models.ManyToManyField(Person ,null=True , blank=True)
    deadlineDay = models.DateField(null=True , blank=True)
```

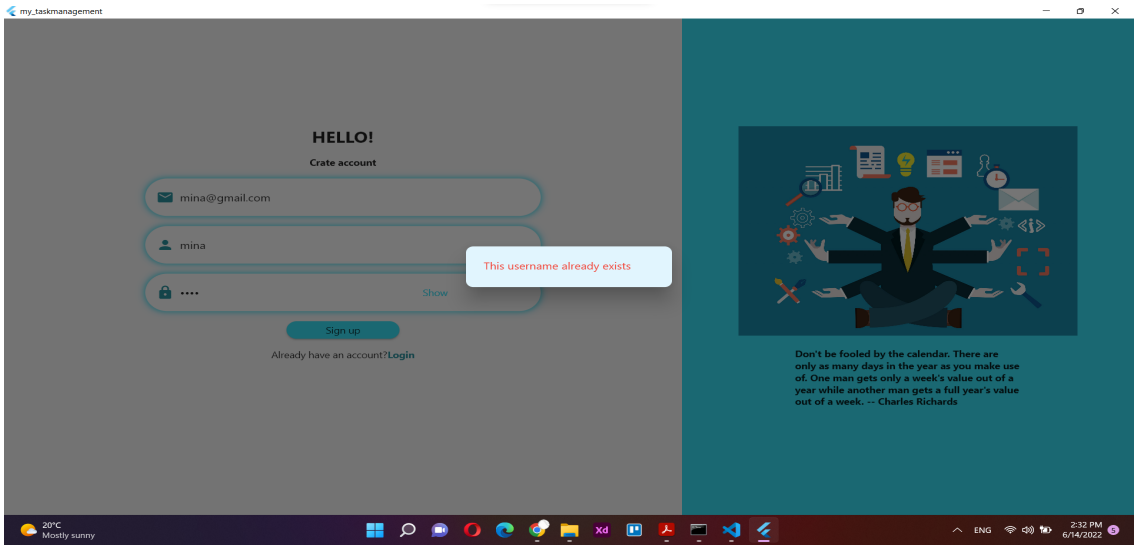
```
    startTime    = models.TimeField(null=True , blank=True)
    endTime      = models.TimeField(null=True , blank=True)
    create_time  = models.DateTimeField(auto_now_add=True)
    is_done      = models.BooleanField(null=True , blank=True)
    def __str__(self) -> str:
        return str(self.name)
```

3. Implementation

3.1 Signup



Picture 1.



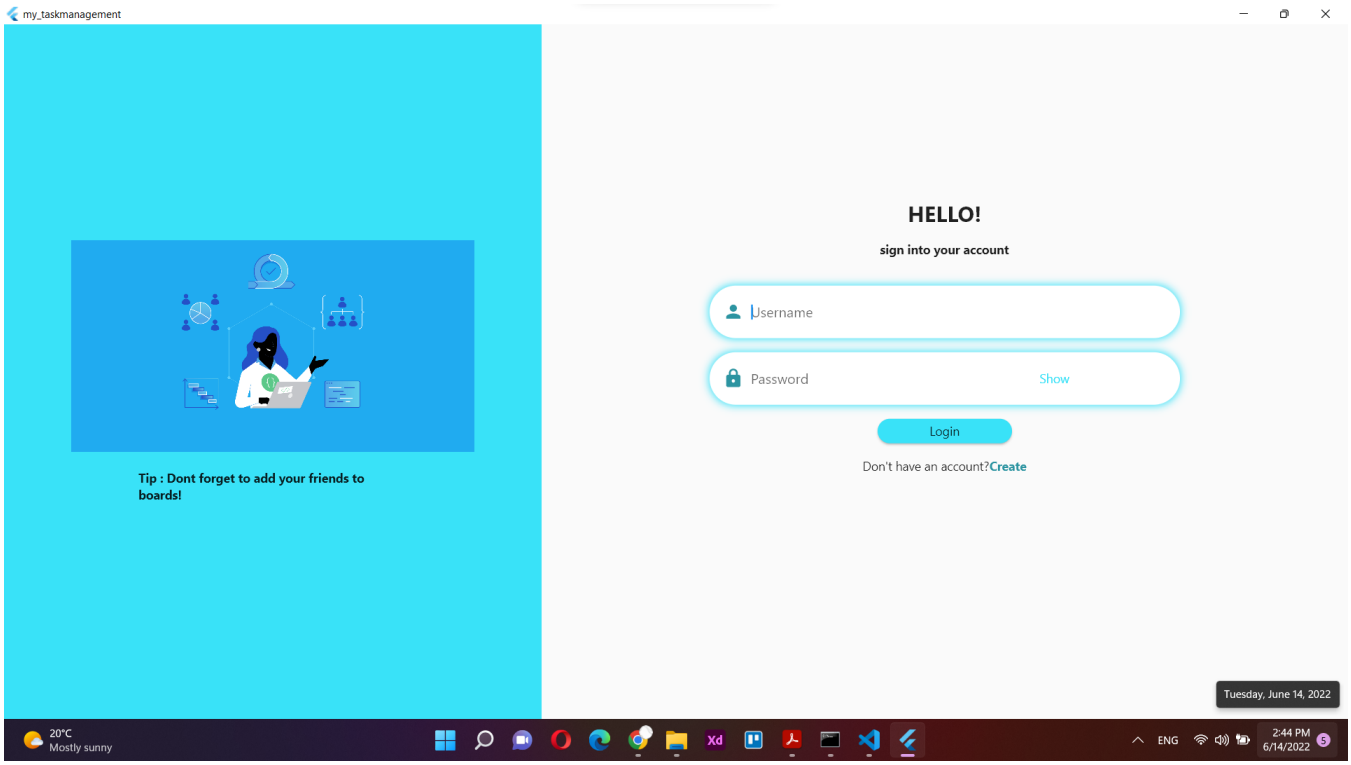
Picture 2.

```python
class CreateAccount(CreateAPIView):
    serializer_class = serializers.PersonSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        username = request.data.get('username')
        email = request.data.get('email')
        password = request.data.get('password')

        allusernames = models.Person.objects.values_list('username' , flat=True)
        allemails = models.Person.objects.values_list('email' , flat=True)

        if (username in allusernames):
            return Response({"msg" : "This username already exists" , "status": 500})
        if (email in allemails):
            return Response({"msg" : "This email already exists" , "status": 500})
        if (username is None or email is None or password is None):
            return Response({"msg" : "Fill all the filds" , "status": 500})
        else:
            newPerson = models.Person(username=username , password = password,
                                      email = email )
            newPerson.save()
            print("newperson added to database")
            return Response({"msg" : "newperson added to database" , "pk" : newPerson.pk , "status": 200})
```

Picture 3.

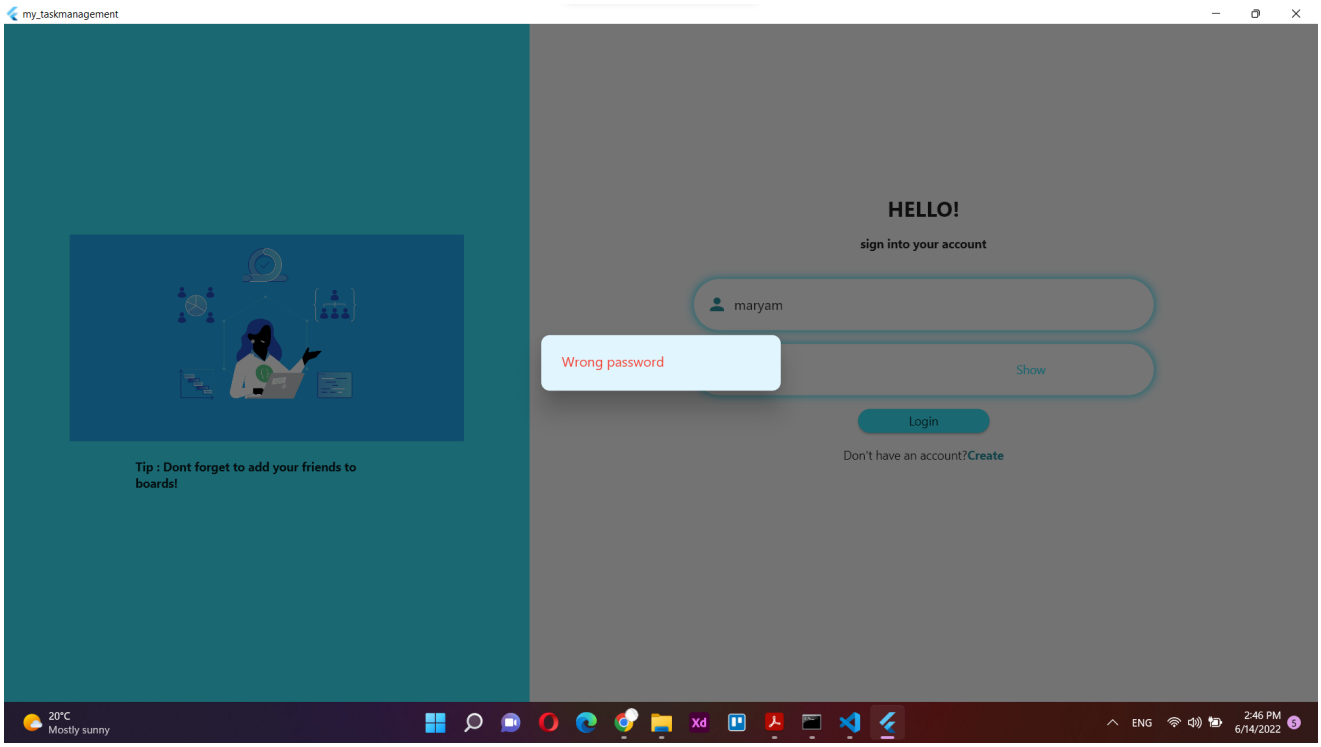
First, we have to sign up for the application. You can see the Page above (picture 1). You have to enter the Email, username, and password. the username and email should be unique and if there were any other users with that username or email it does not make a new account for you(picture 2).
As you observe in picture 3, I wrote a view for getting data(username, password, email) from the user in the front end, checking if the username or email exists in the database or if each value is null, and sending an appropriate message to the front and then showing it to the user.

3.2 Login



Picture 4.

If you had made an account before, now you have to just log in to the application by entering your username and password. If you enter the wrong username or password it does not allow you to enter the application and shows you the related error message (picture 5).
As you observe in picture 6, I wrote a view for getting data from the user in front and checking the username and password whether they have any problem or not. In case they have, show the related error message to the user else it would enter the application successfully.

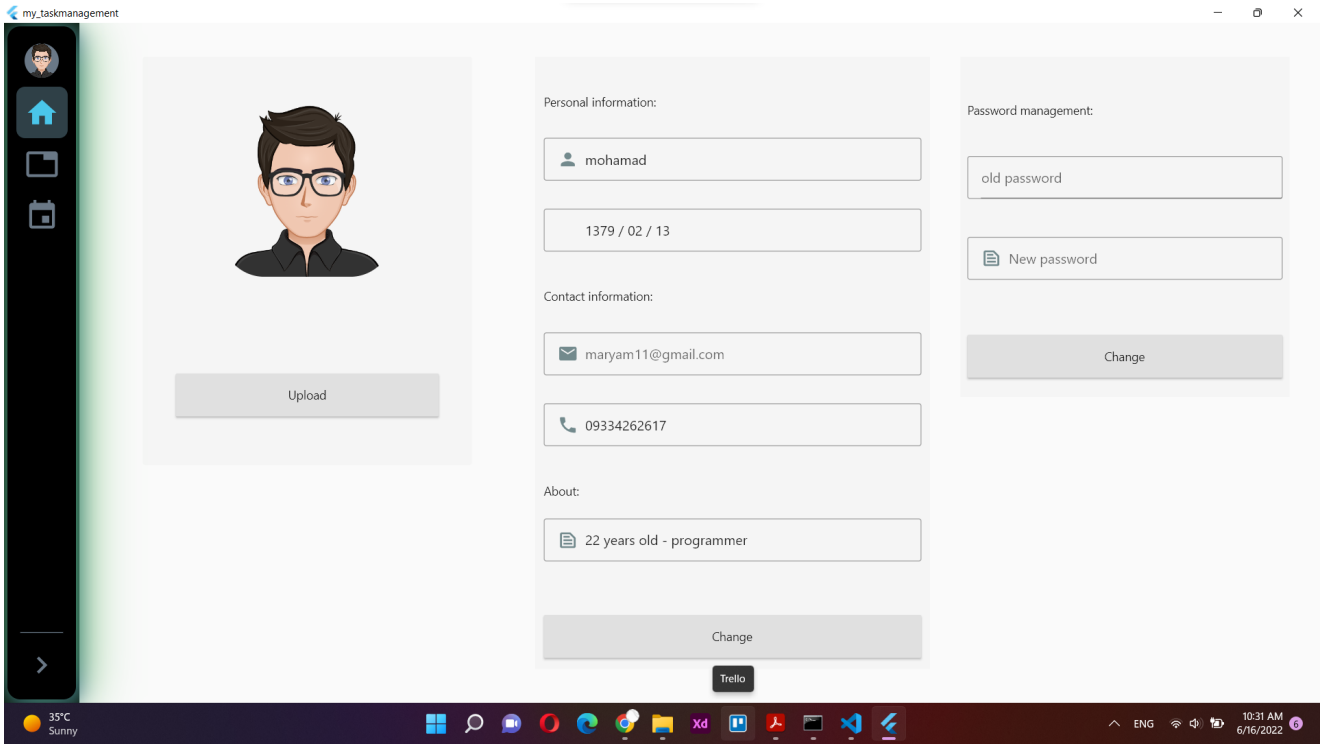


Picture 5.

```python
class Login(CreateAPIView):
    serializer_class = serializers.PersonSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        username = request.data.get('username')
        password = request.data.get('password')
        allusernames = models.Person.objects.values_list('username' , flat=True)
        if (username is None or password is None):
            return Response({"msg" : "fill all the fields" ,"status": 200})
        elif username in allusernames :
            person = models.Person.objects.get(username = username)
            if person.password == password :
                return Response({"msg" : "successful","pk" : person.pk ,"status": 200})
            else :
                return Response({"msg" : "Wrong password" ,"status": 200})
        else :
            return Response({"msg" : "wrong username" ,"status": 200})
```

Picture 6.

3.3 Profile



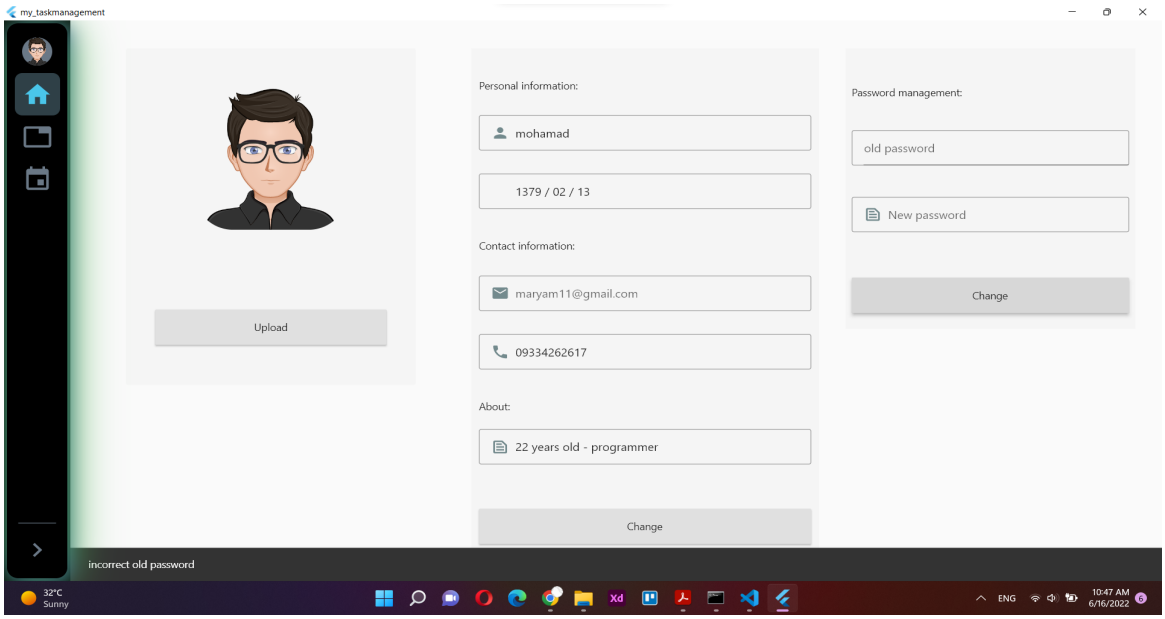
Picture 7.

As you enter the application, you can see the collapsible sidebar which contains 4 different tabs. (this is a ready flutter package that I used.)

```
import'package:collapsible_sidebar/collapsible_sidebar.dart';
```

In the first tab, you can access your personal information, and photo, and also you can change this data. The part “Personal information” consists of 5 Texfields with HintLabels which come from the previous value in the database or a default value if it is empty. For changing them you just have to click and enter whatever you want and then click the button “Change”.

To change your password, you have to enter your current password and the new one, then click the button “Change”. It would check your current password and then if it was correct then it will change it. (picture 8. The “Incorrect old password” error is shown at the bottom of the page)



Picture 8.

```python
class ProfileData(CreateAPIView):
    serializer_class = serializers.PersonSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        personID = request.data.get("personID")
        print("person id is :")
        print(personID)
        person = models.Person.objects.get(pk = personID)
        print("person is :")
        print(person)
        username = person.username
        birthday = person.birthday
        email = person.email
        phoneNumber = person.phoneNumber
        about = person.about
        pic = person.pic
        password = person.password
        return Response({
            "msg" : "success" ,
            "username" : username ,
            "password" : password ,
            "birthday" : birthday ,
            "phoneNumber" : phoneNumber ,
            "about" : about ,
            "pic" : pic ,
            "email" : email
            })
```

Picture 9.

(The view for showing personal information on the profile tab.)

```python
class ChangeProfile(CreateAPIView):
    serializer_class = serializers.PersonSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        username = request.data.get("username")
        email = request.data.get("email")
        birthday = request.data.get("birthday")
        phoneNumber = request.data.get("phoneNumber")
        about = request.data.get("about")
        pk = request.data.get("pk")
        print("about")
        print(about)
        person = models.Person.objects.get(pk = pk)
        person.email = email
        person.username = username
        person.birthday = birthday
        person.phoneNumber = phoneNumber
        person.about = about
        person.save()
        return Response({
            "msg" : "success" ,
            "username" : person.username ,
            "birthday" : person.birthday ,
            "phoneNumber" : person.phoneNumber ,
            "about" : person.about ,
            "email" : person.email
            })
```

Picture 10.

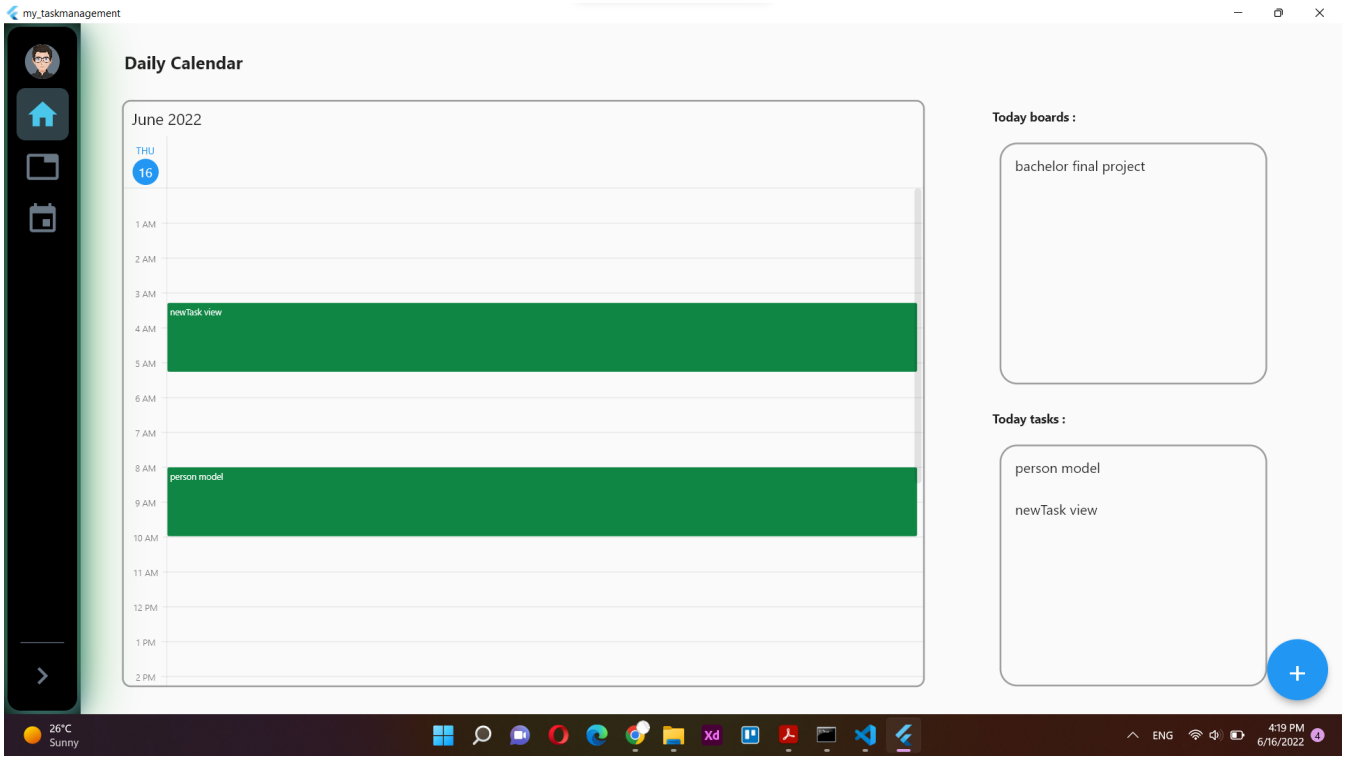
(The view for changing personal information on the profile tab.)

```python
class changePassword(CreateAPIView):
    serializer_class = serializers.PersonSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        personPk = request.data.get("pk")
        oldPass = request.data.get("oldPassword")
        newPass = request.data.get("newPassword")
        person = models.Person.objects.get(pk = personPk)
        if person.password == oldPass :
            person.password = newPass
            person.save()
            return Response({"msg" : "password changed succesfully!"})
        else:
            return Response({"msg" :"incorrect old password"})
```

Picture 11.

(The view for changing passwords on the profile tab.)

3.4 Daily Calendar



Picture 12.

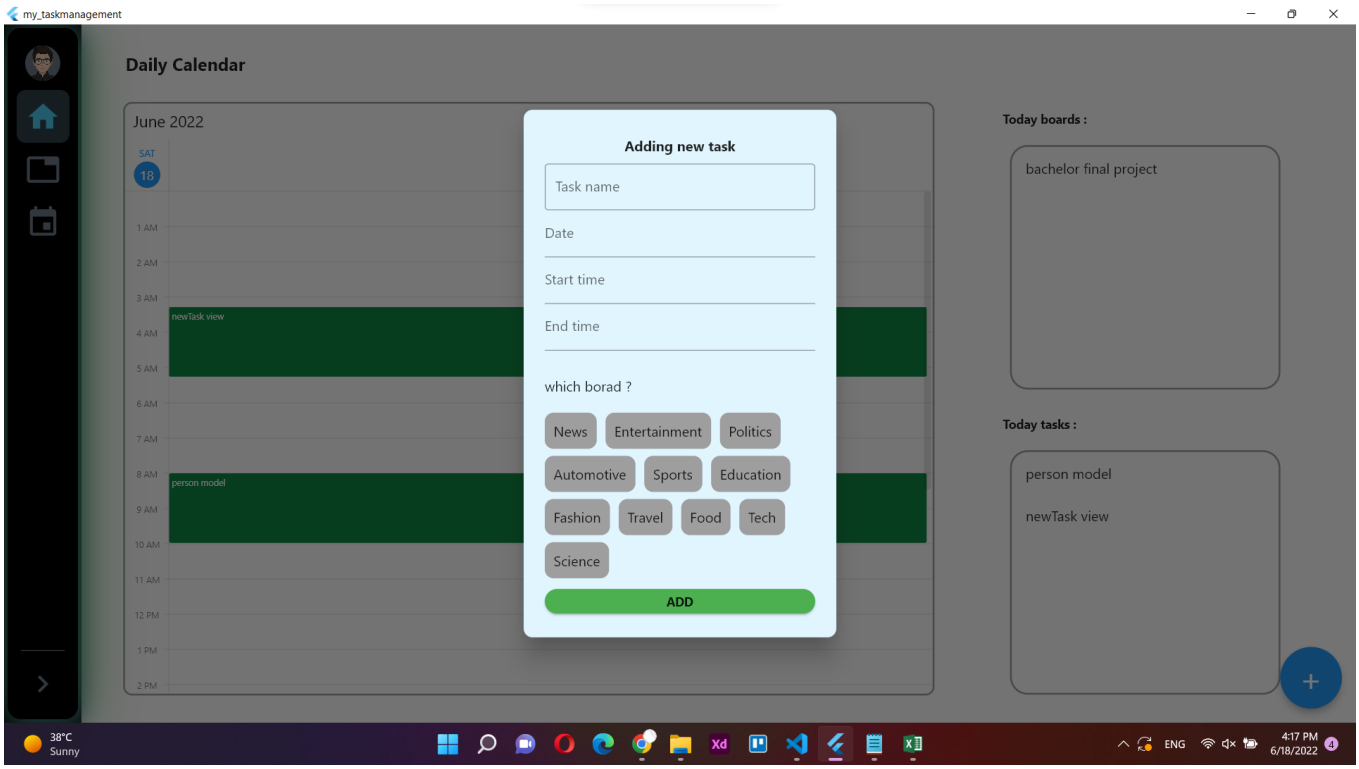
Here you can observe today's tasks and boards. For the API of this tab, it needs to send the Primary Key of the related person and get other data like each task's name, start and end times of tasks, and all of the boards that have tasks for today.

On the bottom right of the page, there is a blue button. By clicking on this button, a dialog box appears on the page and you can make a new task (picture 14) by entering the task name, date, start and end time. also if this task is related to a specific board, you can choose the board. (this is optional and you can create a new task without adding the board)

```python
class DailyCalendar(CreateAPIView):
    serializer_class = serializers.TaskSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        personID = request.data.get("pk")
        person = models.Person.objects.get(pk = personID)
        today = timezone.now().date()
        todayTasks = []

        TaskLists = models.Task.objects.filter(members= person, deadLineDay=today)
        if TaskLists.exists() :
            for item in TaskLists:
                temp = {}
                temp["name"] = item.name
                temp["startTime"] = item.startTime
                temp["endTime"] = item.endTime
                temp["is_done"] = item.is_done
                todayTasks.append(temp)
                if item.boardList != None :
                    temp["boardName"] = item.boardList.board.name
                else:
                    temp["boardName"] = None

        return Response(todayTasks)
```

Picture 13.

(the view related to getting data for the daily calendar.)

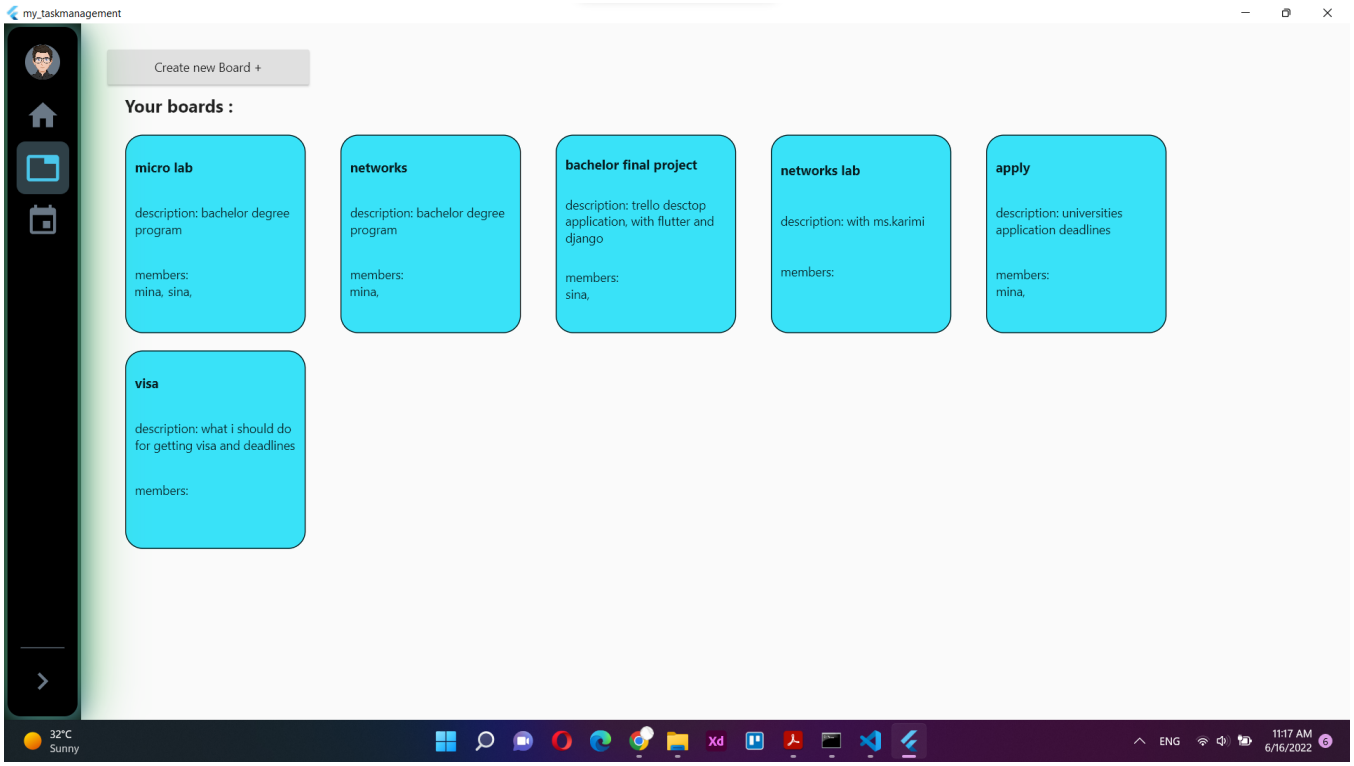


Picture 14.

```python
class AddTaskDailyCalendar(CreateAPIView):
    serializer_class = serializers.TaskSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        memberID = request.data.get("pk")
        startTime = request.data.get("startTime")
        endTime = request.data.get("endTime")
        deadlineDate = request.data.get("date")
        taskName = request.data.get('name')

        person = models.Person.objects.get(pk = memberID)
        newTask = models.Task(name = taskName , startTime = startTime ,
        endTime = endTime , deadlineDay = deadlineDate , create_time = datetime.now())
        newTask.save()
        newTask.members.add(person)
        newTask.save()
        return Response({
            "new task added"
            })
```

Picture 15.

3.5 Boards



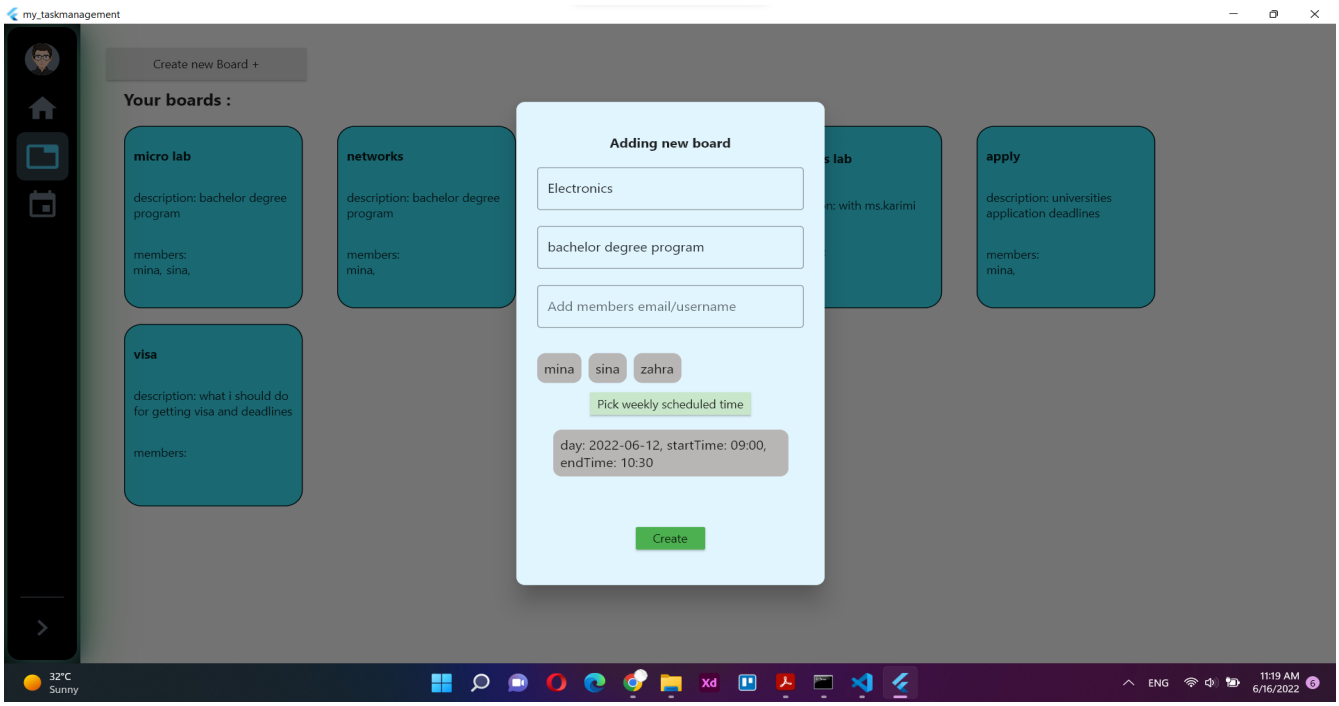
Picture 16.

In this tab, you are able to see your current boards and also create new ones. At the bottom of this tab, there is a grid view that shows each board and its description and members, in a container.

By clicking on each board (Light blue boxes), it leads you to the related board page (part 2.5.1 Single Board)

By clicking on the “Create new board”, a dialog will pop up (picture 17) and you are able to enter the board name and description, the member’s username or email, and some pre-scheduled tasks (or sessions) for your new board.

In pictures 18 & 19, in the “Create board view” I check the board name entered by the user, and whether there is any similar board name in the database or not. if not, it is going to create a new board and then add the members and session chosen by the user to that board.



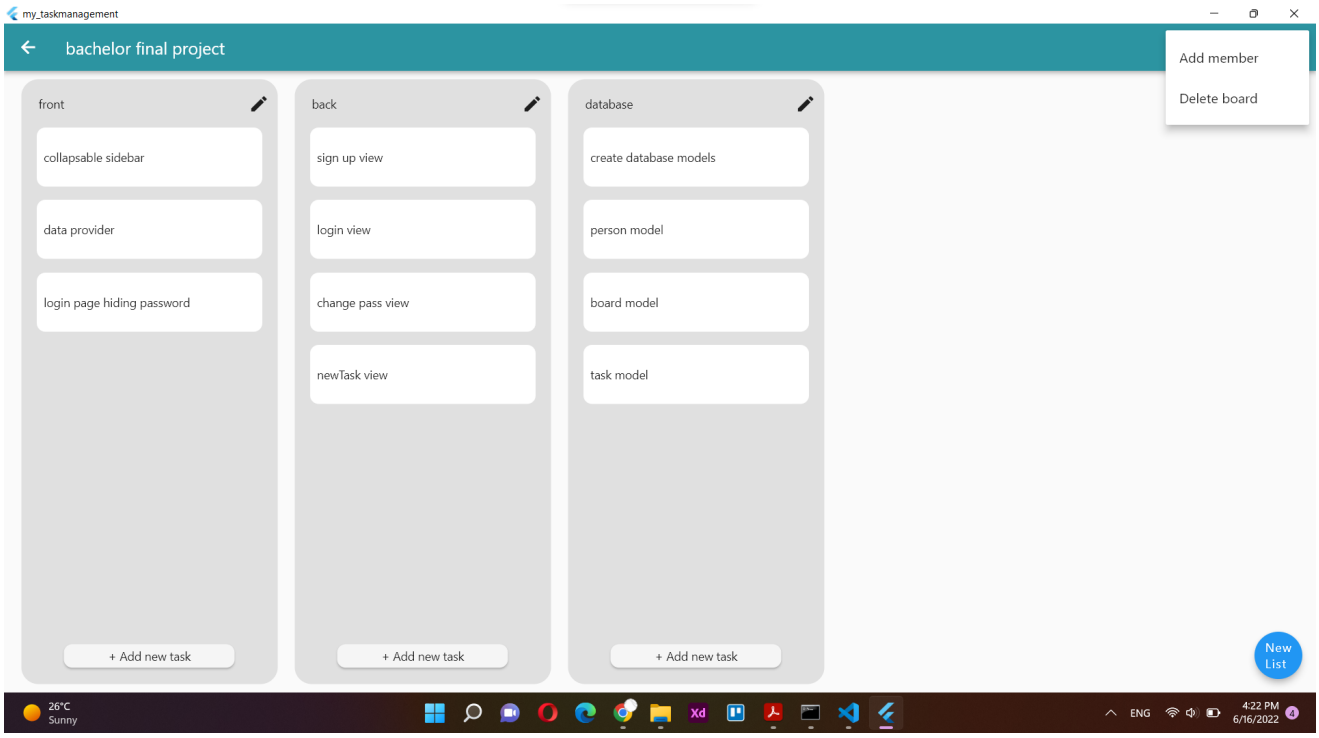
Picture 17.

```python
class CreateBoard(CreateAPIView):
    serializer_class = serializers.BoardSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):

        personID = request.data.get('pk')
        members = request.data.get('members') ##it is a list wich contains username or email or both
        boardName = request.data.get('boardName')
        sessions = request.data.get('holdings')

        description = request.data.get('description')

        owner = models.Person.objects.get(pk = personID)
        allBoards = models.Board.objects.filter(owner = owner).values_list('name' , flat= True)

        if boardName in allBoards:
            return Response({"msg" : "You already have a board with this name!" ,"status": 200})

        else: ##here we have to create the board
            allPersons = models.Person.objects.all()
            membersList = []
            allsessions = []

            newBoard = models.Board(name = boardName , owner = owner ,
            # create_time = datetime.now ,
            description = description)
            newBoard.save()
            print("newBoard added !")
```

```python
            for member in members:
                for person in allPersons :
                    if member == person.username:
                        membersList.append(person)
                        newBoard.members.add(person)
                    if member == person.email :
                        membersList.append(person)
                        newBoard.members.add(person)
                    else:
                        print("invalid member name or email")
                        print("persons")
                        print(person.username)
                        # return Response({"msg" : "invalid name or email!" , "status" : 200})
            if sessions is not None:
                print("is not none")
                for eachSession in sessions :

                    newSession = models.Holding(board = newBoard , day = eachSession[5:15] ,
                     startTime = eachSession[28:33] , endTime = eachSession[44:49])
                    newSession.save()
            else:
                print("is none")


            return Response({"msg" : "board and sessions added!" , "status" : 200})
```

Pictures 18&19.

3.5.1 Single Board



Picture 20.

On the single board page, the details of each board such as lists and tasks are shown. On the top right of picture 20, there is a menu that gives you 2 options, adding a new member to this board and deleting the current board.

In the center of picture 20, there is a ListviewBuilder consisting of all lists related to this board and in each list, there is a ListviewBuilder consisting of all tasks related to that list.

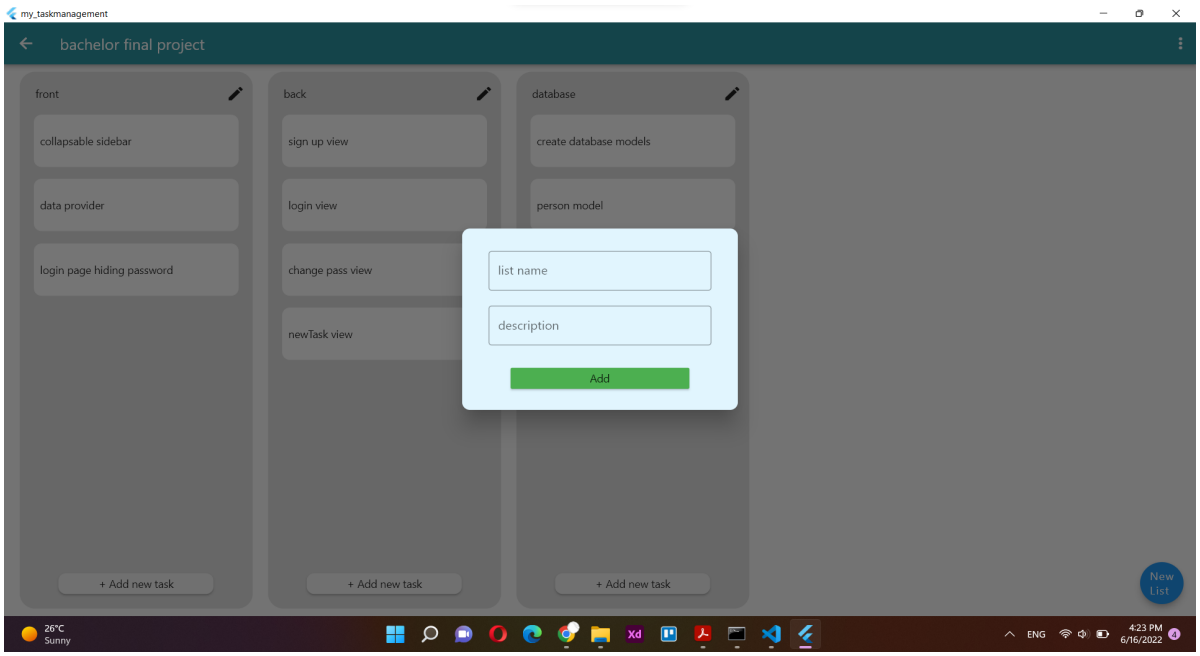
At the bottom of each list, there is a RaisedButton with the text “+ Add new task”. It allows you to create a new task by entering the task details (picture 22). As you see in picture 22, in part “members”, it shows all the board members and you can assign that task to members by choosing

them (after you click on a member the color of that member will change to green).
Also in picture 22, there are 3 fields, “date”, “start time” and “end time”. I used a ready flutter package, called Date_time_picker, to get these data in a good way from the user (in pictures 23 & 24 you can see what they look like).

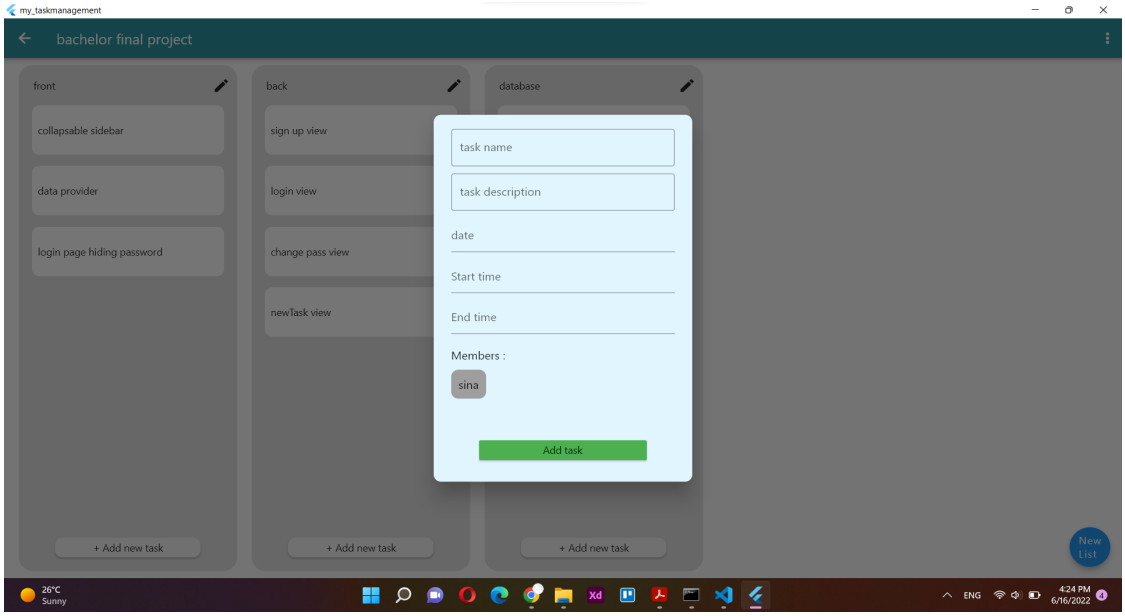
```
import 'package:date_time_picker/date_time_picker.dart';
```

There is a blue button on the right bottom of picture 20. By clicking on this button you are able to create a new list (picture 21).

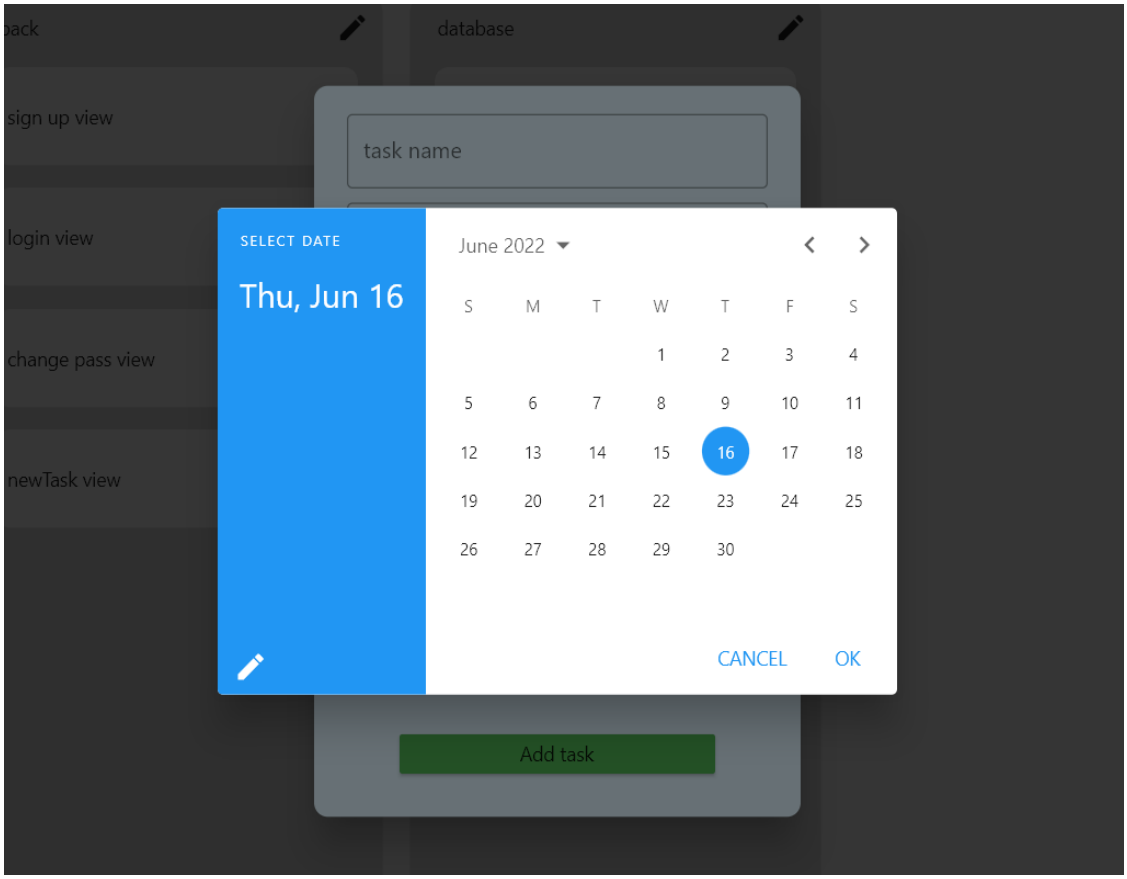
And finally, in pictures 25 & 26 you can observe how all these features' APIs are working.



Picture 21.



Picture 22.



Picture 23.

Picture 24.

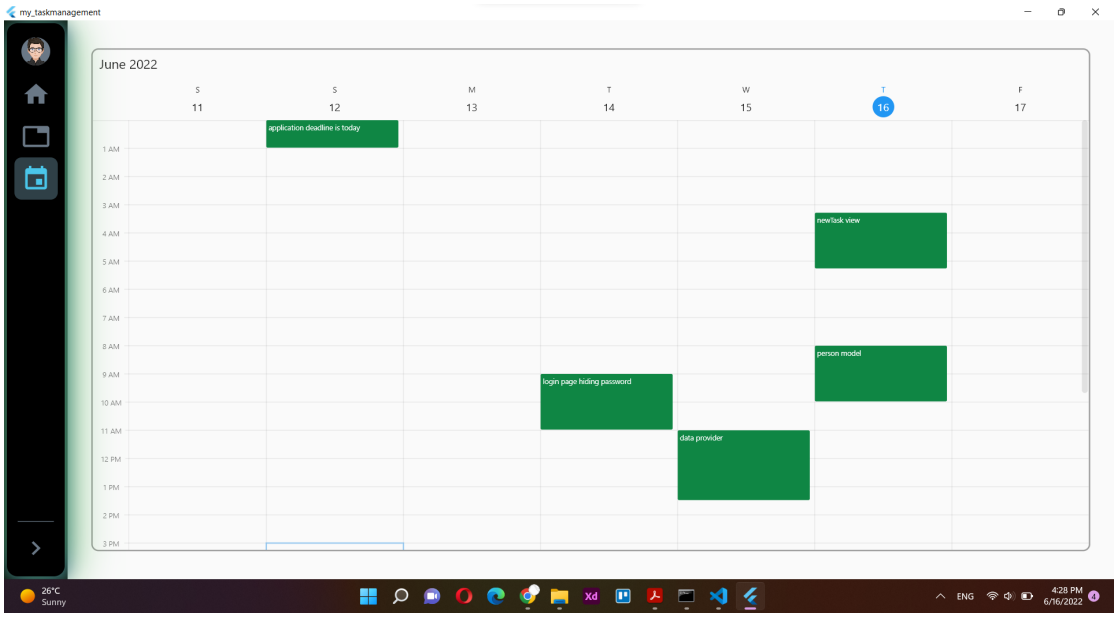
```python
class NewList(CreateAPIView):
    serializer_class = serializers.BoardListSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        boardPk = request.data.get("pk")
        listName = request.data.get("listName")
        listDescription = request.data.get("description")
        board = models.Board.objects.get(pk = boardPk)
        newList = models.BoardList(board = board , name = listName , description = listDescription)
        newList.save()

        return Response({"msg" : "new list addded!"})
```

Picture 25.

```python
class NewTask(CreateAPIView):
    serializer_class = serializers.TaskSerializer
    allowed_methods = ["POST"]
    def post(self, request, *args, **kwargs):
        personPk = request.data.get("personPk")
        person = models.Person.objects.get(pk = personPk)
        listPk = request.data.get("pk")
        boardList = models.BoardList.objects.get(pk = listPk)
        taskName = request.data.get("taskName")
        description = request.data.get("description")
        date = request.data.get("date")
        startTime = request.data.get("startTime")
        endTime = request.data.get("endTime")
        membersIds = request.data.get("members")
        members = []

        newTask = models.Task(boardList = boardList ,name = taskName ,
         description = description , startTime = startTime , endTime = endTime , deadlineDay = date ,
        is_done = False)
        newTask.save()
        newTask.members.add(person)
        newTask.save()
        for id in membersIds:
            member = models.Person.objects.get(pk = id)
            newTask.members.add(member)
            newTask.save()
```

Picture 26.

3.6 weekly calendar



Picture 27.

On the last tab of this application, I decided to give the users a general overview of their tasks in the current week.
The only point here was finding the start and the end of the current week by date, which you can see how I did on pages 28 and 29.

```
    def post(self, request, *args, **kwargs):
        personPk = request.data.get("pk")
        person = models.Person.objects.get(pk = personPk)
        today = datetime.today()
        todayweekday = today.weekday()
        sat = today
        friday = today
        # find previous saturday
        while True:
            if sat.weekday() == 5 :
                break
            sat = sat - timedelta(days=1)
        # find next friday
        while True:
            if friday.weekday() == 5 :
                break
            friday = friday + timedelta(days=1)
        allTasks = []
```

Picture 28.

```
for i in range(0,6):
    taskList = models.Task.objects.filter(members = person , deadlineDay = sat + timedelta(days=i))
    for task in taskList:
        allTasks.append(task)
res = []
for item in allTasks:
    temp = {}
    temp["name"] = item.name
    temp["startTime"] = item.startTime
    temp["endTime"] = item.endTime
    temp["day"] = item.deadlineDay
    res.append(temp)
return Response(res)
```

Picture 29.

3.6 sending email as reminder

In order to send an email, we have to change settings.py as you can see in picture 30. (I made a new account "managementproject2223@gmail.com")
Also in picture 31, you can see how I create each email and the way I send it to members of each task.

```
# SMTP Mail service with decouple
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = "smtp.gmail.com"
EMAIL_HOST_USER = "managementproject2223@gmail.com"
EMAIL_HOST_PASSWORD = "13781378"
EMAIL_PORT = 587
EMAIL_USE_TLS = True
DEFAULT_FROM_EMAIL = EMAIL_HOST_USER
```

picture 30.

```
from django.core.mail import send_mail

class sendEmail(CreateAPIView):
    today = timezone.now().date()
    allTasks = models.Task.objects.all()
    for task in allTasks:
        if task.deadlineDay == today :
            members = task.members.all()
            for member in members :
                msg = "Hi " + member.name + "!     " +task.name + "has to be done today! dont forget :)"
                send_mail(
                    'task reminder', #subject
                    msg , #message
                    "managementproject2223@gmail.com" , #from
                    [member.email ], #to
                    fail_silently= False
                )
```

picture 31.

4. Run

a. Install Django rest framework: https://www.geeksforgeeks.org/django-rest-framework-installation/
b. Install flutter(you may need a VPN due to prohibitions in Iran): https://docs.flutter.dev/get-started/install

After installing all requirements we have to run back and front individually:

back) in folder “taskmanagement_back” write these commands (if you made a virtual env first you have to active the env by env\Scripts\activate)

- Python manage.py runserver

front) in folder “my_taskmanagement” write these commands

- flutter run

  Then choose 1. Windows application