07.06.2020

AKDENIZ UNIVERSITY

COMPUTER ENGINEERING

optime

SENIOR PROJECT

2nd SEMESTER REPORT

by

Fatma ERKAN

Supervised by

Murat AK

Table of Contents

1. ABSTRACT

Optime is an intelligence schedule generation application to plan works. It is an application for Android. In detail, the application will plan tasks and projects. It provides users to schedule their time and manage business projects. There are many features of the application. However, the most important property is the intelligent schedule generation. The program can be used both as personal and as teamwork. With this application, everyone can easily schedule their daily routines. Thus, users can save time and do their works with a plan. There are many goals of the project that is help users to optimize time with a minimum loss, remind users about task deadlines, offer simple and clean user interface, etc.

2. PROJECT DESCRIPTION AND FUNCTIONS

Optime will be a mobile application that is aimed to arrange the time thriftily so as to achieve success in one's own life. In project management part, application which is named "optime" supplies some opportunities to reach information that belonging to the project as documentation which includes reports, provide to create task and share it with other users or team members, assign task to anyone else, shuffle tasks depend on their deadline and priority. In daily life, people may forget their plans, works, meets, or some obligations to do once in a while. Optime is an assistant application to prevent some unexpected results that comprised from the giving something amiss.

There are different meanings of a painting according to beholder's perspective. Somebody looks at this painting from a distance, and they like it or not. It is the missing the small details that given the meaning to whole then only then. However, the others see it and get closer to it, Thus, they can investigate all part of the painting, analyze and make it more meaningful. Like that painting, our life has a meaning with small details. Optime helps you for catch the small point of life besides obligations. If we mention about the obligation, an old and high blood pressure patient person has to take his drug neatly, but he may slip his mind, so it can end up badly. By means of Optime, when the time of drug comes, user keep informed by notifications and alarms. On the other hand, these small details which make our life more meaningful we mentioned can be reading a book. User wants to read a book 2000 pages and continue to read regularly. Optime be helpful to organize it, reading is regulated according to the day by day or pages. This choice belongs to user on two type of using application. Optime gives an opportunity to categorize as both long-term and short-term plans like those.

Other feature of Optime that it can be chosen as personal or business part by user. Personal part reminds generally social activity, not so important plans, special date like birthdays, etc. Business part of it is about work. It can be a meet, a congress, a deadline, an important knowledge to add, etc. Of course, both of them can be arranged according to user's desire and Optime send user to notifications and reminds by alarm regularly.

The most effective thing is the color. Because of color, Optime gives user to a facility to remark plans according to the order of significance. For example, an old person's taking drug is more important and one has to do it first of all. Its color is red, the most important things are remarked red. On the other hand, reading an essay for presentation is important but not so much urgent. So,

it is remarked with color yellow. The other color is white. White is general. It can be a vocabulary's meaning which makes him wonder. User searches it whenever he wants. If person has a time pressure and he cannot determine which one is the most important, colors that chosen by the user helps him. At the same time, encolouring the plans that renders the application more attractive and more entertaining.

Optime reminds the habitual progress, which is not done for a long time, like as to make exercise. A fat and reluctant to lose weight person does exercise rarely after some time. Our application sends a notification after a while to him. Optime, which is arranged according to user request, reminds subjects, plans, habits, etc. in this manner via notification.

In addition to these, user can mark plans by done or not done. It makes easier the determining the time to end of user's daily duties discriminatingly and because of Optime, user can achieve success in his own life by following his goals step by step carefully.

The most important feature of Optime is that it makes the general productivity analysis of user and shows it as a table timely. And it causes to increase the application's general usage and its user group. It is sure that it increases the discipline and this qualification isolates the Optime from other application in this direction.

Optime belongs to user completely and be arranged according to user's desire. Personal or business part features help his abstaining from mess. For example, daily plans are not viewed in the time of business part of application. Also, user can arrange the time intervals every feature of Optime.
Also, it is able to make private a work schedule for users. For instance, Optime can be a guide for a student who prepares for an exam. This schedule arranges the time period of a working plan by determining break also which user requests. Working 25 minutes and giving a break 5 minutes, and after again working 25 minutes that can be an example for the work schedule on our application. Like the others, this is also reminded by notifications and alarm. In addition to these features, keywords can be searched on the seek bar, and so user rules over the all knowledge from past to present easily.

Optime is a mobile application that is so private and a quick that user can always achieve and use this technological agenda at anytime and anywhere easily in today's world. In brief, Optime gives an opportunity to its users not to missing the little details that makes one's life more easier, more efficient, more productive, so to have a more meaningful life, thus; user can achieve the success in his life step by step.

In briefly, task management software has become wildly popular nowadays, that means there are plenty of options to choose from. Whether to looking for a simple way to organize tasks or a powerful enterprise-grade system, there is a task management tool everywhere. However, most of them are just for project management. Optime is an online schedule management application for android and web based. In more detail users can create tasks with assigning due time. Application's main features will be planning tasks as well as planning a project. User will also identify priority of the task and the plan of the day will be changed depending on it. To detail a task user can add

images that increase the visual quality. They also be free to create a repeated task which means it can be located in a plan every day. Briefly, optime is an application to help organizing time optimally. On the other hand, users can organize their business as project management. Business professionals rely on a project management software to help them oversee multiple endeavors. Companies can conveniently mitigate risk by identifying failing aspects of a project using time tracking that estimate time of completion for each stage of project. This part is used to project planning, scheduling and management. It provides managers to control developers, to control project costs and budgeting, to check quality of development, documentation, resource planning, time tracking, and many more. It is also used for collaboration between team members. This management application provides users to schedule their personal time and manage business projects with two parts which is named "optime". There are many fundamental properties of the application, but the most important feature is the optimize time in the course of schedule.

2.1 FUNCTIONALITIES

2.1.1 Functional

In project management part, users should be able to

- reach information that belonging to the project as documentation which includes reports,
- provide to create project,
- share tasks with other users or team members,
- assign tasks to anyone else in the project team,
- shuffle tasks depending on deadlines and priorities,
- give a break between tasks according to the pleasure of user,
- take note to any task,
- give report of productivity analysis at the end of the week,
- change color of the task depending on priority.

In personal time management part, users should be able to

- divide big task to small tasks,
- define free times which is out of the working time for designing personal time,
- remind things that are not done for a long while,
- control repeating tasks,
- take note to any task,
- shuffle tasks depending on priorities,
- give a break between tasks according to the pleasure of user,
- change color of the task depending on priority,
- determine if task reduplicate.

2.1.2 Non-functional

Usability

- The system will be usable by students, businessmen, academicians, some companies, all ages and various groups of people. Because of that, it will be well organized in such a way that user errors are minimized.

- Any user can use the system after a minute of training.
- Optime will be have a user-friendly and simple user-interface for all of potential user types.

Reliability

- System can handle many numbers of usage at the same time.
- Optime will run without a failure. It needs to be stable.

Security

- The application needs to secure user information from threats.
- Private data stored in the application will be encrypted that it can not expose data in case of a cyber-attack.

Performance

- The system will have three main performance requirements which are response time, workload, and platform.
- Load time of the application should be low. Most of the response times will at most 0.2 seconds. Some features like upload a photo to a task may have longer response time like bigger than 5 seconds. In this situation, small feedback may be necessary. Response time can depend on users' internet connection status.
- Workload may vary to the system's server status which depend on the chosen system server and system's capability of supporting customers.
- The database should be optimized so that users can quickly and easily look for the information they can access.

Extensibility

- The application will be easy to maintain.
- It is available on multiple platform as mobile and web.
- To improve and modify the application, data examples will be extendable.

Portability

- Any device with android operating system will be able to run optime.
- It is also available on web. Users can reach it from any browser.
- The system can be updatable to add new feature changes.

Accessibility

- The application is designed for all kind of users who has android device or a computer. Optime has a small documentation tutorial about how to use it with visuals. Clients will be able to use application with any problem as complex information.
- Optime will be free to download by anyone in Google Play Store for mobile version.

3. ALGORITHM

The schedule algorithm of Optime will be created by taking into consideration of time in the personal scheduling part. Schedule algorithm is used to optimize user's time. Algorithm's steps are shown below:

- The first step is calculation of free times to schedule tasks. To calculate free times, it is important to know some information as waking time of user and sleeping time of users. Then if user have a job, the working time is required. Beginning of working time and end working time are necessary to find free times between waking time and beginning of the working time, between end of the working time and sleeping time. Thus, with all that the free time of the day will be calculated.
- User can have off days. In the circumstances, the days will be known to prepare schedule list. Whole day will be scheduled to optimize time for user. The free time of day will be selected between waking time and sleeping time.
- After finding leisure times, personal task of users will be listed.
- Tasks are separated by their status which definition is completed tasks and uncompleted tasks. Completed tasks cannot join the schedule algorithm. Because the aim of the schedule algorithm is that to preparation of plan using uncompleted tasks.
- Then, if due date of some of the uncompleted tasks which are close, priority of the tasks will be increased by the algorithm depending on nearest due dates.
- Next, it is controlled whether uncompleted tasks have a pomodoro idea which is the most popular working algorithm with 5 minutes breaktime after every 25 minutes working. Every task, which contains pomodoro, has some subtasks depending on task time. A subtask lasts 25 minutes. If a task's time is 5 hours, then it contains 12 subtasks. Correspond to tasks which contains pomodoro, subtasks will be scheduled. If task does not contain pomodoro, then time span calculated by task time. If task lasts 3 hours, 3 hours will be reserved for this task without any break.
- If a task is reminder like taking medicine, it will be added to schedule depend on reminder time. If taking medicine time is 10.00 AM, the notification will be shown at 10.00 AM. It is also important to know how many times it is shown in a week or month. Depending these properties, schedule must include also reminder tasks.
- Using available tasks and time of tasks, a list will be used to generate the algorithm.
- With tasks list and free times, algorithm will generate a schedule for daily, weekly and monthly.
- If new task added to the program, schedule will be generated again.
- User can mark tasks as done or delay. If task is marked done in the correct time, there is no change in the schedule list. On the other hand, if task is marked as done earlier, schedule algorithm calculates time and tasks depend on new part of free time. In addition to this, user can want to delay task for a later time. The algorithm calculates time and tasks depend on new part of free time and delayed task.

The project management algorithm is used to manage projects or events which are planned with a team. Steps of algorithm are explained below:

- The project is formed as defining its name and definition by a director. At the same time, the members of the project are identified by using the user list in the check user page with checkboxes.
- After the project is formed, tasks can be added to it and these tasks can be added to it and these tasks can be assigned to the members of the project. Thus, project members can be work simultaneously.
- Tasks can be added to boards. Also, these tasks can be added on the project without boards.
- Due date can be added to tasks. While making a to do list for tasks which has the earliest due date take place on the top. So, user give priority to his/her plans according to deadline.
- Tasks on the project schedule will not be used. Because schedule algorithm is used for organizing personal tasks.

4. REQUIREMENTS (C# and Visual Studio)

To build this application, there must be a platform for mobile application development. It is planned to use Microsoft's Xamarin technology which is in Visual Studio. Besides, C# programming language will be used to develop. C# is a general-purpose programming language that provides to develop applications for many platforms. It is an object-oriented programming language which is created by Microsoft. It was derived from C language family and it is close to C++ and Java. The reason to select of this language is that C# is used for Mobile applications, Web applications, Desktop applications, Web services at the same time.

Visual Studio 2019 will be used in this project. Microsoft published Visual Studio 2019 in April. Microsoft Visual Studio is an IDE. It is used to develop websites, web apps, web services, and mobile applications. Visual Studio also supports different programming languages and allows developers to debug easily and quickly. Visual Studio provides developers these programming languages: C, C#, C++, Visual Basic, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML, CSS, Python, Ruby, Node.js, M. Most developers choose Visual Studio to build an application, so there are many documentations in the internet. That is why Visual Studio is selected to build project.

Visual Studio also provides .Net Core Framework which is extensible and the latest version of the Microsoft's Framework Technology. It was designed to support cross platform development. There is Entity Framework Core part of .Net Core technology to build objects and its relations. Entity Framework Core technology is also open source and cross platform for data access technology which is also known as an object relational mapper (O/RM).

Xamarin is a platform that developed by Miguel de Icaza, Nat Friedman, Joseph Hill. It provides to build IOS, Android and Universal Windows Platform at the same time. It is possible to develop a native application. It is not necessary to learn Objective-C, Swift or Java.

Xamarin will be used with Entity Framework Core to avoid writing one code many times. Database part will be created by Entity Framework Core technology which is based on code first approach.

Microsoft SQL Server is a relational database management system that is developed by Microsoft. It can run on the same computer or on another computer across a network. From this approach, it provides developers to run in synchronism. In addition, SQL Server keep 4.1 billion records.

5. RESPONSIBILITIES OF THE PROJECT

- Doing a feasibility study to define possible problems and identify if the project is useful.
- Observing similar applications and using that information to design optime. (research and development)
- To define requirements and conditions depending on users' needs. (requirement analysis)
- Designing a flow diagram to show system requirements graphically.
- To define business process to manage the project professionally.
- To complete project development steps to begin implementation.
- To generate database tables in MSSQL Server.
- Defining relationships between tables.
- Development of Stored Procedure, View and Function in MSSQL Server.
- Identifying information architecture to make the project more understandable.
- Drawing wireframes to specify the basic steps of the project.
- Interaction design to create the project that enables user to achieve objectives. It tends to be an application.
- Identifying a mobile template to start generating the mobile application.
- To complete templates with images and contents.
- Process of architectural design and preparing documentation.
- Implementation of project layers in Visual Studio IDE.
- Development of services.
- Implementation of the web panel to control project performance, security and bugs.
- Development of user interfaces.
- Coding XAML to build mobile design.
- Implementation of Android application with Xamarin.
- Testing database tables of the project.
- Testing architecture design.
- Tests of performance and security.
- Mobile application tests.
- Wearer tests of the application.
- To control cross-platform interaction.
- Evaluating feedbacks and logs of the project.
- Reporting activities.

6. WIREFRAMES/MOCKUPS

Figure 6.1 Optime Icon

The icon of any application is one of the more important parts of the application for the distinctive character. The icon of the optime will be shown in Figure 6.1. It contains the name of the application with a type font called “Lightning Crashes”. The purpose of the logo is to associate with application and simple.

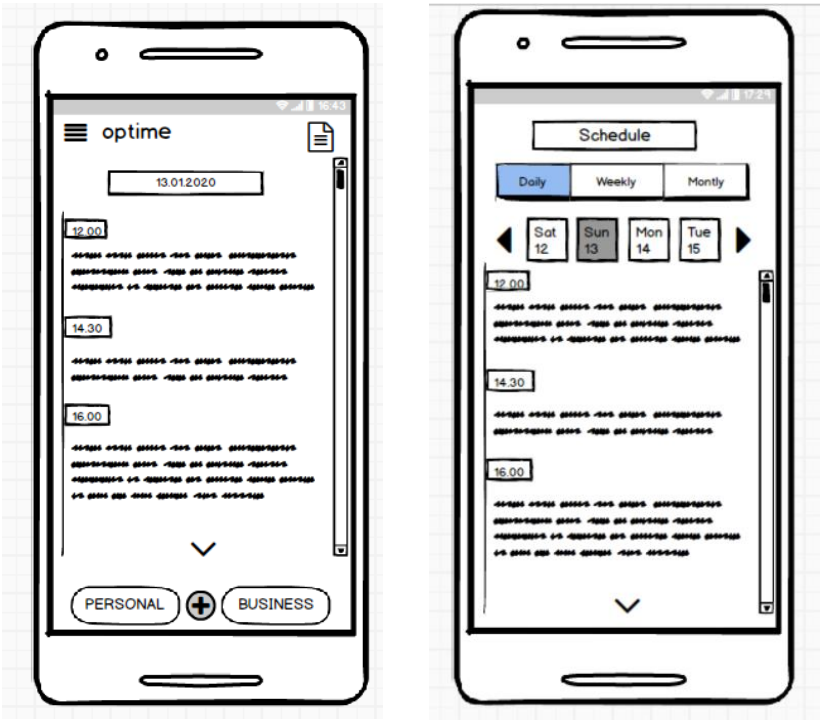


Figure 6.2 Optime Task Schedule Pages

Optime is an application that online schedule management for Android. In more detail, users can create tasks by defining due time. The main purpose of the application will be planning tasks daily, weekly, and monthly. This application provides users to organize their works and manage projects if any exists.

In Figure 6.2, task lists are shown in the left part. At the top of the screen icon of the application is located to refresh the page. At the right of the top an icon shown to see quick notes for users. The application contains some notes. The aim of those notes, if a user needs to note something about tasks or projects, it is possible to create a quick reminder. For instance, the user can join a meeting and a task assigned to the user about creating a project management board for the management of the business plan as an IT project. Users can note it at that moment and after the meeting, the user can control their notes to create a new project management board using the project screen. On the bottom of the screen, there are three buttons located. The first one is to show the personal tasks schedule daily. As can be seen, tasks are placed with their schedule times and task names on the screen. On the right part of the figure, the whole schedule is shown to the client. The client can select the daily, weekly, or monthly schedule to show which one client wants to see. In this section, tasks are shown with their schedule times and names too.

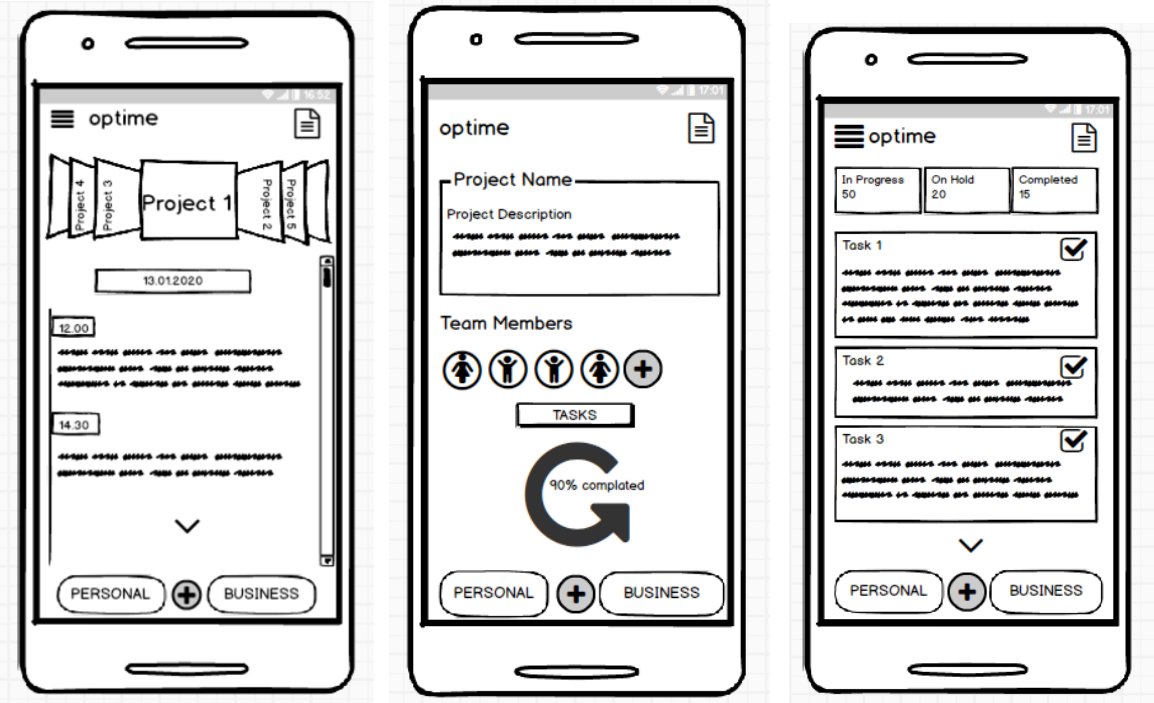


Figure 6.3 Optime Project Management Parts

In the application, there is a project management section. This part is generally used by IT professionals and students. However, this part can be used to organize works by anyone. For example, a hotel manager can create teams for different departments as front office, housekeeping, guest relations, sales, security, etc. It is because there are different tasks for different areas.

In Figure 6.3 at the left, there are projects shown at the top of the screen and it shows the project list with a slider. If any project is selected from the slider project's tasks are shown on the screen with their schedule times and names. In the center of the page, the selected project detail is shown in the Figure. Project name, description, team members, and project status shown on the page. There is also a button located called "Tasks" on the project detail page. If the client clicks the Tasks button, the project tasks list on the other page shown on the right side of the figure. Projects have some boards as In Progress, On Hold, Completed, etc. Board can be added on the screen with a plus icon. Tasks can belong to boards to distinguish.

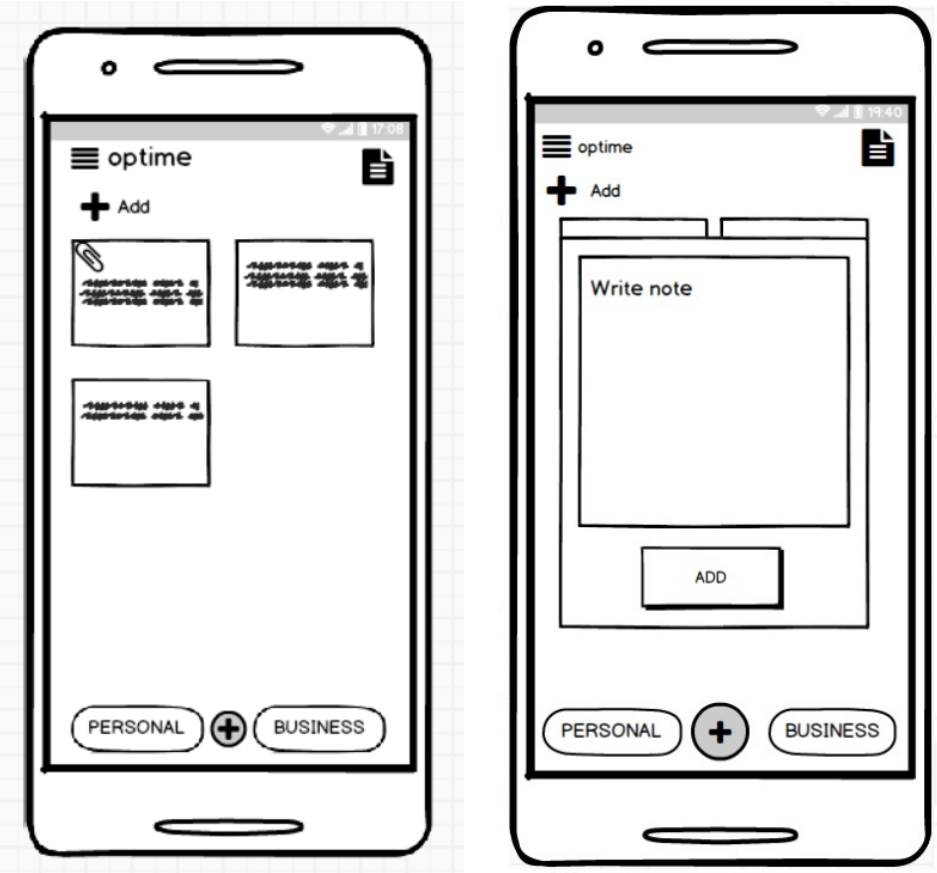


Figure 6.4 Note Pages

In Figure 6.4, notes are shown on the page shown on the left side. Notes can be set as important to show at the top of the list. There is an “Add” button located on the top of the screen and after clicking the button, add note popup is displayed on the screen as shown in the right part of the figure. One textbox and a button located on the screen to add a note. Notes are listed depend on register time and important status. Important notes are listed first, and then other notes are listed depend on register times. The last added notes are located at the top of the list.

In Figure 6.5, how to add a task is shown in the left section. There are some text boxes to type task name, description, and DateTime picker to select a due time of the task. After clicked the “Add” button, the task will be automatically added to the generated schedule algorithm. On the right part of the figure, the add project page is shown. It looks like an add task page.

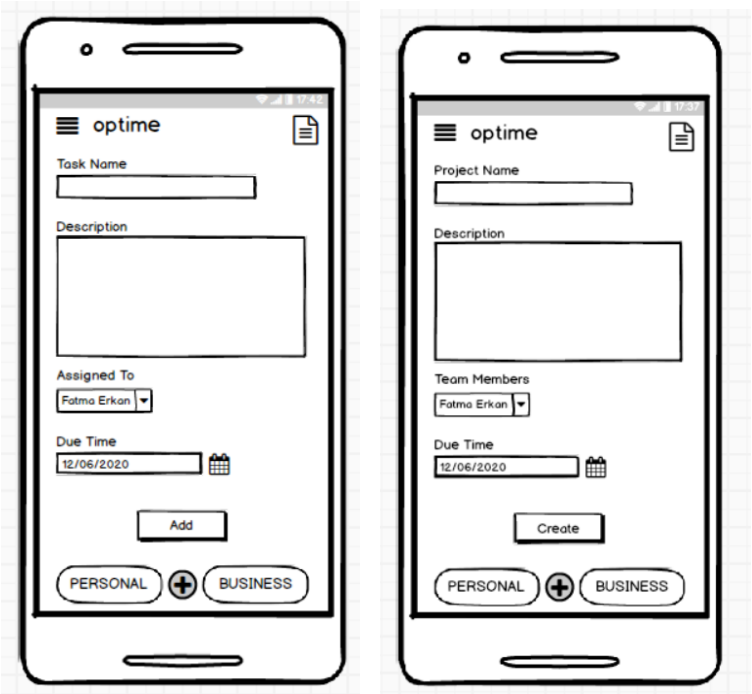


Figure 6.5 Create Task and Project Pages

7. SCREENSHOTS

Figure 7.1 Login and Register Screen

The ‘Login’ and ‘Register’ screens are shown in the Figure 7.1. User can create an account in the application to begin use the Optime. If user has an account using the ‘Login’ screen s/he can sign in the application and s/he can continue to use. If client does not have any account in the project, s/he can create a new account on the system using the ‘Register’ screen as shown in the left side of the Figure 7.1. Some important questions are asked to user such as Name and Surname, Email, Password. These are the main requirement to use application. The first opened screen is Login screen. If client have no account in the application, s/he can use REGISTER text to open Register screen.

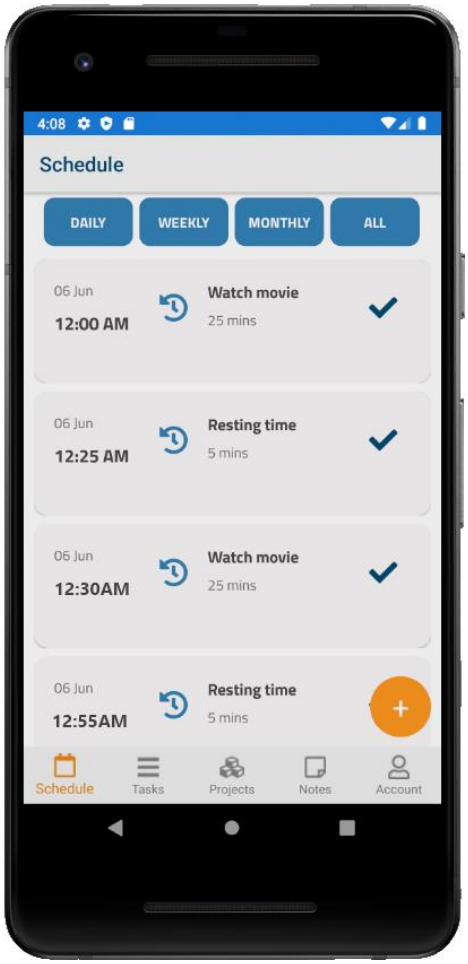


Figure 7.2 Schedule List Screen

In the Schedule List Screen, some tasks are shown with its time. Schedule Algorithm, which was explained in the Algorithm topic above, prepare this list to create plans daily, weekly, monthly for user depend on free times of user. With an icon at the right side of tasks, task can be assigned as ‘done’. The other icon is used to delay task for another time. To show tasks Task Schedule Time, Task name and Task Session Time are used.
At the right of the bottom, there is a button to open ‘Add New Task Popup’ which is shown in the Figure 7.5.

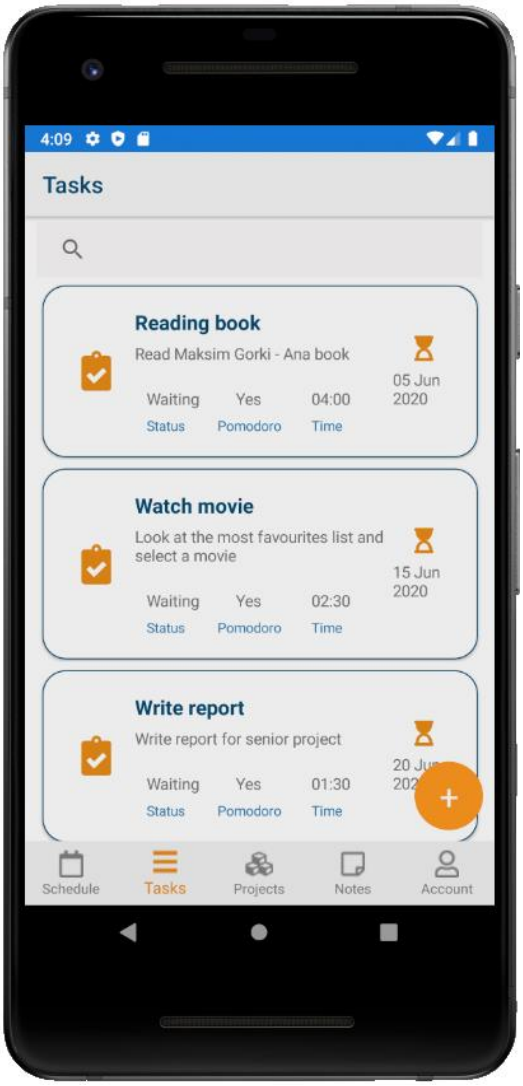


Figure 7.3 Task List Screen

In the Figure 7.3, Tasks are shown. This list is used to show all tasks. The difference between Schedule list and Task List is that Task list shows all the tasks that can be done or not done. However, Schedule list shows tasks which is not done and scheduled to be done. In this screen, Task Title, Task Detail, Due Time, Pomodoro, Status and Task Session properties are used to show tasks.

At the right of the bottom of the screen, there is a button, as shown in the Schedule List screen, to create new task. There is a search bar at the top of the page, this button is used to query tasks depend on Task title.

In the Figure 7.4, Selected task where it can be selected clicked to the Task on the Task List Screen is shown. All the properties of the task are shown in the screen. Client can also edit the task in this screen and save it using Save Button. If client want to delete the task, there is a button right side of the Save button which is called Delete to delete task.

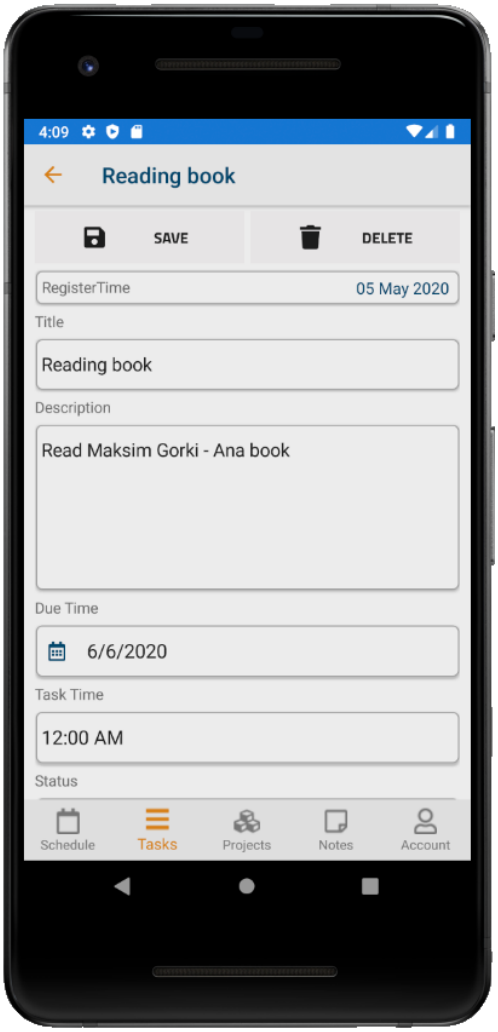


Figure 7.4 Task Detail and Update Screen

Figure 7.5 shows that how task added to the application. Added tasks are listed in the Task List page and the Schedule Algorithm works again to schedule all tasks with new added task.

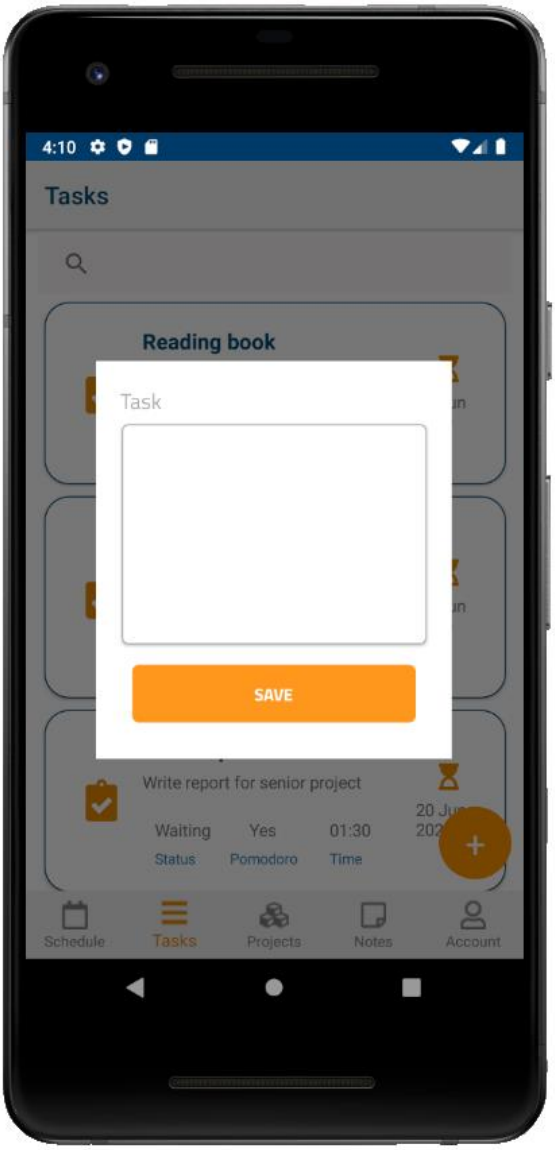


Figure 7.5 New Task Popup Screen

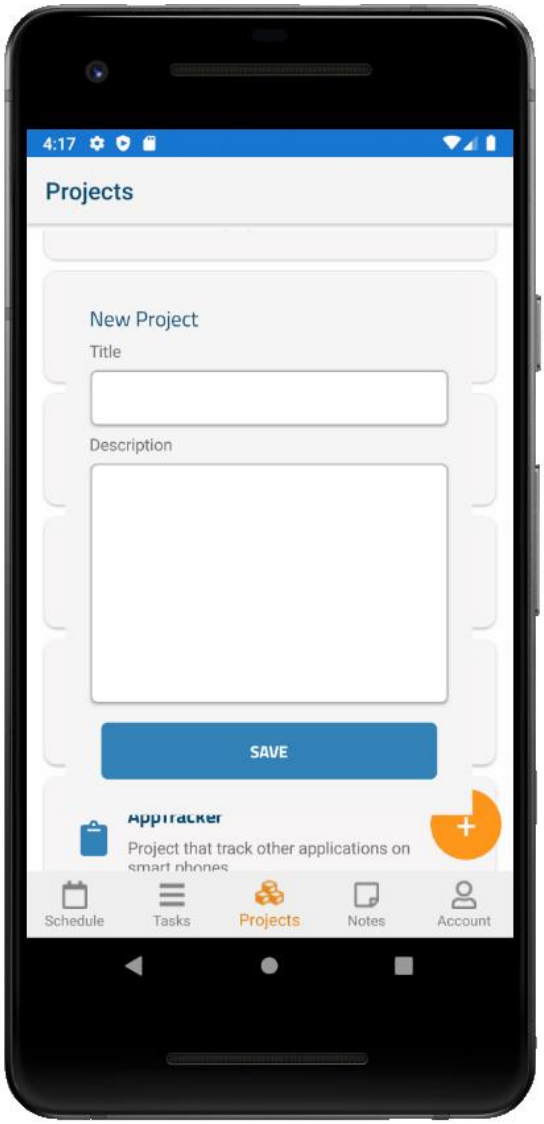


Figure 7.6 New Project Adding Popup Screen

In the New Project Adding Popup, there are two important properties called Title of the project and description of the project. Using this popup screen, new project will be added to the application.

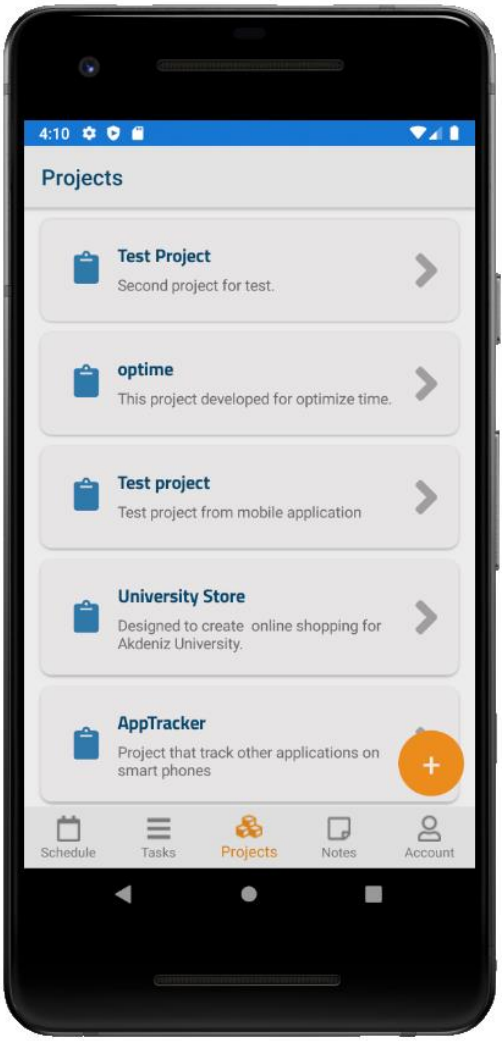


Figure 7.7 Project List Screen

In the Project List screen, projects are shown which is created by the user or if the user is a member of the project. Project title and description are shown in the list. To get detail information of any project, click the project in the list and it opens the Project Detail screen. At the right of the bottom of the Project List screen, there is a add button to add new project to application. This button opens the New Project popup as shown in the Figure 7.6.

After selecting the project in the Project List screen, Project Detail page is opened.At the top of the screen project title is located and at the right side of the navigation bar, edit button is located to edit project. In the screen, project details is located with Register time and project title properties. The screen contains some buttons called Members, Tasks, Boards, and Documents. Members button is used to list project members. Tasks button is to show project tasks that is not belong to board. Documents button is used to show documents of the project and add new document.

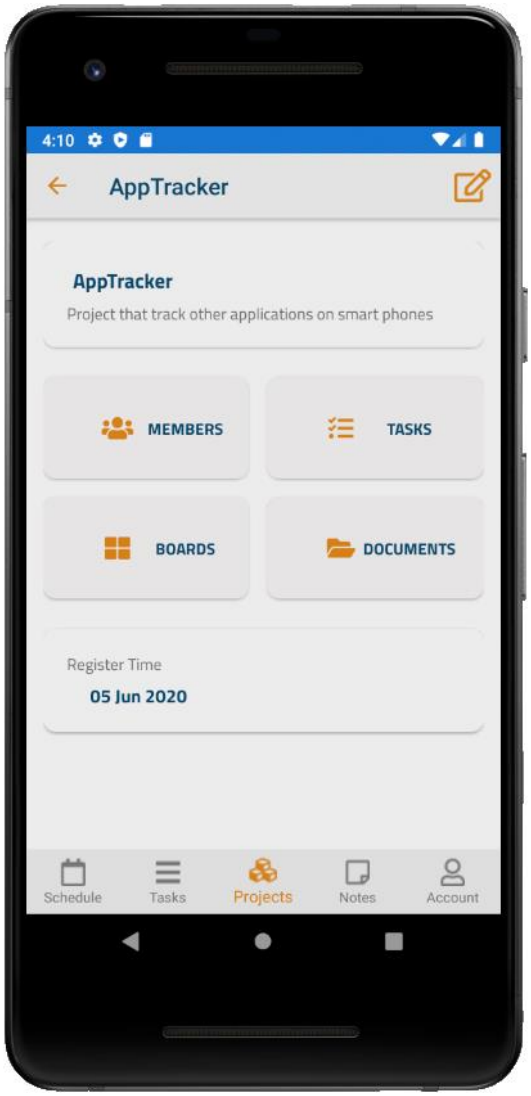


Figure 7.8 Project Detail Screen

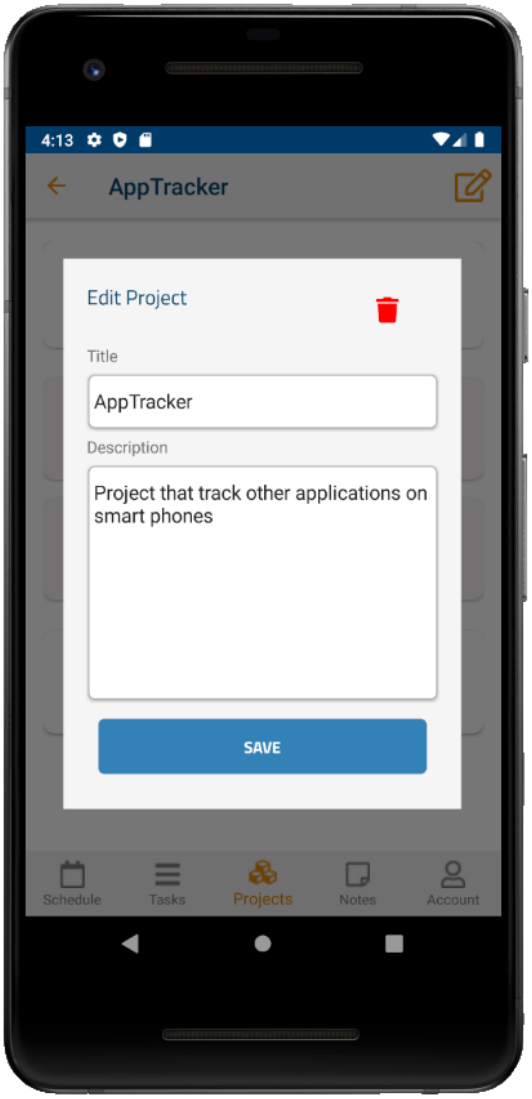


Figure 7.9 Project Edit Popup Screen

Project edit screen is used to edit selected project which is opened using Project Detail screen. If client want to delete the project in the application, there is an icon at the right top of the popup to delete project. Using Save button, project can be updated with its new information.

In the Figure 7.10, Project members are shown with its Name Surname, and Email. At the right of the bottom of the screen, there is an add button to add new members to the project. Add new members page is shown in the Figure 7.11. This page shows the users of the application and it can be queried with its name or email. Using checkbox at the right side of the username user can be selected to add project and after clicking Select Button, users are added in the project or removed from the project.

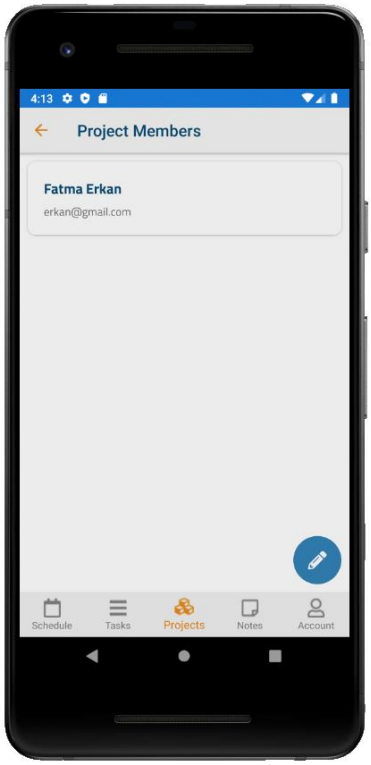


Figure 7.10 Project Members List Screen

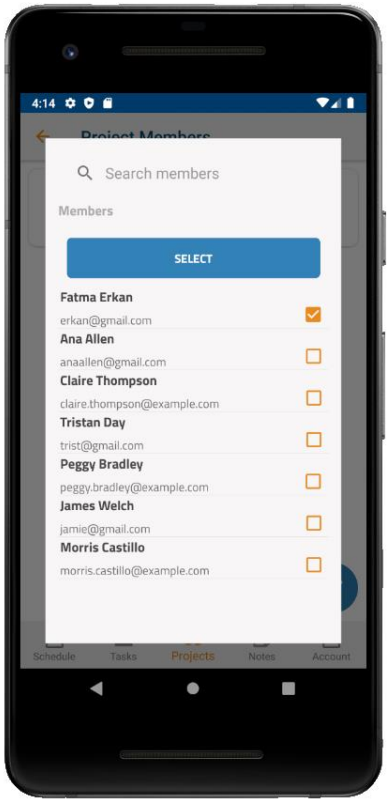


Figure 7.11 Selecting Project Members Screen

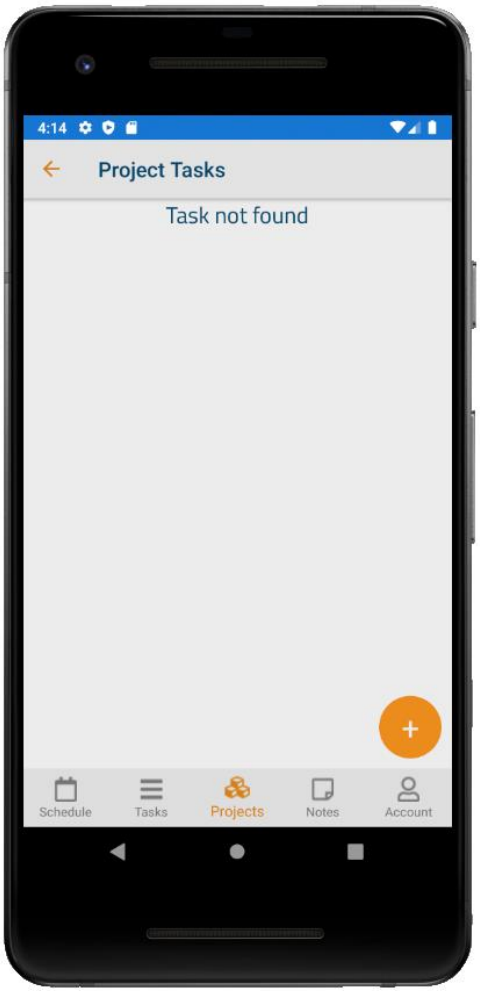


Figure 7.12 Project Tasks Screen

After clicking Tasks from the Project Detail page, Tasks are listed. At the right of the bottom, there is a add button to add new task to project.

After clicking Boards button in the Project Detail screen, Board list is shown as Figure 7.13. To add new Board, select the add button in the screen and a popup opens. The Add Board Popup is shown in the 7.14. New board will be added with its name.

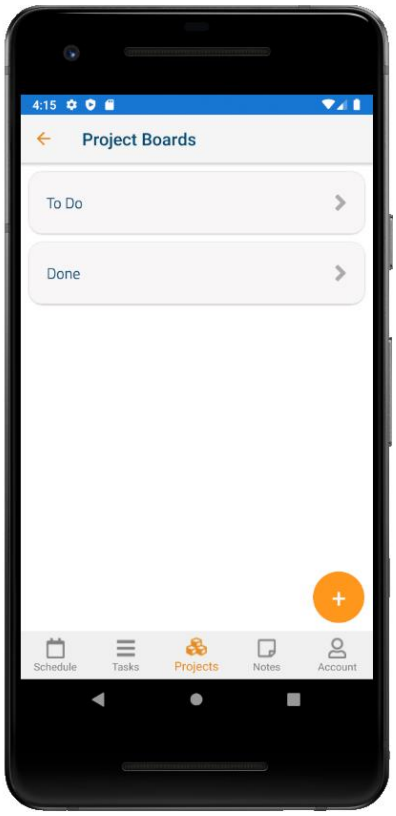


Figure 7.13 Project Board List Screen

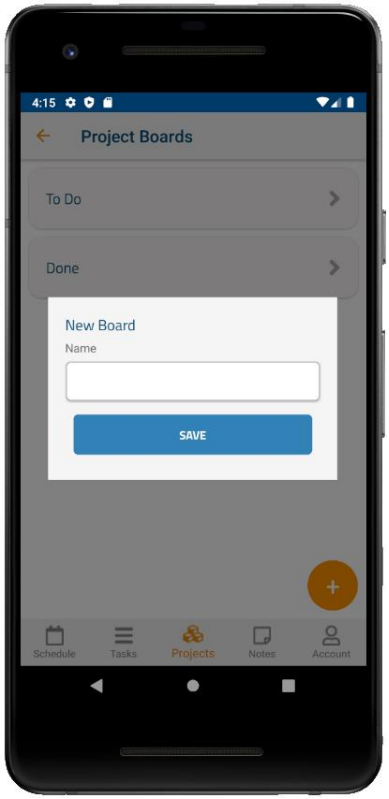


Figure 7.14 Adding New Board to Project Screen

Figure 7.15 Note List Screen

In the Figure 7.15, Note lists are shown with details of note and register time. To add new note, there is a add button in the page, clicking this page opens the screen shown in the Figure 7.16.

In the Figure 7.17, description of note is shown. Description of note can be changed in this screen. The note can be assigned to important to list at the top of the page. At the top of the page, there are two buttons. First one is named as Save, to change note. Second one is called as Delete, to delete note from the list. At the bottom of the page, register time of the note can be seen.

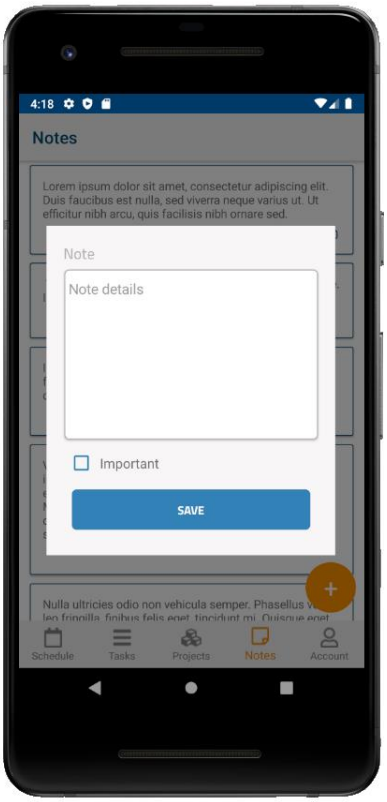


Figure 7.16 Adding New Note Screen

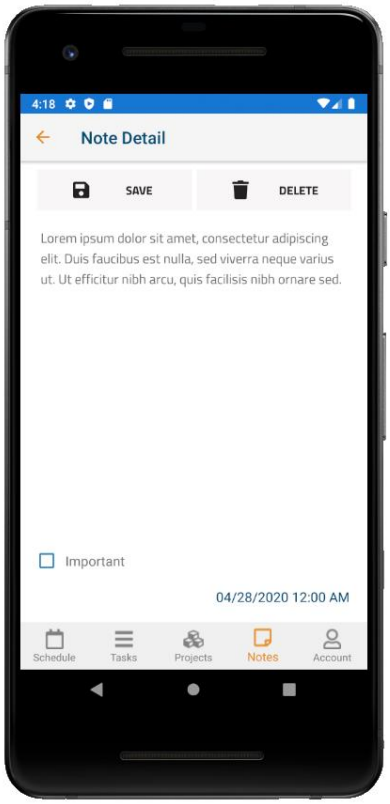


Figure 7.17 Note Detail and Update Screen

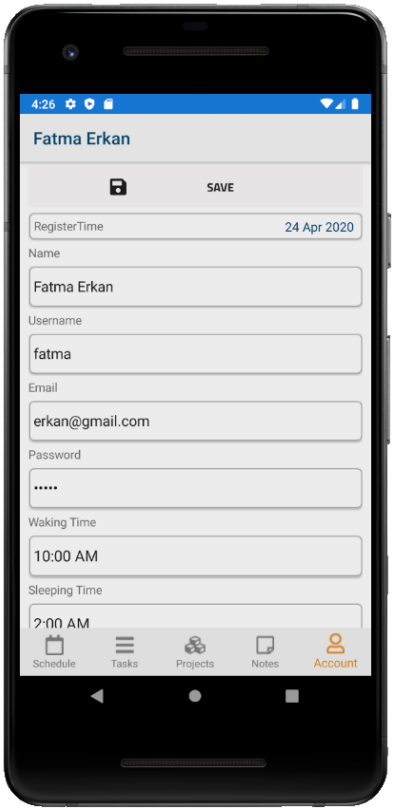


Figure 7.18 Account Detail and Update Screen

There is a screen called Account in the project to show information of client as shown in the Figure 7.18. User information can be updated with button called Save.

8. IMPLEMENTATION

The database diagram is shown in Figure 3.1. There are some tables called Users, UserDetails, Notes, Tasks, TaskDetails, SubTasks, Projects, ProjectToUsers, Documentations, TaskToUsers, Boards.

Figure 3.1 Database Diagram

Multi-layered architecture is used to develop Optime application. Layers are BLL which means Business Logic Layer, DAL that stands for Data Access Layer, Core Layer, Entities, WebApi, Mobile and Android layer as shown in Figure 3.2.

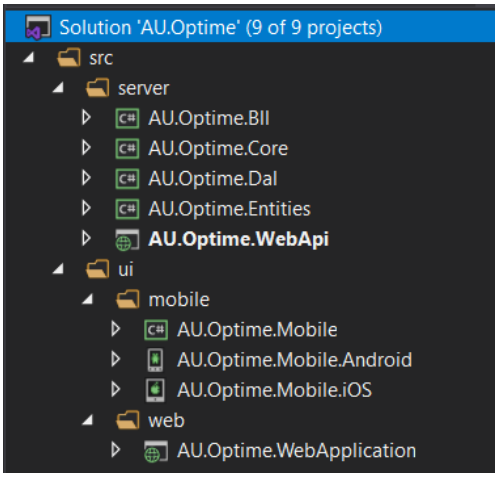


Figure 3.2 Layered Architecture

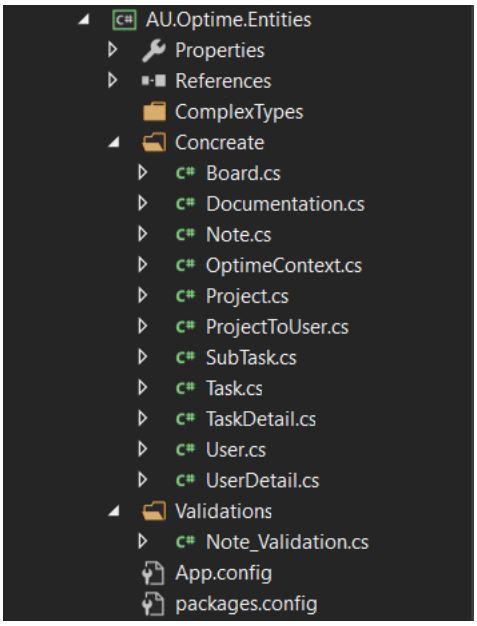


Figure 3.3 Entities Layer

In Figure 3.3, Entities layer is located. In the folder Concreate contains Entities that comes from the database connection.

```
/// <summary>
/// OptimeContext is the primary class that is responsible for interacting with the database.
/// </summary>
14 references
public partial class OptimeContext : DbContext {
    5 references
    public OptimeContext()
        : base("name=OptimeContext") {
    }

    // The entity set of type DbSet<TEntity> for all the entities.
    0 references
    public virtual DbSet<Board> Boards { get; set; }
    0 references
    public virtual DbSet<Documentation> Documentations { get; set; }
    0 references
    public virtual DbSet<Note> Notes { get; set; }
    0 references
    public virtual DbSet<Project> Projects { get; set; }
    0 references
    public virtual DbSet<ProjectToUser> ProjectToUsers { get; set; }
    0 references
    public virtual DbSet<SubTask> SubTasks { get; set; }
    0 references
    public virtual DbSet<TaskDetail> TaskDetails { get; set; }
    3 references
    public virtual DbSet<Task> Tasks { get; set; }
    0 references
    public virtual DbSet<UserDetail> UserDetails { get; set; }
    2 references
    public virtual DbSet<User> Users { get; set; }
```

Figure 3.4 OptimeContext Class

OptimeContext class is placed in Figure 3.4 which is derived DbContext. It contains tables as a property in which data type is DbSet.

```
/// <summary>
/// Note class is used to store Notes in the database and its usage in the code as Note class
/// Metadatatype is used to connect validation rules of the note table
/// </summary>

[MetadataType(typeof(Note_Validation))]
22 references
public partial class Note : IEntity {
    7 references
    public int NoteId { get; set; }

    7 references
    public int? UserId { get; set; }
    //Note detail text
    7 references
    public string Description { get; set; }

    6 references
    public bool? IsImportant { get; set; }

    [Column(TypeName = "smalldatetime")]
    5 references
    public DateTime? RegisterTime { get; set; }
    //Every note depend on a user.
    0 references
    public virtual User User { get; set; }
}
```

Figure 3.5 Note Entity

In Figure 3.5, Note entity is shown with its properties. It is the view of Notes table from the database. Every table has its own view in its own class.

```
//Configure validation rules of Note table
[Table("Note")]
1 reference
public class Note_Validation {
    0 references
    public int NoteId { get; set; }

    0 references
    public int UserId { get; set; }

    [Display(Name ="Description")]
    [Required(ErrorMessage = "Description is required.")]
    0 references
    public string Description { get; set; }

    [Display(Name ="Important")]
    0 references
    public bool? IsImportant { get; set; }

    [Column(TypeName = "smalldatetime")]
    0 references
    public DateTime RegisterTime { get; set; }

    0 references
    public virtual User User { get; set; }
}
```

Figure 3.6 Note_Validation Class

Note_Validation is the class that identify rules of Note entity. It is connected to each other with "[Table("Note")]" property. Rules are defined over the property such as Required data annotation in the Description property. These operations are shown in the Figure 3.6.

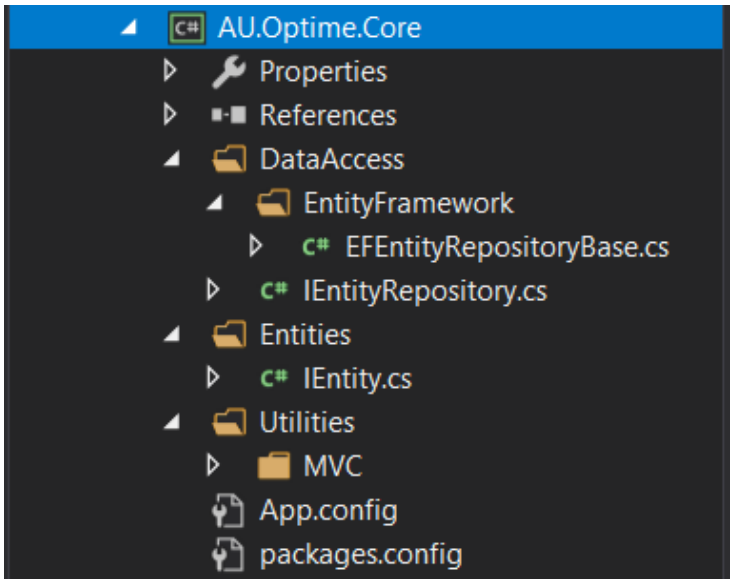


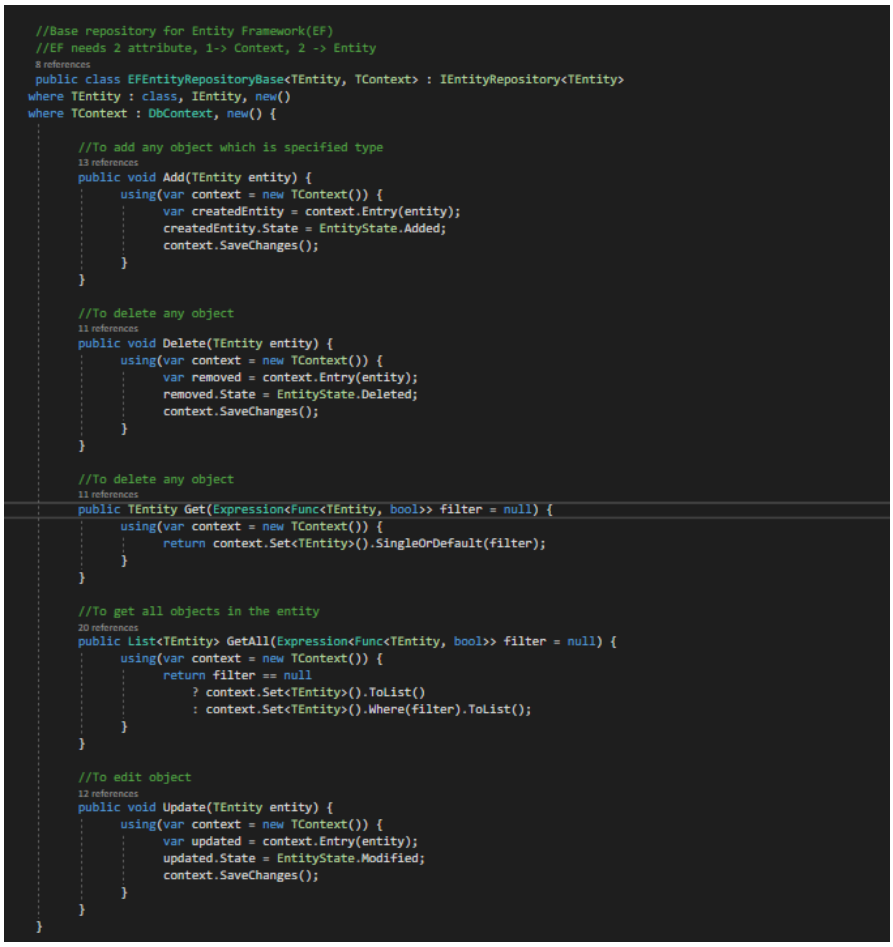
Figure 3.7 Core Layer

Core layer is defined for general features of the project in the Figure 3.7.

```
//To create a generic skeleton which run with all database table objects. Generic structure use T.
//Where all basic operations are defined.
9 references
public interface IEntityRepository<T> where T : class, IEntity, new() {

    //Return one object. It return specificed typed. So, it begins with T.  To filter object, Expression structure is used.
    11 references
    T Get(Expression<Func<T, bool>> filter = null);

    // To return all objects which is type of specified. To filter these objects, Expression structure is used.
    20 references
    List<T> GetAll(Expression<Func<T, bool>> filter = null);

    //Used for creating an object. It returns nothing
    13 references
    void Add(T entity);

    //To update object. It returns nothing
    12 references
    void Update(T entity);

    //To delete an object. It deletes with object id
    11 references
    void Delete(T entity);
}
```

Figure 3.8 IEntityRepository Interface

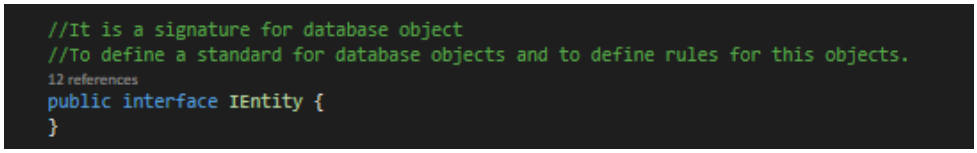
IEntityRepository interface is defined for the skeleton of Reposity class. It is a generic interface. The common operations of entities are defined in the interface as seen in the Figure 3.8.



```
//Base repository for Entity Framework(EF)
//EF needs 2 attribute, 1-> Context, 2 -> Entity
8 references
public class EFEntityRepositoryBase<TEntity, TContext> : IEntityRepository<TEntity>
where TEntity : class, IEntity, new()
where TContext : DbContext, new() {

    //To add any object which is specified type
    13 references
    public void Add(TEntity entity) {
        using(var context = new TContext()) {
            var createdEntity = context.Entry(entity);
            createdEntity.State = EntityState.Added;
            context.SaveChanges();
        }
    }

    //To delete any object
    11 references
    public void Delete(TEntity entity) {
        using(var context = new TContext()) {
            var removed = context.Entry(entity);
            removed.State = EntityState.Deleted;
            context.SaveChanges();
        }
    }

    //To delete any object
    11 references
    public TEntity Get(Expression<Func<TEntity, bool>> filter = null) {
        using(var context = new TContext()) {
            return context.Set<TEntity>().SingleOrDefault(filter);
        }
    }

    //To get all objects in the entity
    20 references
    public List<TEntity> GetAll(Expression<Func<TEntity, bool>> filter = null) {
        using(var context = new TContext()) {
            return filter == null
                ? context.Set<TEntity>().ToList()
                : context.Set<TEntity>().Where(filter).ToList();
        }
    }

    //To edit object
    12 references
    public void Update(TEntity entity) {
        using(var context = new TContext()) {
            var updated = context.Entry(entity);
            updated.State = EntityState.Modified;
            context.SaveChanges();
        }
    }
}
```

Figure 3.9 EfEntityRepositoryBase Class

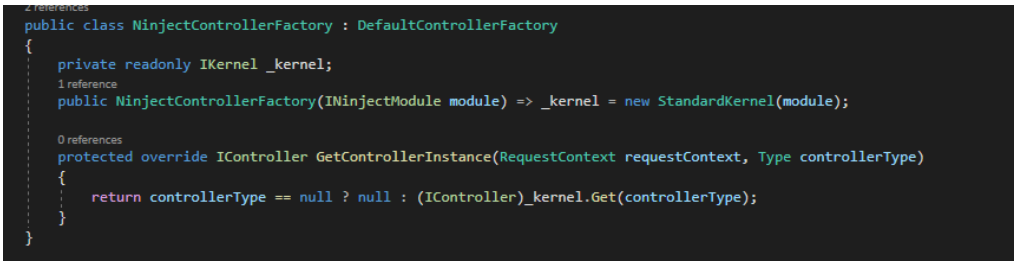
EfEntityRepositoryBase which is shown in the Figure 3.9 used to define common operations of entities like Add, Delete, Get,GetAll and Update. It is inherited by IEntityRepository which is skeleton for this class.



```
//It is a signature for database object
//To define a standard for database objects and to define rules for this objects.
12 references
public interface IEntity {
}
```

Figure 3.10 IEntity Interface

IEntity is defined for base of entity as shown in the Figure 3.10.



```
2 references
public class NinjectControllerFactory : DefaultControllerFactory
{
    private readonly IKernel _kernel;
    1 reference
    public NinjectControllerFactory(INinjectModule module) => _kernel = new StandardKernel(module);

    0 references
    protected override IController GetControllerInstance(RequestContext requestContext, Type controllerType)
    {
        return controllerType == null ? null : (IController)_kernel.Get(controllerType);
    }
}
```

Figure 3.11 NinjectControllerFactory

NinjectControllerFactory is come from Ninject package which can be installed using Nuget Package Manager. It is used by Controllers.

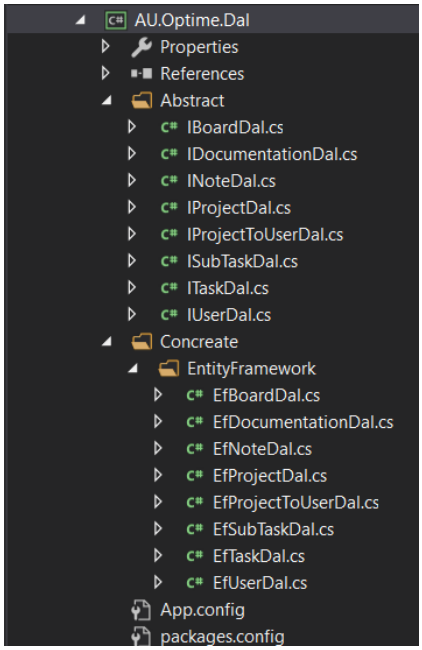


Figure 3.12 Data Access Layer

Data Access Layer is defined in the Figure 3.12. It contains data operations for entities.

```
//To implement Note objects operations
4 references
public interface INoteDal : IEntityRepository<Note> {
    //Specific operations
}
```

Figure 3.13 INoteDal Interface

INoteDal interface is shown in the Figure 3.13 which is the skeleton of EfNoteDal class and it is derived from IEntityRepository.

```
//To execute CRUD operations depend on Note object
1 reference
public class EfNoteDal : EFEntityRepositoryBase<Note, OptimeContext>, INoteDal {
    //specific operations
}
```

Figure 3.14 EfNoteDal Class

EfNoteDal is created for Note entity operations in the Figure 3.14. It is inherited from EfEntityRepositoryBase because of common operations.

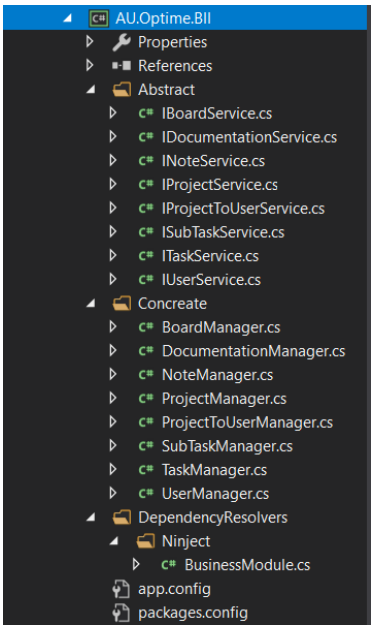


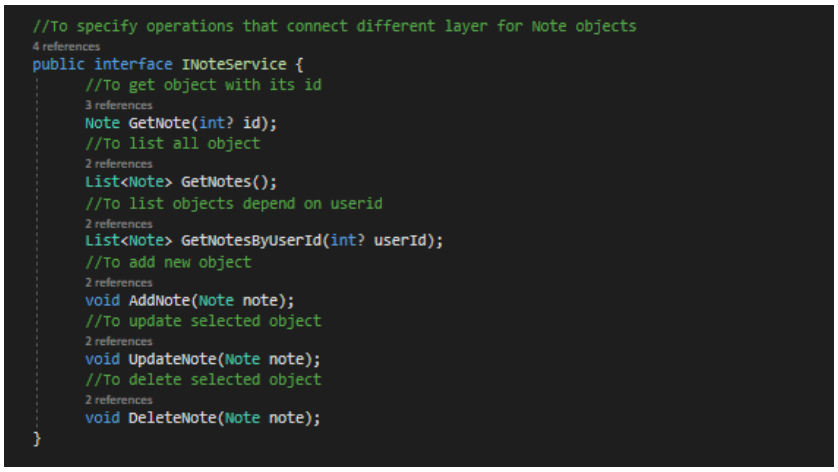
Figure 3.15 Business Logic Layer

Business Logic Layer is defined in the Figure 3.15. This layer is defined to catch errors before DAL layer.

```
1 reference
public class BusinessModule : NinjectModule {
    //To bind object manager to object service and to bind object operations with Entity Framework (EF) technology module
    0 references
    public override void Load() {

        //Note
        Bind<INoteService>().To<NoteManager>().InSingletonScope();
        Bind<INoteDal>().To<EfNoteDal>();

        //Board
        Bind<IBoardService>().To<BoardManager>().InSingletonScope();
        Bind<IBoardDal>().To<EfBoardDal>();

        //Documentation
        Bind<IDocumentationService>().To<DocumentationManager>().InSingletonScope();
        Bind<IDocumentationDal>().To<EfDocumentationDal>();

        //Project
        Bind<IProjectService>().To<ProjectManager>().InSingletonScope();
        Bind<IProjectDal>().To<EfProjectDal>();

        //Task
        Bind<ITaskService>().To<TaskManager>().InSingletonScope();
        Bind<ITaskDal>().To<EfTaskDal>();

        //User
        Bind<IUserService>().To<UserManager>().InSingletonScope();
        Bind<IUserDal>().To<EfUserDal>();

        //Subtask
        Bind<ISubTaskService>().To<SubTaskManager>().InSingletonScope();
        Bind<ISubTaskDal>().To<EfSubTaskDal>();

        //ProjectToUser
        Bind<IProjectToUserService>().To<ProjectToUserManager>().InSingletonScope();
        Bind<IProjectToUserDal>().To<EfProjectToUserDal>();
    }
}
```

Figure 3.16 Business Module

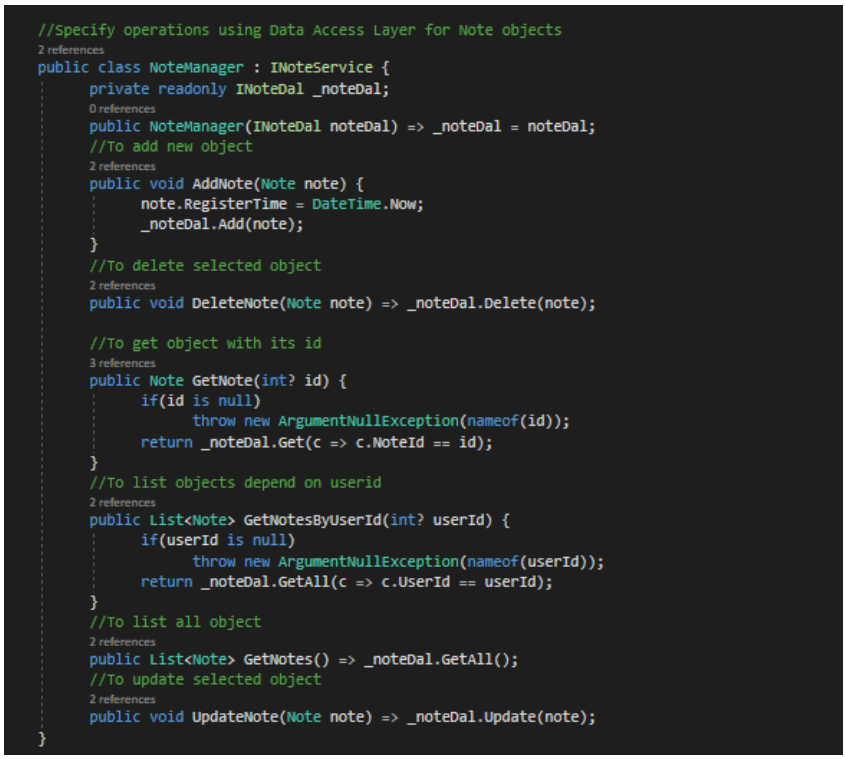
Business Module is defined to bind NoteService and NoteManager to each other, and to bind INoteDal to EfNoteDal. The class is derived from NinjectModule which is comes from Ninject package in the Figure 3.16.



```
//To specify operations that connect different layer for Note objects
4 references
public interface INoteService {
    //To get object with its id
    3 references
    Note GetNote(int? id);
    //To list all object
    2 references
    List<Note> GetNotes();
    //To list objects depend on userid
    2 references
    List<Note> GetNotesByUserId(int? userId);
    //To add new object
    2 references
    void AddNote(Note note);
    //To update selected object
    2 references
    void UpdateNote(Note note);
    //To delete selected object
    2 references
    void DeleteNote(Note note);
}
```

Figure 3.17 INoteService Interface

INoteService is defined operations of Note entity as seen in the Figure 3.17. There are some basic operations shown.



```
//Specify operations using Data Access Layer for Note objects
2 references
public class NoteManager : INoteService {
    private readonly INoteDal _noteDal;
    0 references
    public NoteManager(INoteDal noteDal) => _noteDal = noteDal;
    //To add new object
    2 references
    public void AddNote(Note note) {
        note.RegisterTime = DateTime.Now;
        _noteDal.Add(note);
    }
    //To delete selected object
    2 references
    public void DeleteNote(Note note) => _noteDal.Delete(note);

    //To get object with its id
    3 references
    public Note GetNote(int? id) {
        if(id is null)
            throw new ArgumentNullException(nameof(id));
        return _noteDal.Get(c => c.NoteId == id);
    }
    //To list objects depend on userid
    2 references
    public List<Note> GetNotesByUserId(int? userId) {
        if(userId is null)
            throw new ArgumentNullException(nameof(userId));
        return _noteDal.GetAll(c => c.UserId == userId);
    }
    //To list all object
    2 references
    public List<Note> GetNotes() => _noteDal.GetAll();
    //To update selected object
    2 references
    public void UpdateNote(Note note) => _noteDal.Update(note);
}
```

Figure 3.18 NoteManager Class

NoteManager is inherited from INoteService to fill inside of the methods which implemented from INoteService interface.

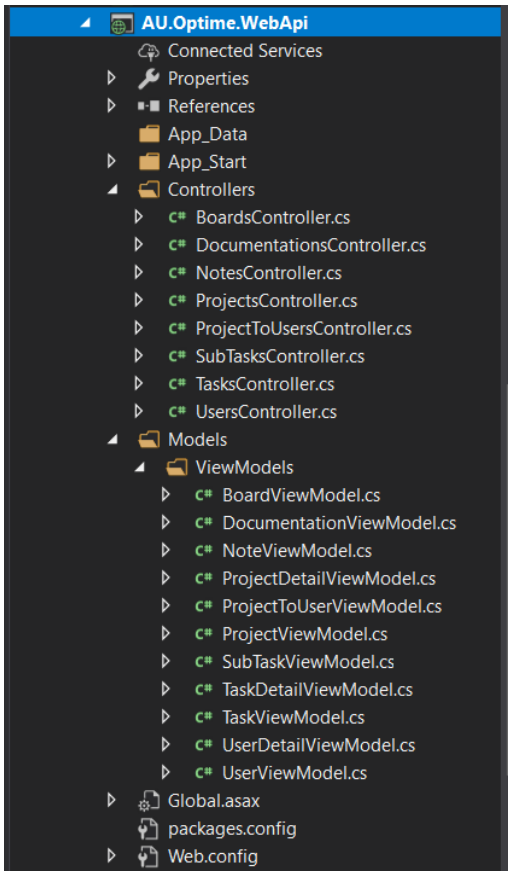


Figure 3.19 WebApi Layer

Web Api was created to connect Mobile device with database operations.

```
//To build services to reach Note objects from mobile devices
[RoutePrefix("api/notes")]
1 reference
public class NotesController : BaseController {
    ApiResult result = new ApiResult();
    private readonly INoteService _noteService;
    0 references
    public NotesController(INoteService noteService) => _noteService = noteService;

    // GET: api/Notes
    [HttpGet, Route("")]
    0 references
    public ApiResult GetNotes() {
        try {
            result.Result = true;
            List<NoteViewModel> notes = _noteService.GetNotes().Select(c => new NoteViewModel { Description = c.Description, IsImportant = c.IsImportant, NoteId = c.NoteId, RegisterTime = c.RegisterTime, UserId = c.UserId }).ToList();
            result.Data = notes;
            result.Message = "Success.";

        } catch(Exception ex) {
            result.Result = false;
            result.Message = $"Error : {ex.Message}";
        }
        return result;

    }

    [HttpGet, Route("getnotesbyuser/{userId:int}")]
    0 references
    public ApiResult GetNotesByUser(int userId) {
        try {
            result.Result = true;
            List<NoteViewModel> notes = _noteService.GetNotesByUserId(userId).Select(c => new NoteViewModel { Description = c.Description, IsImportant = c.IsImportant, NoteId = c.NoteId, RegisterTime = c.RegisterTime, UserId =
                c.UserId }).ToList();
            result.Data = notes;
            result.Message = "Success.";
        } catch(Exception ex) {
            result.Result = false;
            result.Message = $"Error : {ex.Message}";
        }
        return result;
    }
```

Figure 3.20 NotesController Class

```
    // PUT: api/Notes/5
    [ResponseType(typeof(void)), HttpPut, Route("editnote")]
    0 references
    public IHttpActionResult PutNote([FromBody]NoteViewModel model)
    {
        if(!ModelState.IsValid)
            return BadRequest(ModelState);

        _noteService.UpdateNote(new Note { Description = model.Description, IsImportant = model.IsImportant, NoteId = model.NoteId, UserId = model.UserId });

        return StatusCode(HttpStatusCode.NoContent);
    }

    // POST: api/Notes
    [ResponseType(typeof(NoteViewModel)), HttpPost, Route("addnote")]
    0 references
    public IHttpActionResult PostNote(NoteViewModel model)
    {
        if(!ModelState.IsValid)
            return BadRequest(ModelState);

        _noteService.AddNote(new Note { Description = model.Description, IsImportant = model.IsImportant, UserId = model.UserId });

        return CreatedAtRoute("DefaultApi", new { id = model.NoteId }, model);
    }

    // DELETE: api/Notes/5
    [ResponseType(typeof(NoteViewModel)), HttpDelete, Route("deletenote/{id:int}")]
    0 references
    public IHttpActionResult DeleteNote(int id)
    {
        _noteService.DeleteNote(new Note() { NoteId = id });
        return Ok();
    }
}
```

Figure 3.20 NotesController Class

NotesControllers class is used to connect Note methods between mobile and BLL. It is derived from ApiController as shown in the Figure 3.20.

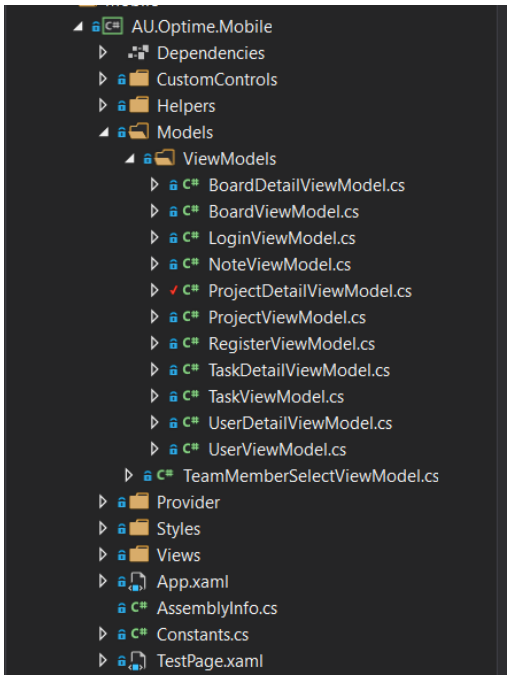


Figure 3.21 Mobile Layer

In the Mobile layer, mobile design is located. There are folders such as CustomControls that contains some items belong to user interface. Helpers models which includes some helper classes, Models that will be used for getting data from WebApi.

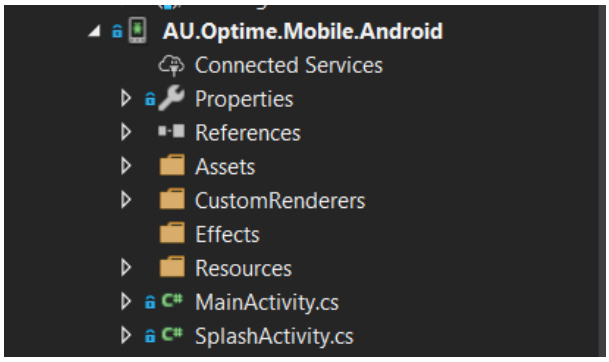


Figure 3.22 Android Layer

In the Figure 3.22, Android layer is located for platform specific operations. There are some folders as CustomRenderers that corresponds to CustomControls in the Mobile layer.

```
public class CustomEditor  : Editor{
        public CustomEditor() {

        }
}
```

Figure 3.23 CustomEditor Class

CustomEditor is identified for add new properties to Editor item in the Android. CustomEditor is inherited from Editor class in the Mobile layer.

```
public class CustomEditorRenderer : EditorRenderer {
        protected override void OnElementChanged(ElementChangedEventArgs<Editor> e) {
                base.OnElementChanged(e);

                if(Control != null) {
                        GradientDrawable gd = new GradientDrawable();
                        gd.SetColor(global::Android.Graphics.Color.Transparent);
                        this.Control.SetBackgroundDrawable(gd);
                        this.Control.SetRawInputType(InputTypes.TextFlagNoSuggestions);
                        //  Control.SetHintTextColor(ColorStateList.ValueOf(global::Android.Graphics.Color.White));
                }
        }
}
```

Figure 3.24 CustomEditorRenderer Class

CustomEditorRenderer is defined in the Android layer for platform specific operations. This class is created to get transparent background in the Editor.

```
public class NoteViewModel {
        public int NoteId { get; set; }
        public int UserId { get; set; }
        public string Description { get; set; }
        public bool? IsImportant { get; set; }
        public DateTime RegisterTime { get; set; }
}
```

Figure 3.25 NoteViewModel

NoteViewModel is defined to get data from web api. It is used to create ModelViewViewModel approach.

```
[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class ProjectsPage : ContentPage {
	ProjectManager projectManager = new ProjectManager();
	IEnumerable<ProjectViewModel> projects = new List<ProjectViewModel>();

	public ProjectsPage() {
		InitializeComponent();
		listIndicator.IsRunning = true;
		listIndicator.IsVisible = true;
		LoadDataAsync();
	}

	public async void LoadDataAsync() {
		projects = await projectManager.GetProjectsByUser(Constants.userid);

		if(projects.Count() < 1) {
			emptyMessage.IsVisible = true;
			emptyMessage.Text = "Project not found";
			listIndicator.IsVisible = false;
			listIndicator.IsRunning = false;
		} else {
			emptyMessage.IsVisible = false;
			listIndicator.IsVisible = false;
			listIndicator.IsRunning = false;
		}


		lstProjects.BindingContext = projects;
	}

	private void OnItemSelected(object sender, SelectedItemChangedEventArgs e) {
		if(e.SelectedItem == null)
			return;

		ProjectViewModel selected = (ProjectViewModel)e.SelectedItem;
		Navigation.PushAsync(new ProjectDetailPage(selected.ProjectId), false);
		//Navigation.PushAsync(new ProjectDetailPage(selected.ProjectId), false);
		ListView listView = (ListView)sender;
		listView.SelectedItem = null;
	}

	private void ProjectSearchTextChanged(object sender, TextChangedEventArgs e) {
		if(string.IsNullOrEmpty(e.NewTextValue)) {
			lstProjects.ItemsSource = projects;
		} else {
			lstProjects.ItemsSource = projects.Where(x => x.Name.ToUpper().Contains(e.NewTextValue.ToUpper()));
		}
	}
```

Figure 3.26 Project List Page Class

ProjectsPage class is defined to get data from web api using NoteViewModel. Some controls can be controlled in the class which are defined in the Xaml page. Xaml page is the design language for Xamarin forms.

```
<ContentPage.Content>
    <ContentView x:Name="contentView">
        <!-- MAIN CONTENT PANEL -->
        <Grid Padding="10">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="*" />
            </Grid.RowDefinitions>
            <Grid Grid.Row="1" HorizontalOptions="Center">
                <Grid.RowDefinitions>
                    <RowDefinition Height="*"></RowDefinition>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="5*"></ColumnDefinition>
                    <ColumnDefinition Width="*"></ColumnDefinition>
                </Grid.ColumnDefinitions>

                <Grid Grid.Row="0" Grid.Column="0" HorizontalOptions="FillAndExpand" VerticalOptions="Center">
                    <controls:CustomSearchBar TextColor="Black" x:Name="projectSearch" TextChanged="ProjectSearchTextChanged" BackgroundColor="#F8F8F8" VerticalOptions="Center"/>
                </Grid>
                <Grid Grid.Row="0" Grid.Column="1">
                    <Button HorizontalOptions="End" Text="&#xf067;" Style="{StaticResource menuBtnStyle}" BackgroundColor="Transparent" Clicked="OnAddProjectClicked"
                      TextColor="#F39C12"  FontSize="Title" />
                </Grid>
            </Grid>

            <StackLayout Grid.Row="2">
                <ActivityIndicator IsRunning="{Binding Source={x:Reference lstProjects}, Path=IsLoading}"  AbsoluteLayout.LayoutFlags="PositionProportional"
                  AbsoluteLayout.LayoutBounds="0,0,-1,-1" x:Name="listIndicator"  VerticalOptions="Center" HorizontalOptions="Center"/>

                <StackLayout VerticalOptions="CenterAndExpand" HorizontalOptions="CenterAndExpand">
                    <Label  x:Name="emptyMessage" FontFamily="{StaticResource TitilliumRegular}" VerticalOptions="CenterAndExpand" VerticalTextAlignment="Center"
                      HorizontalOptions="CenterAndExpand" HorizontalTextAlignment="Center" FontAttributes="Bold" FontSize="Large" Style="{StaticResource grayLblStyle}"
                      IsVisible="False" />
                </StackLayout>
            </StackLayout>
```

Figure 3.27 Project List Page Xaml Codes

```
<ListView Grid.Row="3" x:Name="lstProjects" ItemsSource="{Binding .}" HasUnevenRows="true" ItemSelected="OnItemSelected" >

    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="100"></RowDefinition>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="2*"/>
                        <ColumnDefinition Width="9*"/>
                        <ColumnDefinition Width="*"/>
                    </Grid.ColumnDefinitions>

                    <Grid Grid.Row="0" Grid.Column="0">
                        <Label Text="&#xf328;" FontFamily="{StaticResource FontAwesomeRegular}" TextColor="#0754b0" HorizontalOptions="Center" VerticalOptions="Center"
                          FontSize="Title" />
                    </Grid>
                    <Grid Grid.Row="0" Grid.Column="1">
                        <StackLayout VerticalOptions="CenterAndExpand">
                            <Label Text="{Binding Name}" Style="{StaticResource ContentTextStyle}"/>
                            <Label Text="{Binding ShortDescription}" HorizontalOptions="FillAndExpand"/>
                        </StackLayout>
                    </Grid>
                    <Grid Grid.Row="0" Grid.Column="2">
                        <Label Text="&#xf105;" FontFamily="{StaticResource FontAwesomeRegular}" Style="{StaticResource grayLblStyle}" VerticalOptions="Center"
                          HorizontalOptions="Start" FontSize="Title" BackgroundColor="Transparent" />
                    </Grid>

                </Grid>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
</Grid>
</ContentView>

</ContentPage.Content>
```

Figure 3.27 Project List Page Xaml Codes

ProjectsPage Xaml codes located in the Figure 3.27. With Binding operation, whole data come from class to Xaml page.

```
/// <summary>
/// Schedule Algorithm the create a plan using tasks
/// </summary>
4 references
public class Scheduler {
    EfUserDal userDal = new EfUserDal();
    EfTaskDal taskDal = new EfTaskDal();
    int _userId;

    TimeSpan wakingTime;
    TimeSpan sleepingTime;
    TimeSpan beginWorkTime;
    TimeSpan endWorkTime;
    TimeSpan busyTime;


    //To get user with userId
    2 references
    public User GetUser() {
        return userDal.GetUserwithDetails(_userId);
    }

    //Get Task and Details depend on user
    2 references
    public List<Task> GetTasks() {
        #region
        return taskDal.GetTaskwithFullDetailsByUser(_userId).Where(x => x.Status == false).ToList();

    }
```

Figure 3.28 Scheduler Algorithm

Scheduler algorithm of the project is defined in the Figure 3.28. In this section, some important properties were identified. To use scheduler algorithm, user is important because this algorithm generate a plan depend on user. Then, tasks were got depend on user.

```
//To calculate available time of the user
2 references
public TimeSpan AvailableTimeCalculator() {
      TimeSpan availableTimeInDay = new TimeSpan();
      User user = GetUser();
      //Default waking time
      wakingTime = new TimeSpan(8, 0, 0);
      //Default sleeping time
      sleepingTime = new TimeSpan(23, 0, 0);
      //Assign working time as not defined. Assume user not working in default
      beginWorkTime = new TimeSpan(0, 0, 0);
      endWorkTime = new TimeSpan(0, 0, 0);
      busyTime = new TimeSpan(0, 0, 0);

      //Assigning waking time of user
      if(user.UserDetail.WakingTime != null) {
            wakingTime = TimeSpan.Parse(user.UserDetail.WakingTime.ToString());
      }
      //Assigning sleeping time of user
      if(user.UserDetail.SleepingTime != null) {
            sleepingTime = TimeSpan.Parse(user.UserDetail.SleepingTime.ToString());
      }

      //Assigning working time of user
      if(user.UserDetail.BeginWorkingTime != null && user.UserDetail.EndWorkingTime != null) {
            beginWorkTime = TimeSpan.Parse(user.UserDetail.BeginWorkingTime.ToString());
            endWorkTime = TimeSpan.Parse(user.UserDetail.EndWorkingTime.ToString());
            //Calculate busy time of user
            busyTime = endWorkTime - beginWorkTime;
      }

      //If user sleeps after 00.00
      if(wakingTime > sleepingTime) {
            TimeSpan overTime = sleepingTime - new TimeSpan(0, 0, 0); //Calculating overtime
            sleepingTime = new TimeSpan(23, 59, 0); // Assume user sleeps in 23.59
            //Calculate available time in the day
            availableTimeInDay = sleepingTime - wakingTime - busyTime + overTime + new TimeSpan(0, 1, 0);
      }
      //If user sleeps before 00.00
      else {
            availableTimeInDay = sleepingTime - wakingTime - busyTime;
      }

      return availableTimeInDay;
}
```

Figure 3.30 The Rest Part of Schedule Algorithm

In the Figure 3.30, available time of the user is calculated to schedule tasks depend on time in the algorithm. Some kind of situations are calculated in here such as, user can sleep in 01.00 AM or 23.00 AM.

```
#region CalculateScheduleList
//Schedule algorithm to create a plan using tasks
5 references
public List<ScheduleModel> CreateProgram(int userId) {
      this._userId = userId;
      List<ScheduleModel> schedule = new List<ScheduleModel>();
      List<Task> availableTasks = new List<Task>();
      List<SubTask> availableSubTasks = new List<SubTask>();

      //To get not done tasks
      availableTasks = GetTasks();
      //To get availableHours in a day.
      TimeSpan availableTimeDaily = AvailableTimeCalculator();

      //get availableTask one by one to get subTasks of availableTask
      foreach(var availableTask in availableTasks.Where(x => x.DueTime >= DateTime.Now)) {
            //To add priority if duetime of tasks is close.
            if(availableTask.DueTime == DateTime.Now)
                  availableTask.PriorityNumber = 1;
            else if(availableTask.DueTime == DateTime.Now.AddDays(1))
                  availableTask.PriorityNumber = 2;
            else if(availableTask.DueTime == DateTime.Now.AddDays(2))
                  availableTask.PriorityNumber = 3;
            taskDal.Update(availableTask);

            // If task is reminder, add scheduleList with it's due time. 5 minutes reminder.
            if(availableTask.IsReminder == true) {
                  schedule.Add(new ScheduleModel { Description = availableTask.Description, ScheduleTime = Convert.ToDateTime(availableTask.DueTime), SessionTime = new TimeSpan(0, 5, 0),
                     Status = "Reminder", Title = availableTask.Title, SubTaskId = 0 });
            }
            //If task is not reminder
            else {
                  //To add not completed subtasks to list
                  //var subTaskList = subTaskDal.GetAll(x => x.Status !=true && x.TaskId == availableTask.TaskId);
                  foreach(var subTask in availableTask.SubTasks) {
                        if(subTask.Status != true)
                              availableSubTasks.Add(subTask);
                        //availableSubTaskList.Add(new Tuple<SubTask, bool>(subTask, false));
                  }
            }
      }
```

Figure 3.31 The Rest Part of Schedule Algorithm

In the Figure 3.31, available tasks are calculated which is got by the status of task. Depending on status task, subtasks are got.

```
//To get now datetime to assign tasks
DateTime scheduleDateTime = DateTime.Now;
//to get hour and mins
TimeSpan timeInDay = new TimeSpan(0, 0, 0); // TO DO GET TIME IN DAY !!!!!!

TimeSpan availableTimeInADay = availableTimeDaily;
//assign subtasks to schedule one by one.
foreach(var item in availableSubTasks.OrderBy(x => x.Task.PriorityNumber)) {
    if(availableTimeInADay == new TimeSpan(0, 0, 0) && schedule.Count <= availableSubTasks.Count) {
        scheduleDateTime.AddDays(1);
        availableTimeInADay = availableTimeDaily;
    }

    //To get session time of subTask
    TimeSpan subTaskSessionTime = new TimeSpan(item.SessionTime.Value.Hours, item.SessionTime.Value.Minutes, item.SessionTime.Value.Seconds);
    // remove session time from available time in schedule day
    availableTimeInADay.Subtract(subTaskSessionTime);

    //assigning scheduleTime
    DateTime subTaskScheduleTime = new DateTime(scheduleDateTime.Year, scheduleDateTime.Month, scheduleDateTime.Day, timeInDay.Hours, timeInDay.Minutes, timeInDay.Seconds);

    schedule.Add(new ScheduleModel { Description = item.Task.Description, Title = item.Task.Title, ScheduleTime = subTaskScheduleTime, SessionTime = subTaskSessionTime, Status = "Waiting", SubTaskId = item.SubTaskId });

    // If task contains pomodoro then add schedule a pomodoro time
    if(item.Task.IsPomodoro == true) {
        // calculate pomodoro time which is 5 minutes.
        TimeSpan pomodoroTime = new TimeSpan(0, 5, 0);
        // to add pomodoro time and model to schedule list.
        schedule.Add(new ScheduleModel { Title = "Resting time", Description = "Resting time comes from pomodoro", ScheduleTime = subTaskScheduleTime.AddMinutes(25), SessionTime = pomodoroTime, Status = "Waiting", SubTaskId =
            0 });
        // remove session time from available time in schedule day
        availableTimeInADay.Subtract(pomodoroTime);
    }
}
return schedule;
```

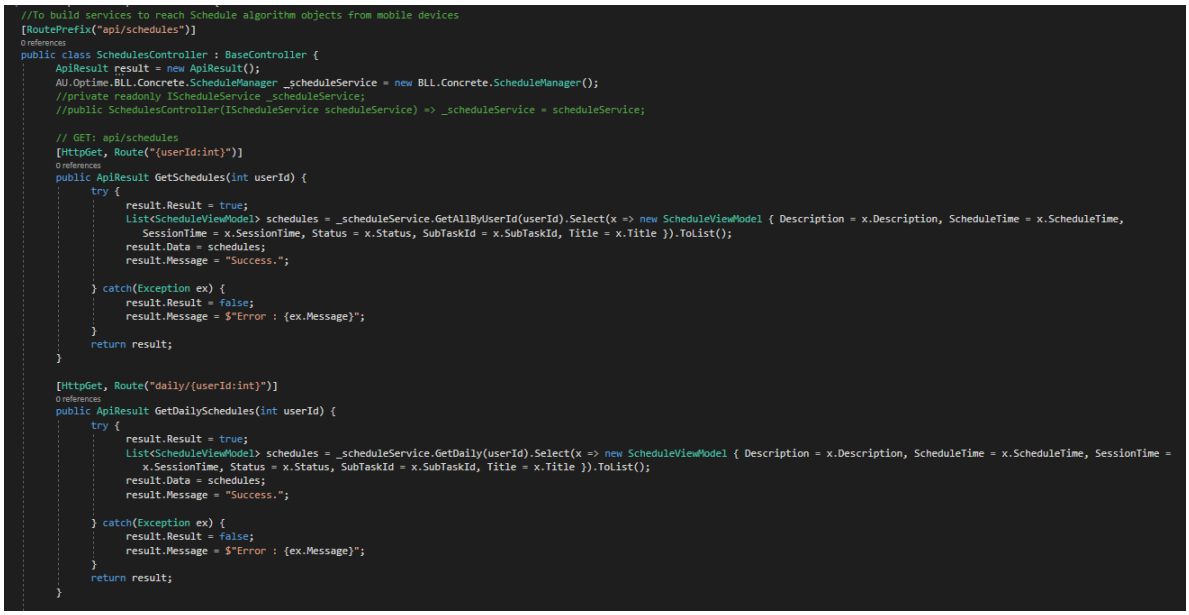
Figure 3.32 The Rest Part of Schedule Algorithm

In the Figure 3.32, schedule is created here with the last part of the algorithm.

```
//Specify operations using Scheduler algorithm
2 references
public class ScheduleManager : IScheduleService {
    Scheduler _scheduler = new Scheduler();

    //to get all schedule by user
    2 references
    public List<ScheduleModel> GetAllByUserId(int? userId) {
        if(userId is null) {
            throw new ArgumentNullException(nameof(userId));
        }

        return _scheduler.CreateProgram(Convert.ToInt32(userId));
    }

    //to get daily schedule list depend on user
    2 references
    public List<ScheduleModel> GetDaily(int? userId) {
        if(userId is null) {
            throw new ArgumentNullException(nameof(userId));
        }

        return _scheduler.CreateProgram(Convert.ToInt32(userId)).Where(x => x.ScheduleTime.Day == DateTime.Now.Day).ToList();
    }

    //to get monthly schedule list depend on user
    2 references
    public List<ScheduleModel> GetMonthly(int? userId) {
        if(userId is null) {
            throw new ArgumentNullException(nameof(userId));
        }
        return _scheduler.CreateProgram(Convert.ToInt32(userId)).Where(x => x.ScheduleTime.Month == DateTime.Now.Month).ToList();
    }

    //to get weekly schedule list depend on user
    2 references
    public List<ScheduleModel> GetWeekly(int? userId) {
        if(userId is null) {
            throw new ArgumentNullException(nameof(userId));
        }
        return _scheduler.CreateProgram(Convert.ToInt32(userId)).Where(x => x.ScheduleTime <= DateTime.Now.AddDays(7)).ToList();

    }

    //to delay tasks from schedule list
    1 reference
    public void UpdateScheduleModel(ScheduleModel model) {

        _scheduler.CreateProgram(Convert.ToInt32(model.SubTaskId));
    }

}
```

Figure 3.33 ScheduleManager

ScheduleManager class is generated to specify operations of schedule algorithm to get in different layers such as Web services send or get data from mobile layer and database layer.



```
//To build services to reach Schedule algorithm objects from mobile devices
[RoutePrefix("api/schedules")]
0 references
public class SchedulesController : BaseController {
    ApiResult result = new ApiResult();
    AU.Optime.BLL.Concrete.ScheduleManager _scheduleService = new BLL.Concrete.ScheduleManager();
    //private readonly IScheduleService _scheduleService;
    //public SchedulesController(IScheduleService scheduleService) => _scheduleService = scheduleService;

    // GET: api/schedules
    [HttpGet, Route("{userId:int}")]
    0 references
    public ApiResult GetSchedules(int userId) {
        try {
            result.Result = true;
            List<ScheduleViewModel> schedules = _scheduleService.GetAllByUserId(userId).Select(x => new ScheduleViewModel { Description = x.Description, ScheduleTime = x.ScheduleTime,
              SessionTime = x.SessionTime, Status = x.Status, SubTaskId = x.SubTaskId, Title = x.Title }).ToList();
            result.Data = schedules;
            result.Message = "Success.";

        } catch(Exception ex) {
            result.Result = false;
            result.Message = $"Error : {ex.Message}";
        }
        return result;
    }

    [HttpGet, Route("daily/{userId:int}")]
    0 references
    public ApiResult GetDailySchedules(int userId) {
        try {
            result.Result = true;
            List<ScheduleViewModel> schedules = _scheduleService.GetDaily(userId).Select(x => new ScheduleViewModel { Description = x.Description, ScheduleTime = x.ScheduleTime, SessionTime =
              x.SessionTime, Status = x.Status, SubTaskId = x.SubTaskId, Title = x.Title }).ToList();
            result.Data = schedules;
            result.Message = "Success.";

        } catch(Exception ex) {
            result.Result = false;
            result.Message = $"Error : {ex.Message}";
        }
        return result;
    }
```

Figure 3.34 ScheduleController

Schedule controller in the Figure 3.34 is created to build services to reach Schedule algorithm objects from mobile devices.

Postscript: Other code statements are defined in the project that is shared as file.

9. TESTS

1. Problem

Showing time on the screen may cause an error to depend on mobile phone language. The program developed using the phone that language is English. If the application is tested on the phone which's language is Turkish, there will be an error about showing date time of properties.

Solution

Application language can be changed by the current language of the system. CultureInfo can be got with a code shown below. The default language can be retrieved using the InstalledUICulture.

```
System.Globalization.CultureInfo.DefaultThreadCurrentCulture=CultureInfo.InstalledUICulture;
```

2. Problem

It is not possible to work with localhost in Android devices. It gets an error while connected WebApi to the Mobile layer. Generated error is "System.Net.WebException: 'Failed to connect to localhost'".

Solution

To solve this problem, Web Api can be published in Azure or other services. The URL which is defined by the developer will be used in the project to connect between Web API and Mobile layers.

3. Problem

The web services published in the server and if SSL certificate does not exist in the server, there will be a security problem on some operating systems in mobile. Error name is "Cleartext HTTP traffic not permitted."

Solution

1. First "https://" will be used instead of "http://"
2. File called "res/xml/network_security_config.xml" can be created

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config cleartextTrafficPermitted="true">
        <domain includeSubdomains="true">api.example.com(to be adjusted)</domain>
    </domain-config>
</network-security-config>
```

and AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        ...
        android:networkSecurityConfig="@xml/network_security_config"
        ...>
        ...
    </application>
</manifest>
```

3. Configurate the AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        ...
        android:usesCleartextTraffic="true"
        ...>
        ...
    </application>
</manifest>
```

4. Problem

While photo which is taken by mobile phone or data which has big size is transferring to Web API. An error occurs because of data which has big size cannot be transferred using RouteValue or Header.

Solution

The data will be transfer using Body because of security problems and service receive it using FromBody.

5. Problem

Using HTML predefined character entities such as “&” in the string gives error.

Solution

Instead of using these characters, there are some rules for them. For example, using “&” gives developer to write & character. Those characters can be found in the http://www.madore.org/~david/computers/unicode/htmlent.html

6. Problem

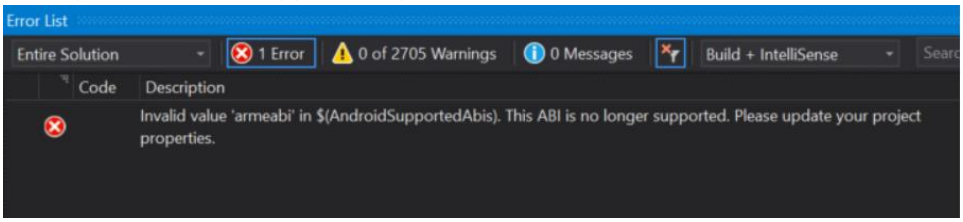
Using List View and Scroll View at the same time, there are some spaces in the design page. Because List View contains its own Scroll View.

Solution

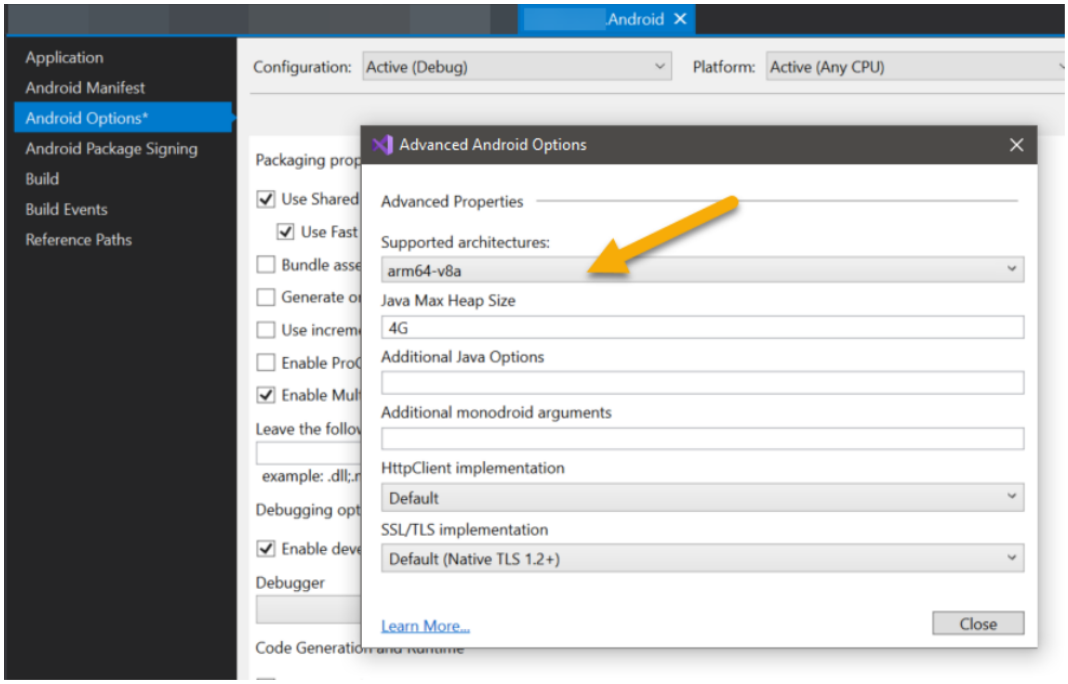
Using only List View is the solution for this problem.

7. Problem

Some smart phones do not support x86 processor architecture.



Solution



Android Options will be selected including system architecture as shown in the Figure above.

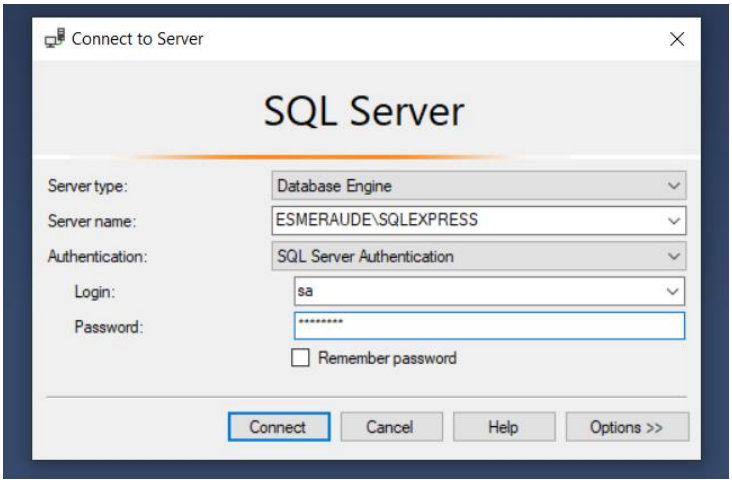
8. Problem

An error occurred while testing getting data for tasks page in the application through Web API. The error message is:

{"Result":false,"Data":null,"Message":"Error : An exception occurred while initializing the database. See the InnerException for details."}

Solution

Sql Server Authentication property module need to be selected as Sql Server Authentication and it needs a username with password as shown in the below.



9. Problem

In some pages of the mobile application gives error with an error message as, “Metadata file .dll could not be found.”

Solution

Some packages appear like not installed on the application because of version conflicts between other packages. The solution is to uninstall the package and reinstalling it solve the problem. In this project the package that is named ‘Rg.Plugins.Popup’ gives the error and it was uninstalled and then reinstalled to solve problem.

10. Problem

In the Web API testing, application throw exception with a message “The ObjectContext instance has been disposed and can no longer be used for operations that require a connection.”

Solution

In the project Entity Framework uses and it has lazy loading. In the project relational tables used for some queries. To solve this problem, it is needed to add new method to get this kind of relational tables’ result. These methods in the project are shown below.

```
2 references
public Task GetTaskwithDetails(int id) {
    using(var context = new OptimeContext()) {
        return context.Tasks.Include("TaskDetail").Where(x => x.TaskId == id).FirstOrDefault();
    }
}

2 references
public List<Task> GetTaskwithDetailsByUser(int id) {
    using(var context = new OptimeContext()) {
        return context.Tasks.Include("TaskDetail").Where(x => x.UserId == id && x.ProjectId == null && x.BoardId == null).ToList();
    }
}

1 reference
public List<Task> GetTaskwithFullDetailsByUser(int id) {
    using(var context = new OptimeContext()) {
        return context.Tasks.Include("SubTasks").Include("TaskDetail").Where(x => x.UserId == id && x.ProjectId == null && x.BoardId == null).ToList();
    }
}
```

With .Include() query, relational table name is used to get data inside of the table.

11. Problem

"Sequence contains more than one element" is shown when getting a result in the task detail page.

Solution

If sequence contains more than one element, the exception is thrown. Using FirstOrDefault() method solved the error.

```
2 references
public Task GetTaskwithDetails(int id) {
    using(var context = new OptimeContext()) {
        return context.Tasks.Include("TaskDetail").Where(x => x.TaskId == id).FirstOrDefault();
    }
}
```

12. Problem

StringFormat for binding TimeSpan in XAML does not work to show timespan in tasklistpage.

Solution

To solve this problem StringFormat was used as shown below.

```
<Label Text="{Binding SessionTime, StringFormat='{}{0:hh\\:mm}'}"
        HorizontalOptions="CenterAndExpand" />
```

REFERENCES

1. https://trello.com/
2. https://app.asana.com/
3. https://www.wunderlist.com/
4. https://keep.google.com/
5. https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin
6. https://dribbble.com/search/management%2C%20project%2C%20team
7. https://www.pinterest.com/search/pins/?q=project%20management&rs=typed
8. http://senior.ceng.metu.edu.tr/2019/mainpage/
9. http://www.cs.bilkent.edu.tr/~sekreter/CS491-2/Projects1920.htm
10. http://www.madore.org/~david/computers/unicode/htmlent.html
11. https://docs.microsoft.com/en-us/xamarin/xamarin-forms/user-interface/controls/
12. https://forums.xamarin.com/discussion/146174/vs2019-error-this-abi-is-no-longer-supported
13. https://stackoverflow.com/questions/45940861/android-8-cleartext-http-traffic-not-permitted