

Universiteti për Biznes dhe Teknologji Departamenti I Shkencave Kompjuterike dhe Inxhinieri

Punoi: Arton Ramadani, Dren Nikqi

ADAPTIVE ACADEMIC DOCUMENT SYSTEM

Mentori: Edmond Jahjaga, Arber Kadriu

Dhjetor/2020

Ky punim është përpiluar dhe dorëzuar në përmbushjen e kërkesave të pjesshme për Shkallën Bachelor

# Product Overview

## Introduction

**E-Ditari**

To prevent errors caused by information systems, a platform to store complete and accurate Student’s information in a **user-friendly format** is needed. As we are aware of many miscommunications caused by the form that already is being used in schools, which is to write down the progress, consistency and grades in a hard paper. We come with a solution to this specific problem, an application which will make it easier to access data for students and for teachers as well. Basically you can log in from a computer, laptop , tablet and even from a smartphone.

This platform is programed to be used from different users, starting from students, teachers, and from the principal of the School. Every user will have its limited access.

## Product Vision and Project Scope

**General requirements**

1. Because of sensitivity of data and because of having all personal information, need to have a Non-Disclosure-agreement (NDA).
2. Need to track the process every time.
3. Weekly report of process of work in written form, also tracing on daily processes and works via Waterfall.
4. While we will have different cities with different schools to provide with support and information the DB should be designed to work with distance radius based starting from schools nearby to those which are to other cities.
5. Responsive 100% to tablets, personal computers.
6. Easy loadable.
7. Different access levels in database:
   1. Read only data
   2. Read, write data on database but cannot change
   3. Read, write, export and generate report.
   4. Super admin, change, execute and delete, FULL ACCESS.
8. Database authentication and protection
9. Structure, programming, Database
   1. N-Tier Structure,
   2. Database: SQL,
   3. Programing language: Java.
10. Instant search: The DB should work and display data from db instantly as you are typing the name of what you are searching for.
11. Log form for authenticated users
12. Database encryption and data stored encryption
13. DB and data should be protected by SSL encryption
14. Database optimization should contain these points as below:
    1. Optimizing SQL Statements
    2. Optimization and Indexes
    3. Optimizing Database Structure
    4. Optimizing for InnoDB Tables
    5. Optimizing for MEMORY Tables
    6. Buffering and Caching
    7. Optimizing Locking Operations
15. User data to be following EU GDPR

## Target Markets

The system is specifically dedicated to primary, lower and secondary schools as an education institutes, regardless of whether they are a public or private schools.

## Stakeholders

**Ministry of Education** – to easly generate reports using our platform.

**Schools-** As an education institute can use this platform to save and generate data for students and staff.

**Professors/ Teachers**- are the stakeholders as they will be the ones who will work the most with our platform.

**Student** – Follow their progress, maintainance and real time notifications made by Teachers.

**Parents**- follow up the progress and maintainance of their children.

**Developers-** will maintain the platform.

## Assumptions and Constraints

### Assumtions:

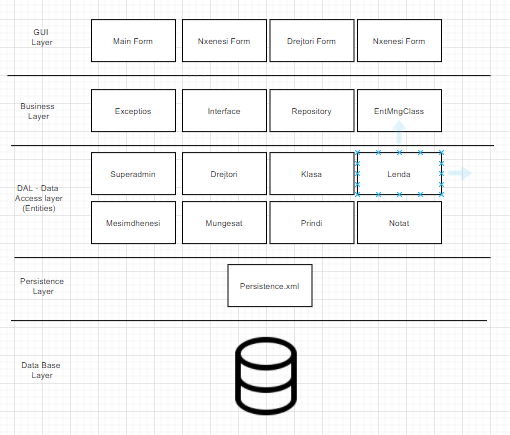
E-ditari test phase will be ready by the end of this semester. Creating the super admin user account and then super admin user will be able to create and edit the other accounts as necessary.

### Constrains:

Log in to E-ditari will be done through a browser, using an account which is created by super admin. To generate reports or to edit data, depending of the type of the account you are provided with.

1. **Architectural Models**

E-ditari is a Layered system, as shown inthe layer diagram in Figure B-1



*Figure B-10-1 E-ditari Layers Overview*

## Layer Responsibilities

|  |  |
| --- | --- |
| Layer | Responsibilities |
| GUI Layer | The Grafical User Interface, is the layer that the CRUD forms are programed, this layer is responsible for the appearance of the program. |
| Business Logic Layer | The BLL is responsible to manage the comunication betwen the user interface and the Database. |
| Data access layer | DAL is responsible to have the source code which it can also be described as the brain layer to better understand the structure of this program. |
| Persistence Layer | It’s used tomake the connection with the SQL Database and the souce code of the project. |
| Database Layer | The database layer is responsible to stores the data in the database. |

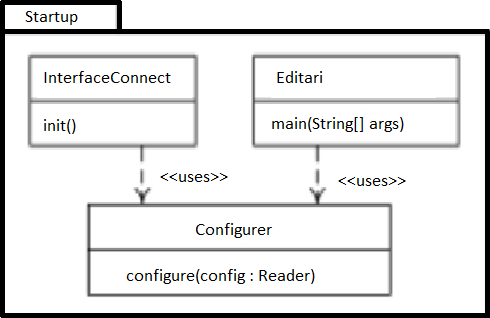
*Table B-10-2 E-ditari Layer Responsibilities*

## GUI Layer Decomposition

This layer is responsible for the main part of the program. Startup layers role is to create and connect all runtime components and to restore the system state form persistent store.

The layer will be the same across all devices and will communicate with the User Interface layer to provide a simple and easy-to-use Graphical User Interface.

The Startup layer contains the main program class, which may be either an application or an applet, assuming the program is written in Java. The main program uses a Configurer, which sets up the program based on the available hardware (real or simulated) and whether it is a simulation or a fielded program, and restores the previous program state. The Configurer reads a configuration specification to determine what sort of hardware (or simulated hardware) the program must use. It reads a persistent store to reset the program state. Figure B-10-3 shows the decomposition of the Startup layer.



*Figure B-10-3 Startup Layer Structure*

## Startup Layer Module Responsibilities

|  |  |
| --- | --- |
| Module | Responsibilities |
| InterfaceConnect | Provide a connection from this class through the Configurer to  call the Graphical User Interface from User Interface layer |
| Editari | Createa Configurer that connects the User Interface layer with the  Startup layer. |
| Configurer | Create and connect runtime components based on configuration specifications and whether the program is a simulation or a fielded  product. Restore the state of the system from persistent store. |

*Table B-10-4 Startup Layer Module Responsibilities*

## Startup Layer Interface Specifications

Services Provided

|  |  |  |
| --- | --- | --- |
| InterfaceConnect initialization | *Syntax:* | Init() |
| *Pre:* | None. |
| *Post:* | The service that calls the GUI has started. |
| Program execution | *Syntax:* | main( args : String ) |
| *Pre:* | None. |
| *Post:* | The main program is set up according to its local configuration and its state is reset to what it was when it  was last executing. |

*Table B-10-5 Startup Layer Interface Specifications*

UserInterface.Create() -Create the user interface.

Userlnterface setDisplay() -Configure the user interface with the virtual display.

## Startup Layer Design Rationale

To minimize the use of repetitive code the configure class should be but in a separate class rather than in main or InterfaceConnect class. That also makes it easier to make changes in the configurer class and also in the other two classes.

## Database Layer Decomposition

The Database Layer contains components whose role is to store and manage data in the database. This layer contains classes like: StafiShkollor, Mesuesi, Personi, Nxenesi, Ditari, Lendet.

## Database Layer Module Responsibilities

|  |  |
| --- | --- |
| Module | Responsibilities |
| Personi | Create or modify a person on the database and then assign the role. |
| StafiShkollor | Create or modify a user on the database from stafi shkollor. |
| Mesuesi | Create or modify a Mesuesi type of person on the database. |
| Nexenesi | Create or modify a Nxenesi type of person on the database. |
| Lendet | Add grades and look at the grades based on each student and subject. |
| Nxenesi | Create, modify and assign students their grades. |

*Table B-10-7 Simulation Layer Module Responsibilities*

## Database Layer Interface Specifications

|  |  |  |
| --- | --- | --- |
| Create the person | *Syntax:* | setPersoni() |
| *Pre:* | None. |
| *Post:* | Personi entity is created. |
| Get all persons | *Syntax:* | getPersonat() : Personi[] |
| *Pre:* | None. |
| *Post:* | An array containing all persons is returned. |
| Get a  specific person | *Syntax:* | getPersoni( ID : int ) : Personi |
| *Pre:* | ID is not null. |
| *Post:* | A specific person searched by ID is returned. |

1. **Mapping Between Models**

Table 3.1 indicates the uses between layers and their constituents. The leyers in the row use the classes in the columns.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Admin Service** | **User Service** | **BLL** | **DAL** |
| **GUI** | LogIn Controller | LogInController |  |  |
| **BLL** |  |  | Fetching data. |  |
| **DAL** |  |  | Routing |  |
| **Persistence** |  |  |  | DB. |

The runtime components shown in Fig. 3.1 are comprised of the modules and the data stores shown in table 3.2

|  |  |
| --- | --- |
| **GUI** | **Modules of the GUI** |
| Model | Includes files that form the structure of the program. |
| View | Are the crud forms that make the program usable to the users. |

# Architectural Design Rationale

In our program we have used the N-Tier architecture design. One of the features of the layered architetures is that it is very flexible. Flexibility is a key feature in our program and with this architecture we can make changes in the Bussines layer very easily without intervening in the User Interface layer.

Another feature that we think it’s really important in our program is Simplicity. On the User Interface layer, we can provide a clean and simple GUI which will be coded into just one version for devices.

The Persistence layer is also very important for this program. This layer contains all the classes that are responsible for technical stuff, like persisting the data in the database, handling the functions, and many other things.

And lastly comes the Database layer where all the data is stored. Data in our program is very crucial because the whole point of this program is to store and manage data.

So, the layered architecture is necessary for our program because by isolating the layers, they become independent from one another. Thus if, for example, we want to change the database, this will cause a big impact on the database layer but that won’t impact any other layers.

Being that this is not a one person project, layered architecture is a bonus because team members can work on a layer that they are more familiar with.