CENG491
INITIAL DESIGN REPORT

MISS PROJECT

SOFTRUNNER

Zerrin Bozel 1297571
Burcu Ardıç 1297472
Pınar Çelikoğlu 1297613

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Goals and Objectives

We aim to build an attractive online multi-user Project Management Tool *MissProject* which is easy to use for every kind of users from different backgrounds and important of all, a totally reliable system in which the companies can trust.

With *MissProject*, the quality in project management will increase, and the time, cost and resource management will be optimum. Our goal is creating a satisfaction chain which includes every-one who are somehow related with planning, students, managers, teachers, even housewives, etc...

We claim that this system will increase the number of people using this kind of tools, because as well as providing a practical way of managing projects, our system contains the automatic e-mails that allow the user to keep track of the project without signing in. This way, a registered user can view the project without loosing time.

Finally, we are aiming this project to be used by several people from different backgrounds, not just by the companies. So our GUI will be user-friendly.

1.2 Statement of Scope

MissProject is a powerful, flexible online project management tool that can be used to control simple or complex projects.

A user signs up to *MissProject*, and becomes the Configuration Manager for his/her company. He/she is responsible from removing and adding the projects to the company account, and assigning a Project Manager for each project.

The project planning phase of projects usually spans a significant length of time and involves many people. During this phase it's important to for the Project Manager to define the objectives, assumptions, and constraints of the project. After initial planning, the Project Manager can create the project file. He/she enters the preliminary project data, and attaches the planning documents to the file.

Once the objectives of the project have been established, the actual product or service that meets those objectives can be defined. After the work involved in the project has been determined, the Project Manager can organize the project into milestones, phases, and tasks and enter it into a *MissProject* file. If this data is stored in another file, it can be copied or imported into *MissProject*.

The user can organize the project using a master project and subproject files when he/she needs to manage one large, complex project or multiple related projects.

By entering durations for tasks rather than desired start or finish dates, the PM allows *MissProject* to create a schedule for him/her. After task durations are entered, the tasks are related to each other where necessary, and are assigned to specific dates. Task dependencies can be created between tasks in different projects. Creating dependencies between projects models the interrelationships between different projects and helps keep the project up to date.

At this point in the project planning process, the project scope has been identified; the task list and estimated task durations have been set up, all resources have been identified, approved, and procured. This information can be used to make preliminary estimates, identify requirements, and start the staffing and procurement processes to acquire the resources that will carry out the project tasks. The Project Manager identifies his/her team members and adds users to the project by creating new accounts for non-registered users or selecting them from the list of current system users as Standard Users or Administrators. The Standard Users can not make any modifications in the project; they can only view the information about tasks, resources, groups; generate reports, etc. The administrators have the right to make modifications in the project, such as creating/removing tasks, assigning resources to the tasks, drawing the Gantt chart, etc.

At this point the Project Manager also decides what material resources he/she while working on a project. Sharing resources is useful for managing resource information and assignments across multiple projects in which the same groups or material resources will be used. The resource information that has been entered into the system by the Configuration Manager can be assigned to the specific tasks, which have been set up as the work of the project by the Project Manager or Administrators.

Cost estimating is the process of developing the approximate resource and/or task costs needed to complete the project activities. When all costs have been entered, the PM or Administrators may want to save them as their budget before they start tracking and managing the plan. At this point they may want to include important notes about budget decisions, share the budget information with others, or transfer it to other programs, such as a financial system that their company may be using. After establishing costs, the necessary preparations can be made for tracking and managing them to ensure the project stays within budget. The PM or Administrator can specify a fiscal year start date, control calculation options, and determine when costs should be payable.

Before a project begins, the quality standards should be identified by the Project Manager to meet all of the project requirements. After the quality requirements are identified, the Project Manager or Administrators can adjust scope, resources, and schedule as necessary to achieve the desired quality.

To optimize the project plan to meet the finish date, scheduled finish date should be reviewed after building the project plan. Reviewing the allocation of the resources and the planned costs also optimize the project for resources and to meet the budget.

After the project has been scheduled, a notification is sent to the registered project members by e-mail.

If the PM or Administrators have changed tasks, resources, or assignments, the most current project information is distributed to the project members via e-mail.

The most effective way of managing resources on a project is to balance their workloads and track progress on tasks. By reviewing such resource information as assignments, overallocations or underallocations, resource costs, and variances between planned and actual work, it can be verified that resources are assigned to tasks to get the results that the PM wants. To get the best performance and results from resources, the users need to manage their workloads to handle overallocations and underallocations. If the resource assignments are changed, the effects of the changes on the overall schedule should be checked to be sure the results will meet the project goals.

To keep costs within budget, the cost problems can be identified by reviewing cost totals and cost variances that occur over time so adjustments can be made by PM or Administrators where needed.

The PM can increase scope or cut scope from the project.

If new risks are identified that have arisen during the project, the users will need to respond to those risks. Most likely, they will have to deal with risks that threaten to delay task, phase, or project end dates; increase the budget; overwhelm resources; or all three.

As the project progresses, its current status can be reported as notification mails to all of the registered team members.

Also even after completion of the project, Miss Project will provide companies to benefit from historical data.

1.3 Success Criteria of the Project

Among the success criteria of *MissProject*, security is the first to be mentioned since the information about the projects should not be accessed by rival companies. One of the critical points in online project management is that: The roles and permissions of the user should be identified distinctly for the consistency of the project. The system should employ a well-tested authentication and authorization mechanism.

When all security issues are handled, the response time, ease of use and the attractive interface will move us one step further. To be able to be used by every kind of people from different backgrounds, the system should be clear and intuitive to use. It will not be necessary for the users to have any additional knowledge or training to use *MissProject*. But in case of any problem, we will have a strong *help* facility. Most web-based applications provide users a complicated system, which repel people who have already doubts about internet&security. Our system will consist of very systematic steps, each of them are clear and easy to use, erasing all the doubts in people's minds.

1.4 Design Constraints, Limitations

1.4.1 Design Constraints

System: *MissProject* should provide a secure environment for managing courses. Multiple users should be able to use the system simultaneously; the response time should be reasonably short.

Interface: The end users will use ordinary browsers to login and use the system, therefore the human-system interface should be supported by as many browsers as possible, even the text browsers. Needless to say that the most important constraint about the interface is that it must be user friendly.

1.4.2 Limitations

Time: The time seems to be an important constraint of our project since it must be completed in 9 months and all of the group members have different courses. It is so important to track the project schedule properly.

Employee Skills: The programming skills of the employees are other restrictions. In spite of the fact that it is not so effective as time, it certainly could limit us from doing some additions to the project during its lifetime.

Hardware: The PC's to be used during the implementation of *Miss Project* will have these features minimum: 733MHz Pentium CPU, 128 MB RAM, 32 MB Graphics Card and 15` monitors.

Portability: The software will be implemented under Windows XP and can be executed in Widows 98 and later versions.

Programming Language: We have decided to use .NET as our development platform. During Analysis phase, we were searching about C# and Java, but we could not decide with which one of them we should implement our system. In order to be able to use .NET technologies, we concluded that C# is better. Some of the reasons of our decision for using C# .NET are as follows:

- Staff's being more experienced in C# rather than Java

- The power and flexibility offered by the advanced features and functionality of ASP.NET is much better then anything on the Java side of the world.

- C# took many of the ideas and concepts of Java and made them better.

- Provides a measure of security

- Doesn't require special permission (e.g. unlike CGI)

- Has better interactivity

- Has SQL database access

- Is easier to use than C++, and above all: it is WEB-oriented, and provides an ever growing wealth of class libraries for ever new application domains.

1.4.3 Scheduling Constraint

gantt_chart.bmp (Gantt chart is given as appendix in hardcopy.)

2. DATA DESIGN

2.1 Data Description

The information of Individuals, Groups, Tasks, Material Resources, Projects, Companies and roles and their relations are kept and updated when necessary in database.

2.2 Data Objects and Complete Data Model

ER_diagram.vsd (Revised ER diagrams are given as appendix in hardcopy.)

2.3 Data Dictionary

1. Individuals Table

- The information of the individuals can be changed through update process.
- Attributes of the Individual table are as follows;

iid	: Integer
username	: String[10]
password	: String[10]
name	: String[40]
company_id	: Integer
Mail_adress	: String[30]

UNIQUE(username)

PRIMARY KEY: iid

FOREIGN KEY: company_id REFERENCES Company(cid)

2. Group Table

- Attributes of the Group table are as follows;

gid	: Integer
name	: String[10]
status	: Integer
proffesion	:String[20]
team_leader	:Integer

PRIMARY KEY: gid

FOREIGN KEY: team_leader REFERENCES Individuals(iid)

3. member Table

- member table contains the relation between the individuals and the groups.
- Attributes of the member table are as follows;

gid : Integer

iid :Integer

PRIMARY KEY: (gid, iid)

FOREIGN KEY: gid REFERENCES Group(gid)

FOREIGN KEY: iid REFERENCES Individuals(iid)

4. Task Table

- The information of the tasks can be changed through update process.
- Attributes of the Task table are as follows;

tid : Integer

name : String[30]

project_id : String[30]

to : Date[DD/MM/YYYY]

from : Date[DD/MM/YYYY]

duration : String[10]

priority : Integer

prerequisite_id : Integer

UNIQUE(name, project_id)

PRIMARY KEY: gid

FOREIGN KEY: project_id REFERENCES Project(pid)

FOREIGN KEY: prerequisite_id REFERENCES Task(tid)

5. Assignedto Table

- Assignedto table shows the relation between the tasks and the groups.

- Attributes of the Assignedto table are as follows;

gid : Integer
tid : Integer

PRIMARY KEY: (gid, tid)
FOREIGN KEY: gid REFERENCES Group(gid)
FOREIGN KEY: tid REFERENCES Task(tid)

6. Material_resources Table

- Parts for the Material_resources table are as follows;

mid : Integer
description : String[30]
type : String[20]
serial_number : String[20]
PRIMARY KEY: (mid)

7. Reserve Table

- Reserve Relation shows the relation between the Tasks and the Material Resources.
- Parts for the Reserve Relation are as follows;

tid : Integer
mid : Integer
from : Date[DD/MM/YYYY]
to : Date[DD/MM/YYYY]

8. Project Table

- Attributes of the Project Table are as follows;

pid : Integer
name : String[20]
company_id : Integer

to : Date[DD/MM/YYYY]
from : Date[DD/MM/YYYY]
duration : String[10]
project_manager_id : Integer

PRIMARY KEY: (pid)
FOREIGN KEY: company_id REFERENCES Company(company_id)
FOREIGN KEY: project_manager_id REFERENCES Individuals(iid)

9. Company Information

- Beginner ID is the Individual ID of the manager that first sign up to the system and opens the account of the company.
- Parts for the Company information are as follows;

cid : Integer
name : String[20]
Configuration_manager_id : Integer

PRIMARY KEY: (cid)
FOREIGN KEY: Configuration_manager_id REFERENCES Individuals(iid)

10. Role Table

- Attributes of the Role table are as follows;

individual_id : Integer
project_id : Integer
role : String[25]
PRIMARY KEY: (individual_id, project_id)
FOREIGN KEY: individual_id REFERENCES Individuals(iid)
FOREIGN KEY: project_id REFERENCES Project(pid)

3. ARCHITECHTURAL DESIGN

3.1 Review of Data Flow Diagrams

DFD.vsd (Revised Data Flow Diagrams are given as appendix in hardcopy.)

3.2 Static View of Design

We examined the design patterns in C# and decided to use the Abstract Factory method. Below is the class diagram of *Miss Project*. The classes and the methods to be implemented are shown in this graph.

ClassDiagram.vsd (Class Diagram is given as appendix in hardcopy.)

3.2.1 Classes in Class Diagram

3.2.1.1 Individual Class

Definition:

The instances of this class contain the information about all users.

Responsibilities:

- Creating an object with the inputs and commands taken from the user during signing up, adding user/Administrator, Update User Information processes.

Operations:

- Individual(…): Arguments that are taken from the user.
  Creates an Individual object from the commands of the user.

3.2.1.2 Project Class

Definition:

The instances of this class contain the information about projects.

Responsibilities:

- Creating an object by the input and commands taken from the Configuration Manager during adding a project.

Operations:

- Project(…): Arguments that are taken from the Configuration Manager.
  Creates a Project object from the commands of the Configuration Manager.

.

3.2.1.3 Company Class

Definition:

The instances of this class contain the information about companies

Responsibilities:

- Creating an object with the input and commands taken from the Configuration Manager during signing-up

Operations:

- Company(…): Arguments that are taken from the Configuration Manager.
  Creates a Company object from the commands of the Configuration Manager.

3.2.1.4 Resources Class

Definition:

The instances of this class contain the information about resources of the company and projects.

Responsibilities:

- Creating an object by the input and commands taken from the Configuration Manager during adding resources or updating resources.

Operations:

- Resource(…): Arguments that are taken from the Configuration Manager.

Creates a Resource object from the commands of the user.

3.2.1.5 IDBConnector Interface

Definition:

The methods of interface is used to connect to any type of database and make transactions according to the users' commands. In *MissProject* SQL will be used, so the SqlDBConnector implemented inherits these methods.

Responsibilities:

- Opening a connection to the database.
- Beginning, committing and rolling back transactions.
- Executing the queries.
- Closing the connection to the database.

Operations:

- void Open()

  Opens the connection
- void Close()

  Closes the connection
- void BeginTransaction()

  Starts the transaction.
- void CommitTransaction()

  This operation is used to commit transactions.
- void RollbackTransaction()

  This operation is used to rollback transactions.
- DataSet ExecuteDataSet(string commandText)
- void ExecuteNonQuery(string commandText)
- IDataReader ExecuteReader(string commandText)
- object ExecuteScalar(string commandText)

  These operations are used to execute the queries.

3.2.1.5 SqlDBConnector Class

Definition:

This class inherits the methods in the IDBConnector interface and is used to connect to the SQL Database and make transactions according to the users' commands.

Responsibilities:

- Opening a connection to the database.
- Beginning and committing and rolling back transactions.
- Executing the queries.
- Closing the connection to the database.

Operations:

- void Open()
  Opens the connection
- void Close()
  Closes the connection
- void BeginTransaction()
  Starts the transaction.
- void CommitTransaction()
  This operation is used to commit transactions.
- void RollbackTransaction()
  This operation is used to rollback transactions.
- DataSet ExecuteDataSet(string commandText)
- void ExecuteNonQuery(string commandText)
- IDataReader ExecuteReader(string commandText)
- object ExecuteScalar(string commandText)
  These operations are used to execute the queries.
- SqlDBConnector(string SqlServerName, string InitialCatalog, string UserID, string Password)
  This is the constructor of the class, and has the parameters to construct a connection string.

3.2.1.7 DBPresenter Class

Definition:

This class contains the information about all of the processes of

Responsibilities:

- In the most abstract level, the responsibility of this class is accomplishing all of the processes of the system correctly.
- Managing the necessary updates in the database.

Operations:

- void SignUp(…): Arguments taken from the first user.
  Creates the company and Configuration Manager account and enters the information into the database.
- void UpdateUserInfo(…): Arguments taken from the user.
  Takes the information that the user enters, and updates it from the database.
- void AddStUser(int IndividualId)
  A standard user is added to the current project.
- void RemoveStUser(int IndividualId)
  A standard user is removed from the current project.
- void AddAdmin(int IndividualId)
  An Administrator is added to the current project.
- void RemoveAdmin(int IndividualId)
  An Administrator is removed the current project.
- void AddProject(….): Arguments taken from the user.
  The user adds a project to database with the information he/she enters.
- void RemoveProject(int ProjectId)
  The user removes a project from the database.
- void AddTask(…):Arguments taken from the user.
  A task is added to the project.
- void RemoveTask(int TaskId)
  A task is removed from the project.
- void AddResource(…..):Arguments taken from the user.
  A resource is added to the company account with the information that the user enters.
- void RemoveResource(int ResourceId)
  A resource is added to the company account.
- void UpdateResource(….): Arguments taken from the user.

The information of a resource of the company account is updated with the information that the user enters.

- void AddGroup(…) : Arguments taken from the user.
  A group is added to the project.
- void RemoveGroup(int GroupId)
  A group is removed from the project.
- void UpdateGroup(…) : Arguments taken from the user.
  Takes the information that the user enters and updates the information about groups.
- void DrawGanttChart(…): Arguments taken from the user.
  Takes the information that the user enters and generates the Gantt chart.
- void DrawResourceGraph(…): Arguments taken from the user.
  Takes the information that the user enters and generates the ResourceGraph.
- void DrawNetworkDiagram(…): Arguments taken from the user.
  Takes the information that the user enters and generates the NetworkDiagram.
- void GenerateReport(….): Arguments taken from the user.
  Takes the information that the user enters and generates a report.
- void Filter(…): Arguments taken from the user.
  Takes the information that the user enters makes the project to be viewed according to the user's request.

3.2.1.8 Controller Class

Definition:

This class contains all the functions of the system.

Responsibilities:

- Controlling the permissions of the users to generate the requests.
- Using the functions of DBPresenter class accomplish the processes.

Operations:

- bool Authenticate(string username, string password)
  Checks the validity of the username and password from the database and makes the user log in to the system if both are valid.

- bool AdminAuthorize(int IndividualId, int ProjectId)

  Checks if the user logged in is an administrator of the project.
- bool PMAuthorize(int IndividualId, int ProjectId)

  Checks if the user logged in is the Project Manager of the project.
- bool CMAuthorize(int IndividualId, int ProjectId)

  Checks if the user logged in is the Configuration Manager of the company.
- bool StUserAuthorize(int IndividualId, int ProjectId)

  Checks if the user logged in has the right to access the project.
- void SignUp(…): Arguments taken from the first user.

  Calls SignUp function of DBPresenter Class.
- void SignOut()

  Closes the current user's session.
- void UpdateUserInfo(…): Arguments taken from the user.

  After checking the Authentication, calls UpdateUserInfo function of DBPresenter Class.
- void AddStUser(int IndividualId)

  After checking the PMAuthorize , calls AddStUser function of DBPresenter Class .
- void RemoveStUser(int IndividualId)

  After checking the PMAuthorize , calls RemoveStUser function of DBPresenter Class .
- void AddAdmin(int IndividualId)

  After checking the PMAuthorize , calls AddAdmin function of DBPresenter Class .
- void RemoveAdmin(int IndividualId)

  After checking the PMAuthorize , calls RemoveAdmin function of DBPresenter Class .
- void AddProject(….): Arguments taken from the Configuration Manager.

  After checking the CMAuthorize , calls AddProject function of DBPresenter Class .
- void RemoveProject(int ProjectId)

  After checking the CMAuthorize , calls RemoveProject function of DBPresenter Class .
- void AddTask(…..):Arguments taken from the user.

  After checking the CMAuthorize or PMAuthorize or AdminAuthorize , calls the AddTask function of DBPresenter Class .
- void RemoveTask(int TaskId)

  After checking the CMAuthorize or PMAuthorize or AdminAuthorize , calls the RemoveTask function of DBPresenter Class .
- void AddResource(…..):Arguments taken from the Configuration Manager.

  After checking the CMAuthorize , calls AddResource function of DBPresenter Class .

- void RemoveResource(int ResourceId)

  After checking the CMAuthorize ,calls RemoveResource function of DBPresenter Class
- void UpdateResource(….): Arguments taken from the user.

  After checking the CMAuthorize or PMAuthorize or AdminAuthorize , calls the UpdateResource function of DBPresenter Class .
- void AddGroup(…) : Arguments taken from the user.

  After checking the CMAuthorize or PMAuthorize or AdminAuthorize , calls the AddGroup function of DBPresenter Class .
- void RemoveGroup(int GroupId)

  After checking the CMAuthorize or PMAuthorize or AdminAuthorize , calls the RemoveGroup function of DBPresenter Class .
- void UpdateGroup(…) : Arguments taken from the user.

  After checking the CMAuthorize or PMAuthorize or AdminAuthorize , calls the UpdateGroup function of DBPresenter Class .
- void DrawGanttChart(…): Arguments taken from the user.

  After checking the Authentication, calls DrawGanttChart function of DBPresenter Class
- void DrawResourceGraph(…): Arguments taken from the user.

  After checking the Authentication, calls DrawResourceGraph function of DBPresenter Class .
- void DrawNetworkDiagram(…): Arguments taken from the user.

  After checking the Authentication, calls DrawNetworkDiagram function of DBPresenter Class .
- void GenerateReport(….): Arguments taken from the user.

  After checking the Authentication, calls GenerateReport function of DBPresenter Class .
- void Filter(…): Arguments taken from the user.

  After checking the Authentication, calls Filter function of DBPresenter Class .

3.3. Use Cases

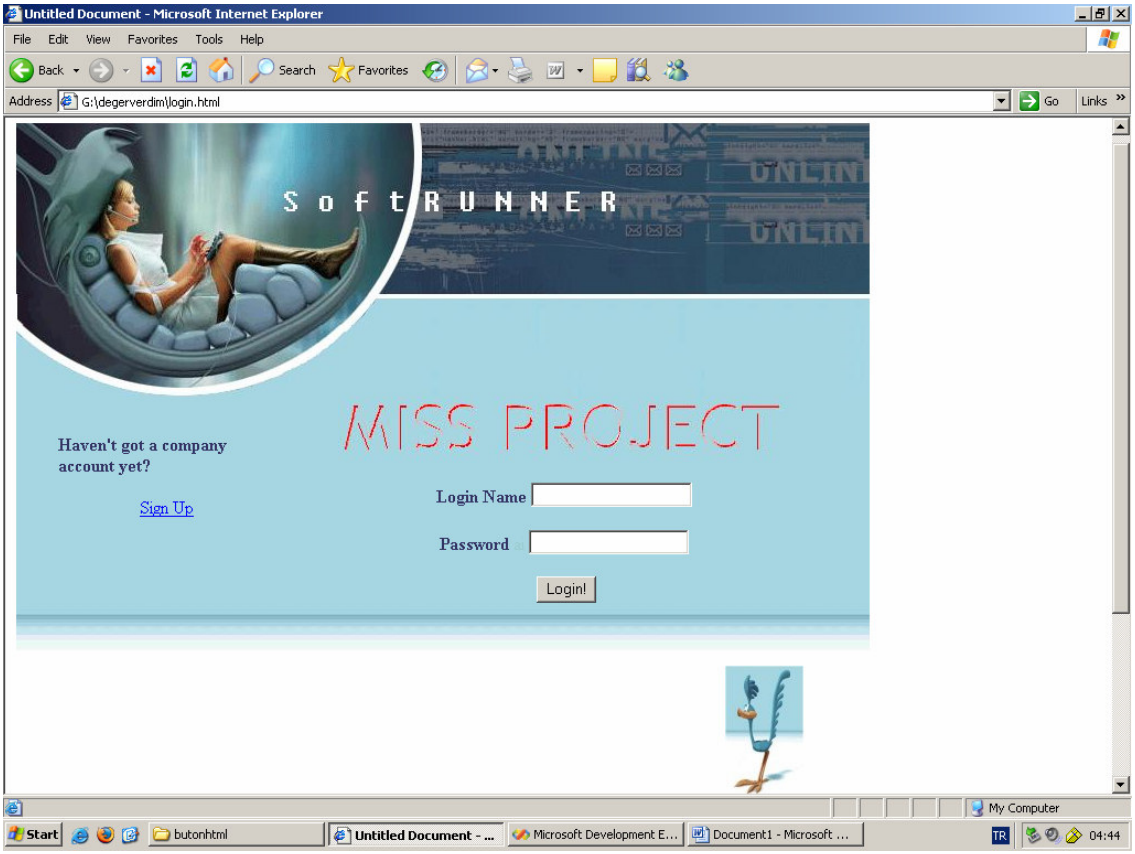
UseCases.sdr (Use Case Diagrams are given as appendix in the hard copy.)

4. INTERFACE DESIGN

4.1. Description of the User Interface

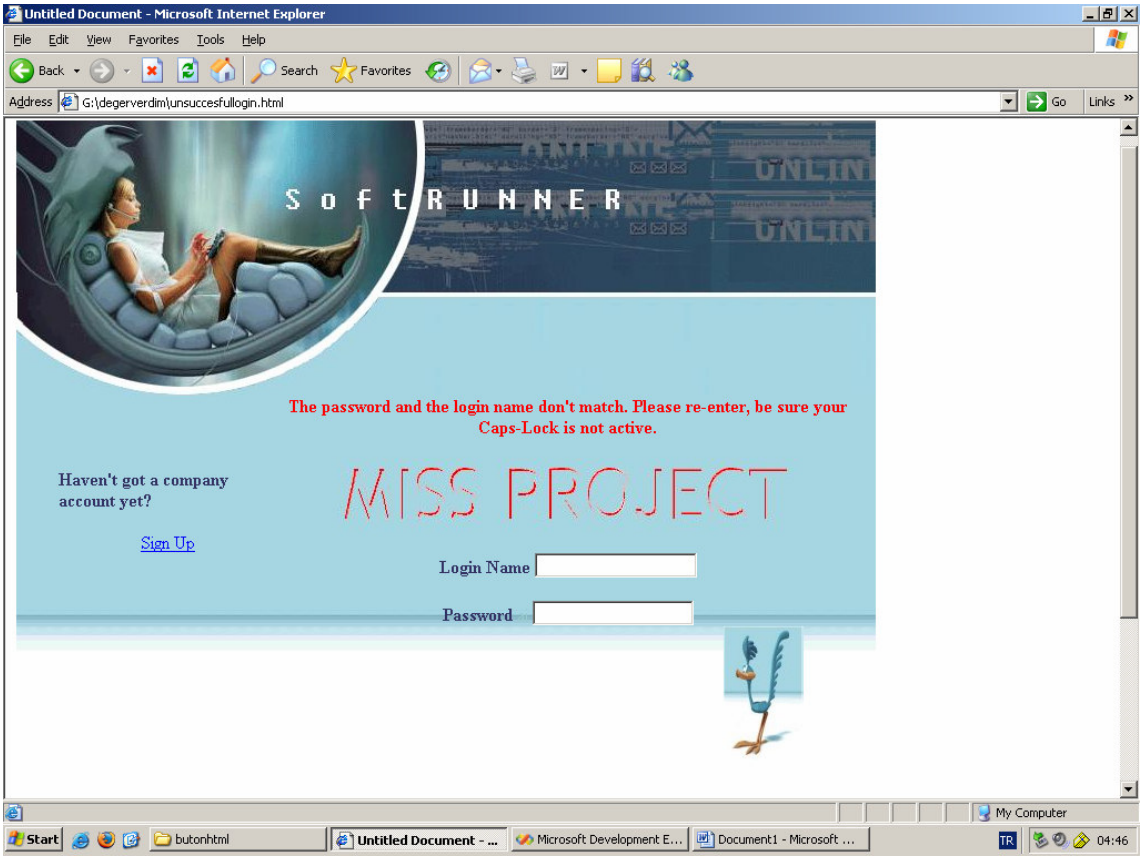
1. Login Interface:

To ensure the security, the system is required to make a password authorization. The user enters his/her login name (which is unique) and the password. If the user wants to register his/her company to the system, he/she follows the Sign Up Link.



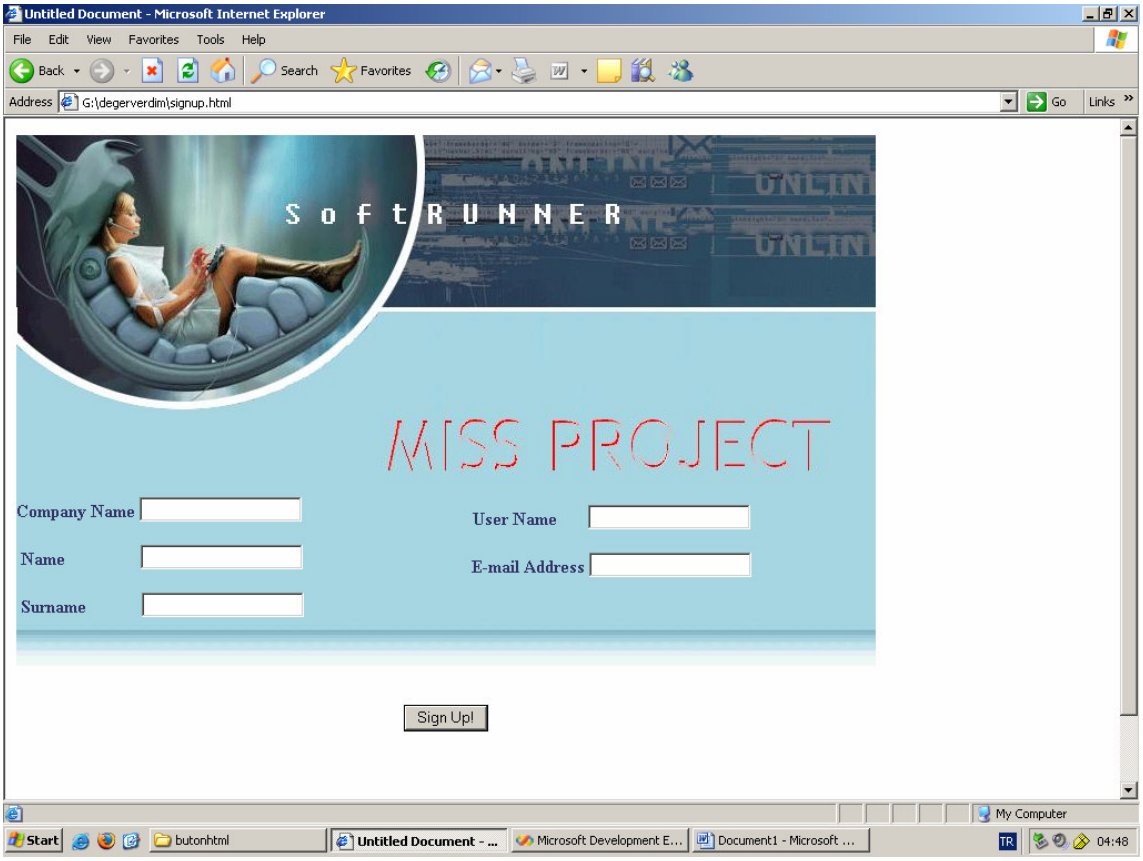
2. Unsuccessful Login Interface:

This page is displayed when the user enters his/her username/ password, or if he/she is not registered.



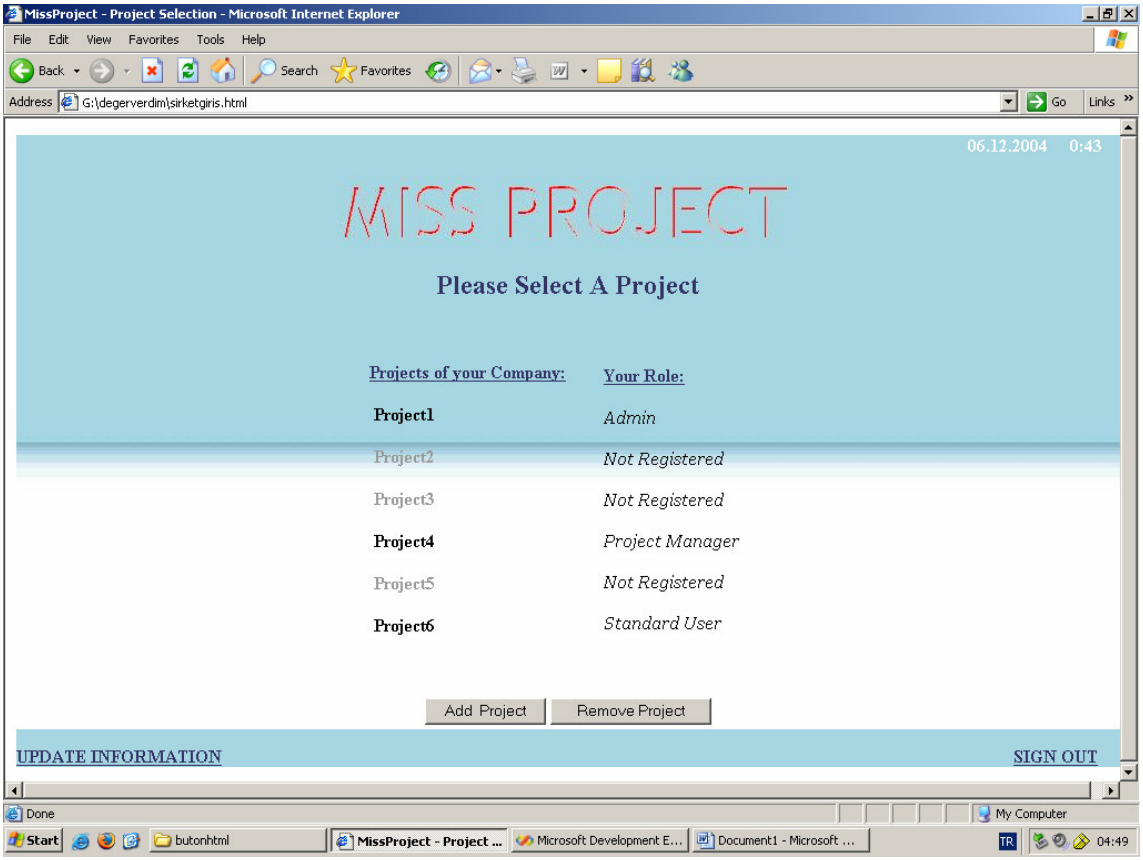
3. Sign-up Interface:

The user who signs up for he company becomes the configuration manager, and thus, the contact point between the company and us. After this process, password is mailed to the new user. And he/she is ready for adding new projects to the account of his/her company.



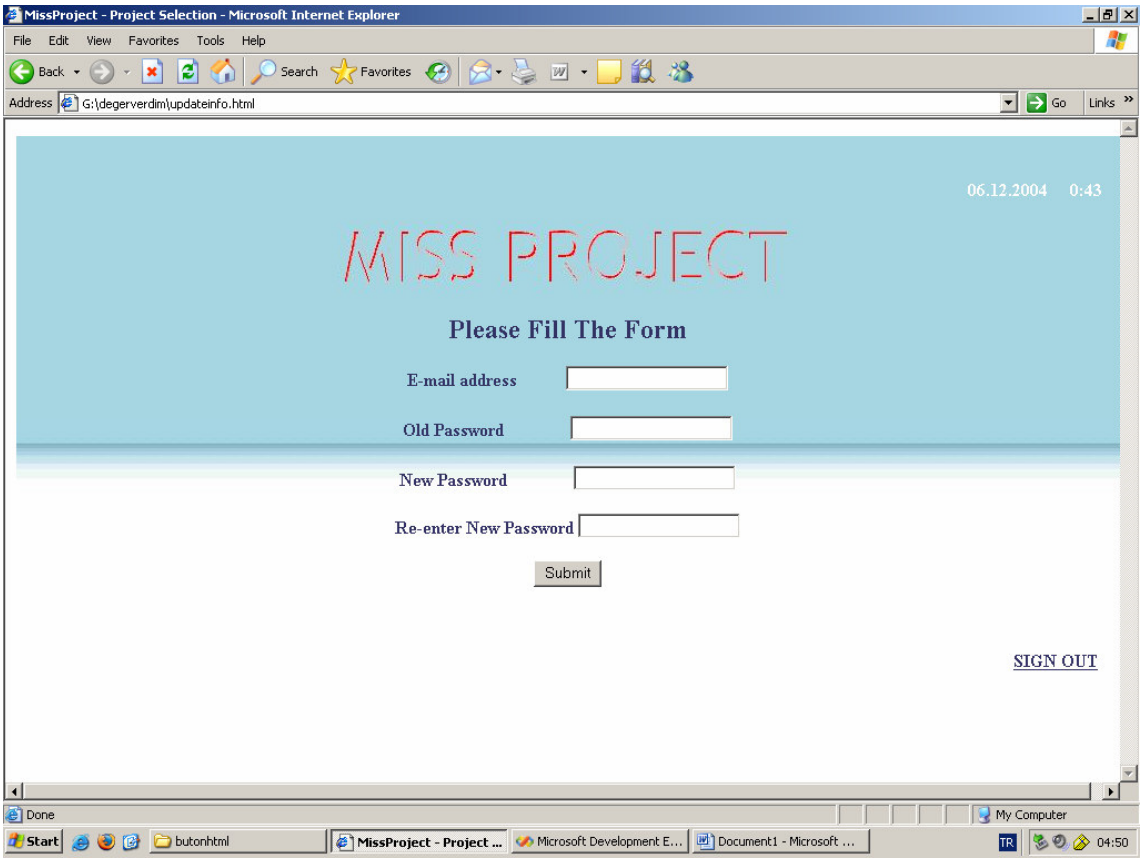
4. Project Choice Interface:

After successful login, the user is directed to the Project selection page of his/her company, according to his/her user name. Project selection page contains the projects of the company. The projects which the current user doesn't have access are disabled, so that the user cannot view the details of these projects. If the current user is the configuration manager of the company, "Add Project" and "Remove Project" buttons are enabled. At this point the user may access to one of the projects that he/she has right to. Or he/she may update personal information by following the "UPDATE INFORMATION" link.



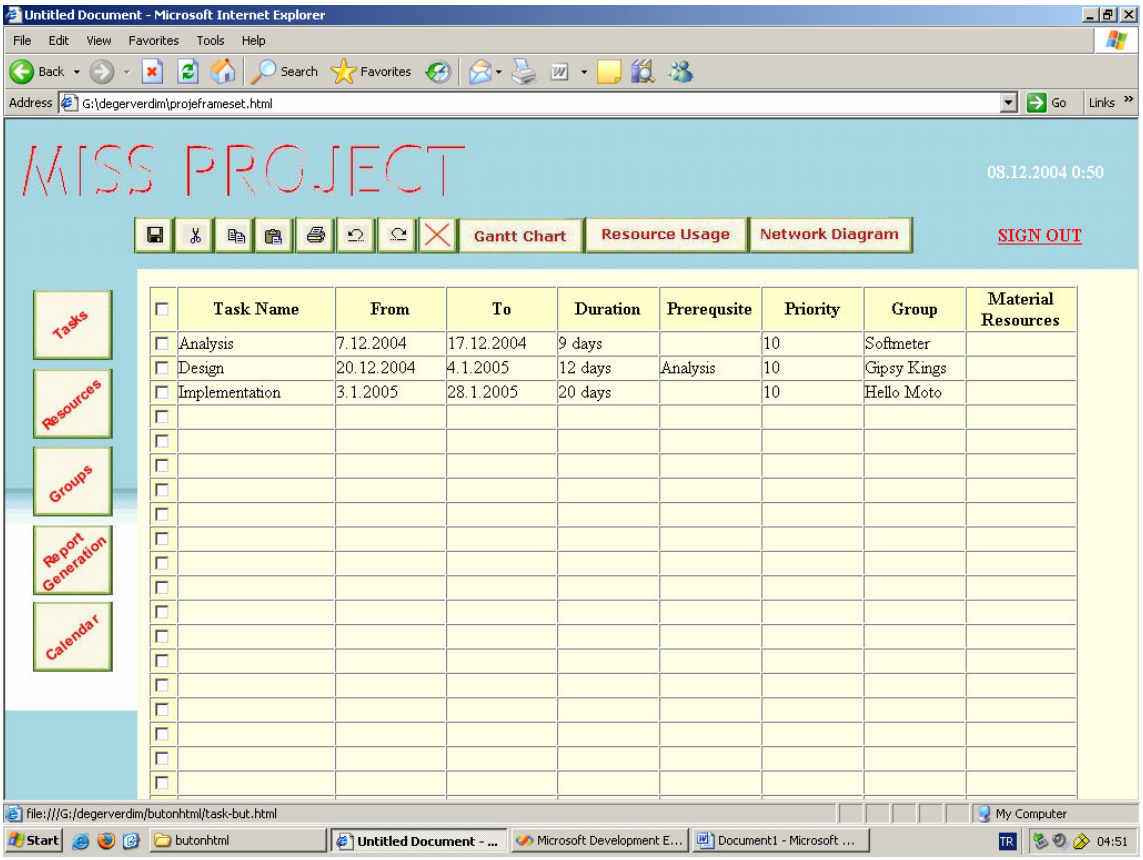
5. Update Personal Information Interface:

The user cannot change his/her username, name and company.



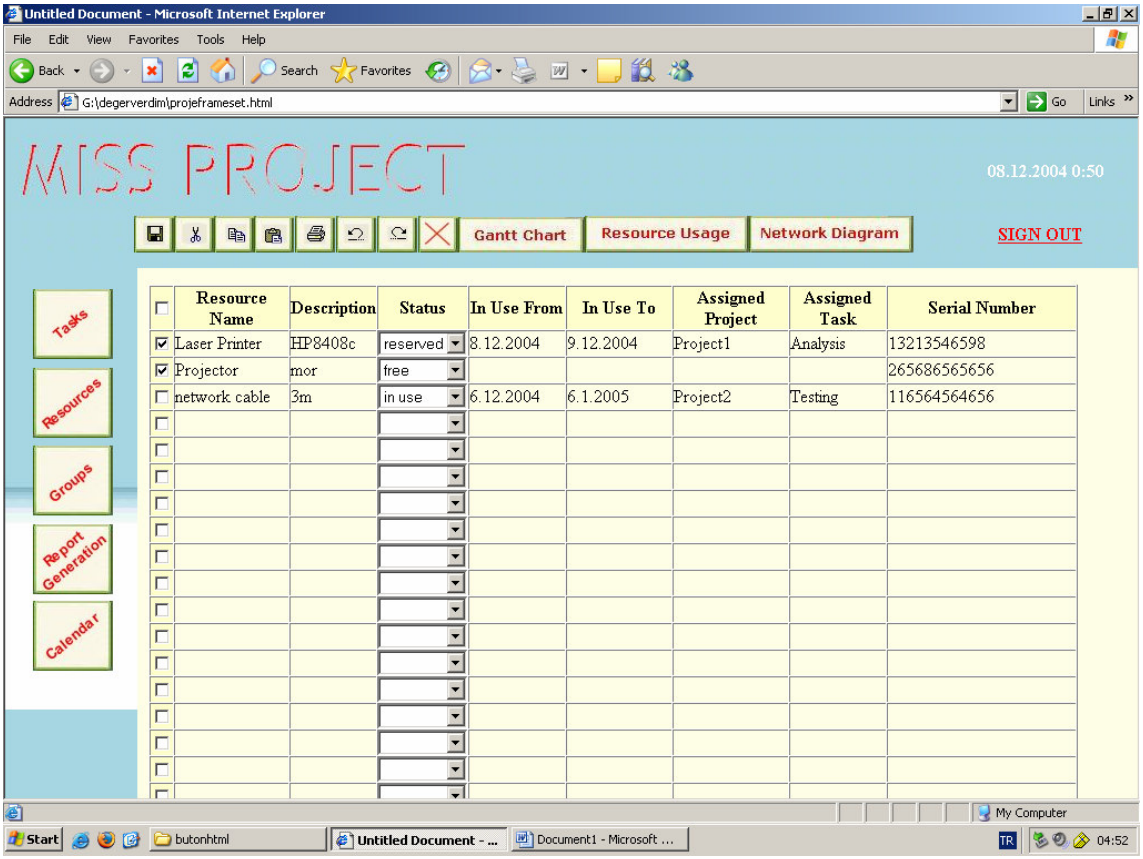
6. Tasks Interface:

At this point, a project is selected and “Tasks” buton is pressed. If the current user is an admin or the manager of the project, he/she can make changes on the project, thus, on the tasks. Before leaving Tasks section, the user is asked if he/she wants to save the changes.



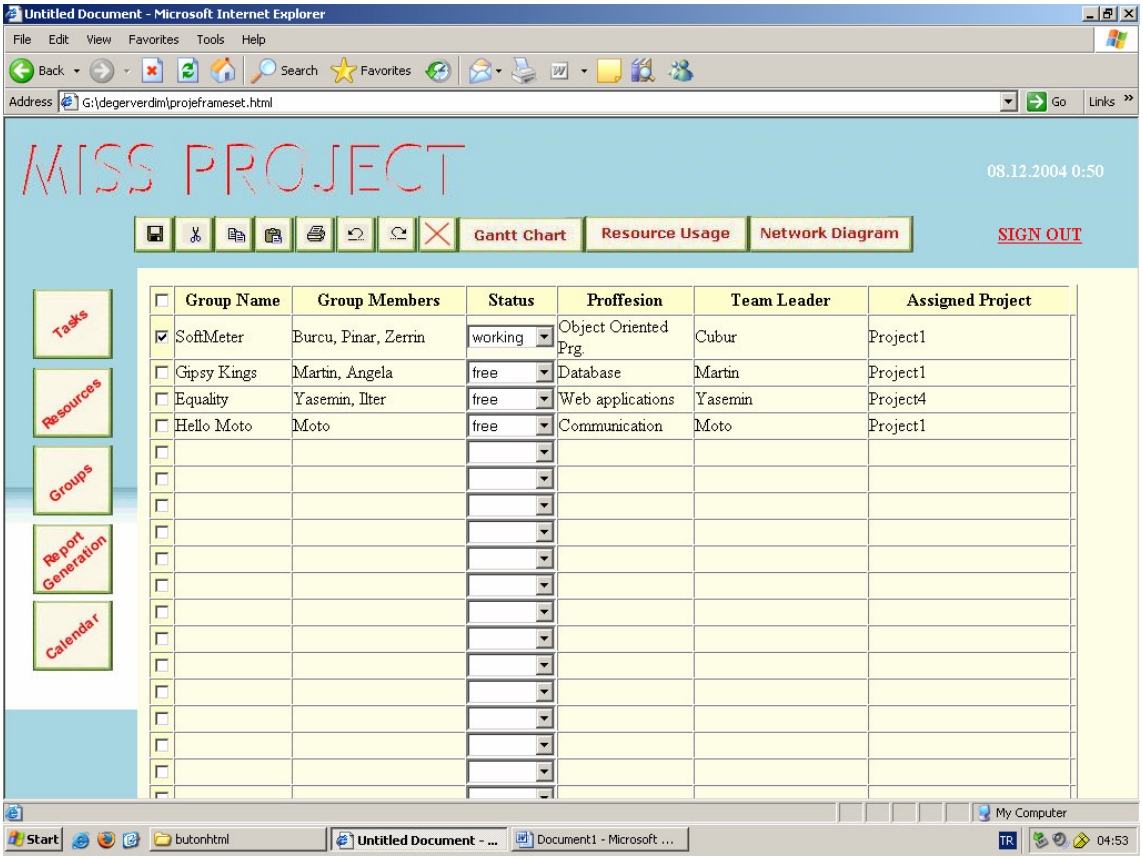
7. Resources Interface:

In this section, resources of the company and their status is viewed. Configuration manager can make updates.



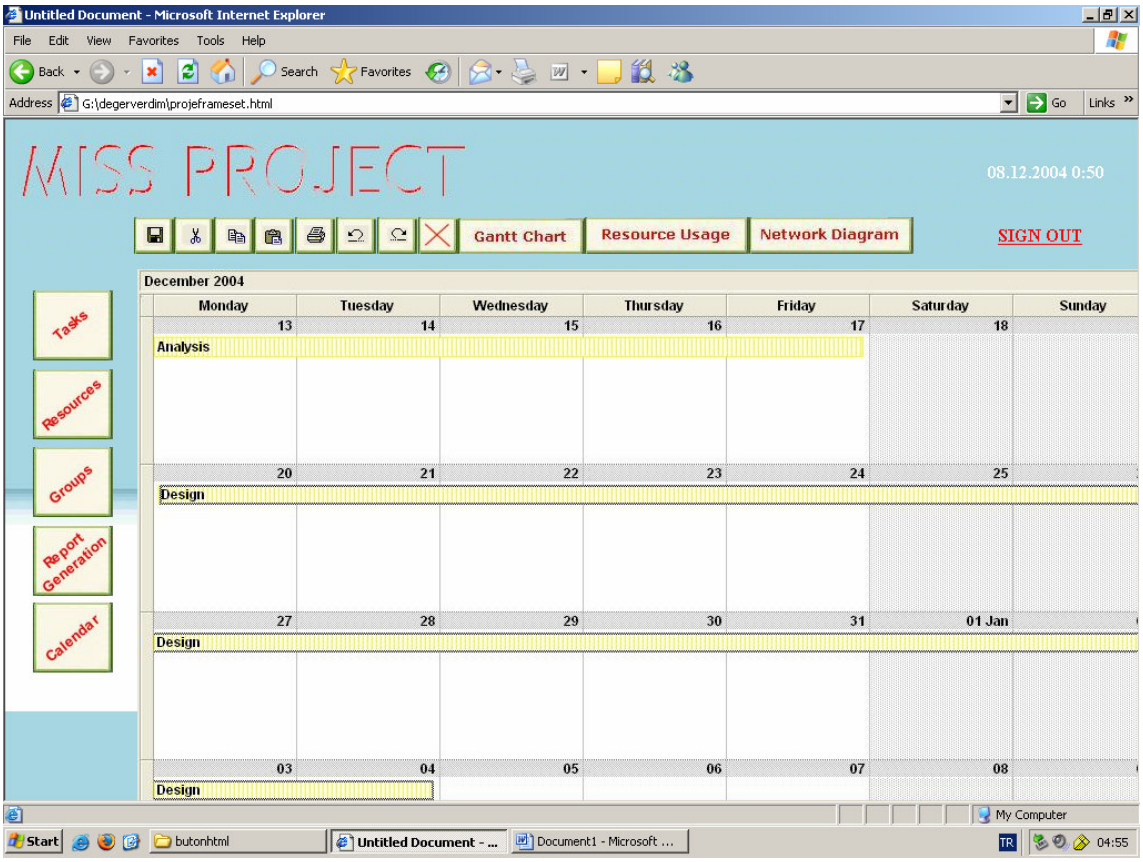
8. Groups Interface:

In this section, resources of the company and their status is viewed. Configuration manager can make updates.



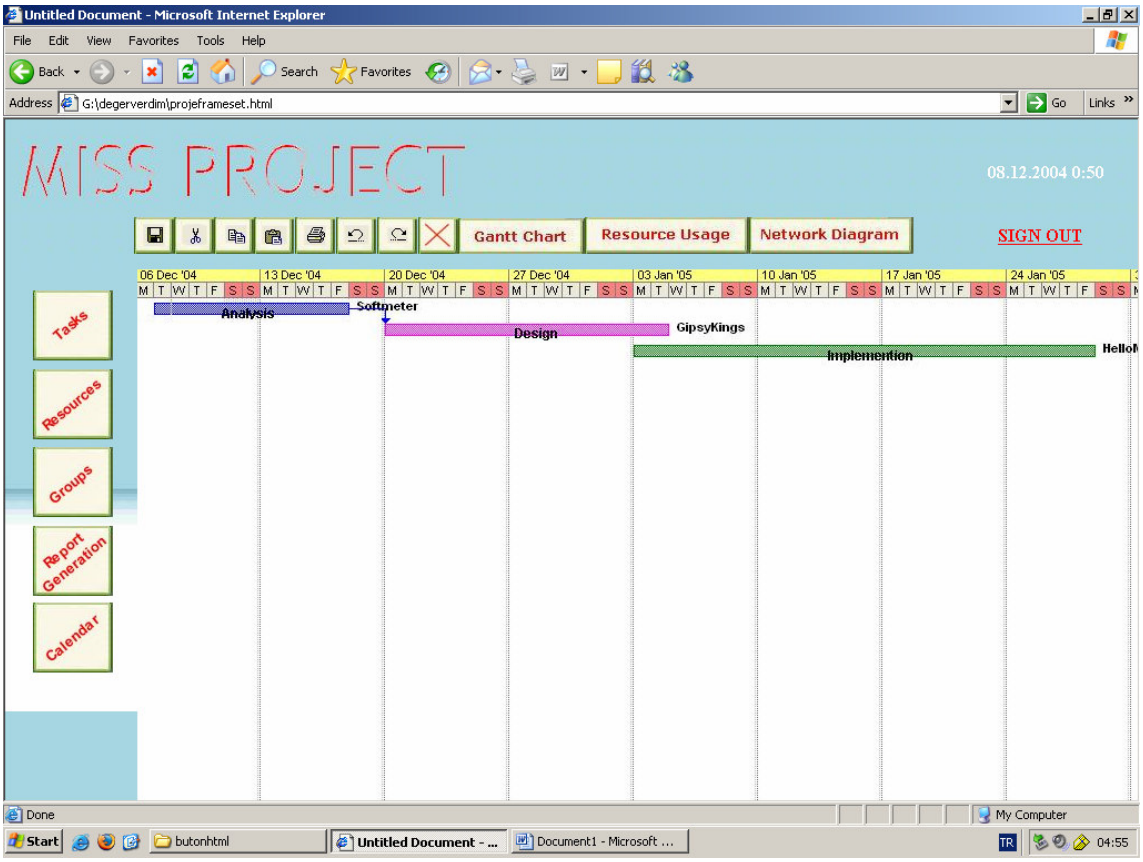
9. Calendar Interface

Calendar is displayed when Calender button is pressed.



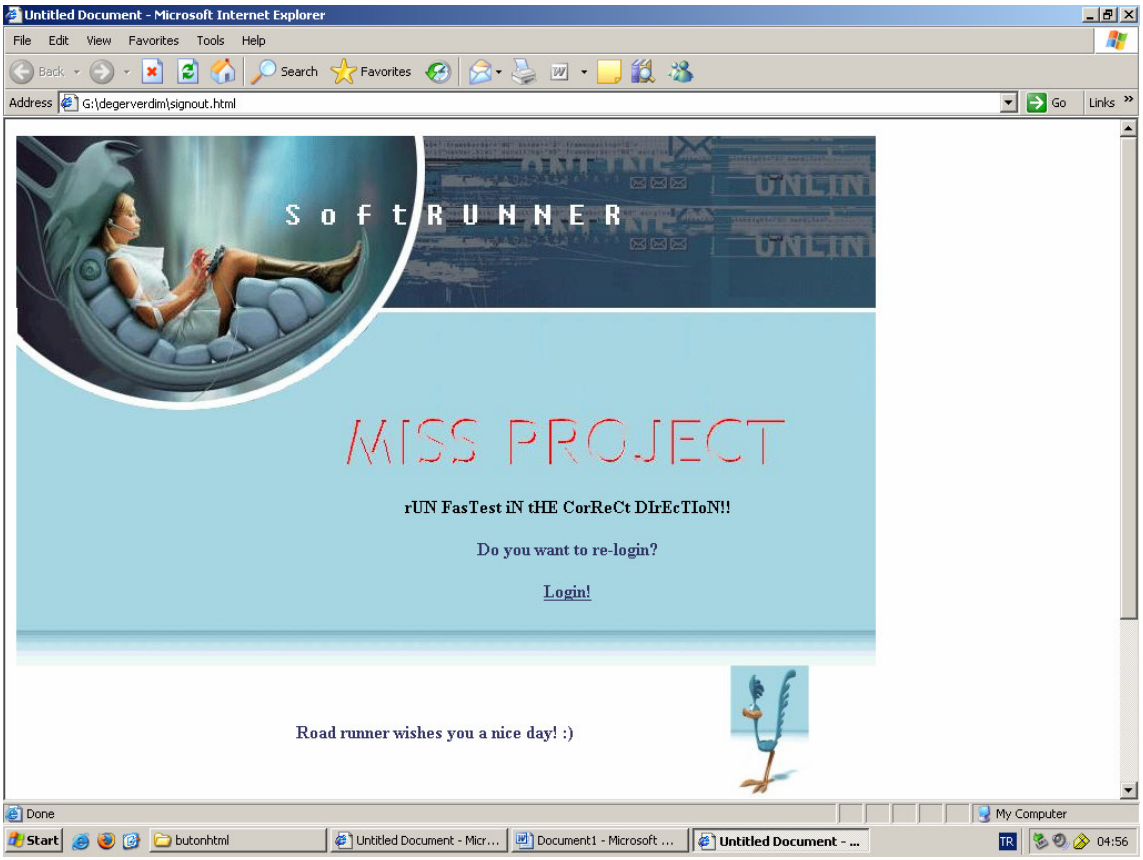
10. Gantt Chart Interface:

Gantt Chart of the project is displayed if “Gantt Chart” buton is pressed.



11. Sign Out Interface:

The user can sign out in every step. He/she may want to login with a different user name.



4.2. Human-machine Interface Design Rules

The following rules are considered while designing the user interfaces:

- The interfaces are made as user-friendly as possible
- The interfaces are consistent. Although every page has a similar look, the information content is well designed.
- Displaying unnecessary or extra information as well as displaying less information than needed is avoided.

- There is a generic error page. When an error occurs, the description of the error along with the possible cause of it and suggestions/warnings to the user are printed. This explanatory error page makes the system user aware of what is happening.

5. PROCEDURAL DESIGN

5.1. Log-in Module

```
//Login Page:
BEGIN login.module
IF PASSWORD.of.the.user==PASSWORD.recorded.in.database
     THEN BEGIN
          IF person.type==configuration.manager
                    THEN goto.configuration.manager.module;
               ELSEIF person.type==project.manager
                    THEN goro.project.manager.module;
               ELSEIF person.type==administrator
                    THEN goto.administrator.module
               ELSE goto.standard.user.module;
          ENDIF
                END
     ELSE display.authentication.failure.page;
ENDIF
END
```

5.2. Configuration Manager Module

```
//Main Page, after logged on as a configuration manager
//fundamental parts of configuration manager module are
//"add project", "delete project", "update company
//information"

BEGIN configuration.manager.module
CASEOF button.PRESSED(b1)
     WHEN b1==add.project
          BEGIN
          show.add.project.typing.screen;
          IF assign.project.manager.button.PRESSED
               THEN BEGIN
               CASE OF button.PRESSED(b2)
                    WHEN b2==select.from.users.list
                         BEGIN
                         show.users.of.company.list;
                         assign.selected.user.to.project;
                    redisplay.add.project.typing.screen;
                         END
                    WHEN b2==add.new.user
                         BEGIN
                         show.add.user.typing.screen;
                         record.new.user.to.database;
                         assign.added.user.to.project;
                    redisplay.add.project.typing.screen;
                         END
                    END CASE
               ENDIF
          record.new.project.to.database;
          redisplay.configuration.manager.main.page;
```

```
          END
     WHEN b1==delete.project
          BEGIN
               show.remove.project.selection.screen;
               remove.selected.project.from.database;
               redisplay.configuration.manager.main.page;
               END
     WHEN b1==update.company.information
          BEGIN
               show.update.company.information.typing.screen;
               record.new.information.to.database;
               redisplay.configuration.manager.main.page;
               END
     WHEN b1==update.personal.information
          BEGIN
               show.update.personal.information.typing.screen;
               record.new.information.to.database;
               redisplay.configuration.manager.main.page;
               END
          END CASE
    END
```

5.3. Project Manager Module

```
//Main Page, after logged on as a project manager
//fundamental parts of project manager module are
//"update project details", "add/remove administrators"

BEGIN project.manager.module
IF project.opretaion.button.PRESSED
THEN BEGIN
CASEOF button.PRESSED(bP)
     WHEN bP == edit.project.details
          BEGIN
          IF technical.processes.button.PRESSED;
               THEN BEGIN
          CASEOF button.PRESSED(b1)
               WHEN b1 == add.task.button.PRESSED
                    BEGIN
                    show.new.task.typing.screen;
                    record.new.task.to.database;
                    redisplay.project.main.page;
                    END
               WHEN b1 == remove.task.button.PRESSED
                    BEGIN
                    show.tasks.selection.screen;
                    remove.selected.task.from.database;
                    redisplay.tasks.selection.screen;
                    END
               WHEN b1 == add.resource.button.PRESSED
                    BEGIN
                    show.new.resource.typing.screen;
                    record.new.resource.to.database;
                    redisplay.project.main.page;
                    END
               WHEN b1 == remove.resource.button.PRESSED
```

```
          BEGIN
          show.reseource.selection.screen;
          remove.selected.resource.from.database;
          redisplay.resources.selection.screen;
          END
     WHEN b1 == add.group.button.PRESSED
          BEGIN
          show.new.group.typing.screen;
          record.new.group.to.database;
          redisplay.group.main.page;
          END
     WHEN b1 == remove.group.button.PRESSED
          BEGIN
          show.group.selection.screen;
          remove.selected.group.from.database;
          redisplay.groups.selection.screen;
          END
     END CASE
     END
ELSEIF assign.roles.button.PRESSED
     THEN BEGIN
          IF add.admin.button.PRESSED
          THEN BEGIN
          IF select.from.users.list.button.PRESSED
               THEN BEGIN
               display.users.of.company.list.screen;
               assign.selected.user.as.admin;
               redisplay.add.admin.selection.screen;
               END
          ELSEIF add.new.user.button.PRESSED
               THEN BEGIN
               display.add.new.user.typing.screen;
               record.new.user.to.database;
               assign.added.user.to.project;
               redisplay.add.admin.typing.screen;
               END
               ENDIF
          END
          ELSEIF add.standard.user.button.PRESSED
          THEN BEGIN
          IF select.from.users.list.button.PRESSED
               THEN BEGIN
               display.users.of.company.list.screen;
               assign.selected.user.as.standard.user;
     redisplay.add.standard.user.selection.screen;
               END
          ELSEIF add.new.user.button.PRESSED
               THEN BEGIN
               display.add.new.user.typing.screen;
               record.new.user.to.database;
               assign.added.user.to.project;
          redisplay.add.standard.user.typing.screen;
               END
               ENDIF
          END
          ELSEIF remove.user.button.PRESSED
               THEN BEGIN
               display.users.of.project.list;
```

```
                         remove.selected.user.from.project;
                         redisplay.remove.user.screen;
                         END
               ENDIF
               END
          ENDIF
     WHEN bP == view.button
          BEGIN
               CASEOF button.PRESSED(b2)
          WHEN b2 == tasks
               BEGIN
                    display.tasks.list.screen;
               END
          WHEN b2 == resources
               BEGIN
               display.resources.list.screen;
               END
          WHEN b2 == groups
               BEGIN
               display.groups.list.screen;
               END
               END CASE
          END
     WHEN bP == graph.generation
          BEGIN
               IF gantt.chart.button.PRESSED
                    THEN BEGIN
                         display.gantt.chart;
                    END
               ELSEIF resource.graph.button.PRESSED
                    THEN BEGIN
                         display.resource.graph;
                    END
               ELSEIF network.diagram.button.PRESSED
                    THEN BEGIN
                         display.network.diagram;
                    END
                    ENDIF
          END
     WHEN bP == report.generation
          BEGIN
          display.report;
          END
     END CASE
     END
ELSEIF update.personal.information.button.PRESSED
     THEN BEGIN
     show.update.personal.information.typing.screen;
     record.new.information.to.database;
     redisplay.project.manager.main.page;
     END
END
```

5.4. Administrator Module

```
// Main Page, after logged on as an administrator
//fundamental parts of administrator module are
//"update project details", "update personal information"

BEGIN administrator.module
IF edit.project.button.PRESSED
     THEN BEGIN
     CASEOF button.PRESSED(b1)
               WHEN b1 == add.task.button.PRESSED
                    BEGIN
                    show.new.task.typing.screen;
                    record.new.task.to.database;
                    redisplay.project.main.page;
                    END
               WHEN b1 == remove.task.button.PRESSED
                    BEGIN
                    show.tasks.selection.screen;
                    remove.selected.task.from.database;
                    redisplay.tasks.selection.screen;
                    END
               WHEN b1 == add.resource.button.PRESSED
                    BEGIN
                    show.new.resource.typing.screen;
                    record.new.resource.to.database;
                    redisplay.project.main.page;
                    END
               WHEN b1 == remove.resource.button.PRESSED
                    BEGIN
                    show.reseource.selection.screen;
                    remove.selected.resource.from.database;
                    redisplay.resources.selection.screen;
                    END
                WHEN b1 == add.group.button.PRESSED
                    BEGIN
                    show.new.group.typing.screen;
                    record.new.group.to.database;
                    redisplay.group.main.page;
                    END
                WHEN b1 == remove.group.button.PRESSED
                    BEGIN
                    show.group.selection.screen;
                    remove.selected.group.from.database;
                    redisplay.groups.selection.screen;
                    END
          END CASE
     END
ELSEIF view.button.PRESSED
     THEN BEGIN
     CASEOF button.PRESSED(b2)
          WHEN b2 == tasks
               BEGIN
                    display.tasks.list.screen;
               END
          WHEN b2 == resources
```

```
                BEGIN
                display.resources.list.screen;
                END
           WHEN b2 == groups
                BEGIN
                display.groups.list.screen;
                END
                END CASE
           END
      WHEN bP == graph.generation
           BEGIN
                IF gantt.chart.button.PRESSED
                     THEN BEGIN
                          display.gantt.chart;
                     END
                ELSEIF resource.graph.button.PRESSED
                     THEN BEGIN
                          display.resource.graph;
                     END
                ELSEIF network.diagram.button.PRESSED
                     THEN BEGIN
                          display.network.diagram;
                     END
                     ENDIF
           END
      WHEN bP == report.generation
           BEGIN
           display.report;
           END
      END CASE

      END
ELSEIF update.personal.information.button.PRESSED
      THEN BEGIN
      show.update.personal.information.typing.screen;
      record.new.information.to.database;
      redisplay.administrator.main.page;
      END
ENDIF
END
```

5.5. Standard User Module

```
// Main Page, after logged on as a standard user
//fundamental parts of standard user module are
//"view project details", "update personal information"

BEGIN standard.user.main.page
IF project.operations.button.PRESSED
      THEN BEGIN
      IF view.button.PRESSED
           THEN BEGIN
           CASEOF button.PRESSED(b1)
                WHEN b1 == tasks
                     BEGIN
                          display.tasks.list.screen;
```

```
                    END
               WHEN b1 == resources
                    BEGIN
                    display.resources.list.screen;
                    END
               WHEN b1 == groups
                    BEGIN
                    display.groups.list.screen;
                    END
                    END CASE
          END
     ENDIF
ELSEIF update.personal.information.button.PRESSED
     THEN BEGIN
     show.update.personal.information.typing.screen;
     record.new.information.to.database;
     redisplay.administrator.main.page;
     END
ENDIF
END
```

6. CLASSES OF TEST

6.1. Interface Testing

We will test our interfaces in the means of;

- Design
- Toolbars
- Buttons
- Menu items in order to check whether they work properly, and they are displayed correctly or not.

6. 2. Black Box Testing

In this method, the system is considered as a black box, that is we only provide it with some inputs and get the corresponding output, no knowledge about internal process is assumed. Black box testing is applied to the interfaces, the descriptions are listed below.

Login Screen: The testing of the login module is easy. All possible combinations of a correct user-id, a wrong user-id, a correct password and a wrong password are entered through the interface. The output is observed. If an erroneous user-id/password pair is entered, the *authentication failure* page is printed, otherwise the user is successfully logged into the system.

Configuration Manager/Project Manager/Administrator/Standard User Main Interfaces: Almost no error can occur through these interfaces since the only input from the user through these interfaces is a click to a button or a link. In other words, the user is restricted with the input capability offered by the interface. But there is a constraint that all the links and the buttons should not be enabled for all users. So the outputs for such conditions must be checked properly. All the links will be followed and checked whether it sends the correct page or not.

For the other interfaces that needs typing some data, or entering some inputs such as Configuration Manager's "Add Project" interface; all the entries will be tested according to the valid inputs. Intentionally wrong inputs will be entered, e.g. a date in a wrong format, and checked whether it sends an error page or not. The content of the error page will be checked

to see if it contains the proper error description and proper solution(s). If all the values are entered correctly, we will check whether the system prints a page that indicates success or not.

The Black Box Testing will be applied to the all interfaces, and the results will be documented with the test report after the implementation phase.

6.3. White Box Testing

Since our system is a web-based application, we cannot simply write some scripts to test the internal processes and functions.

Instead, some software testing tools will be used for automating the white box testing. The class of the tools we use is named as *Capture/Replay Tools*. A capture/replay tool looks like an ordinary browser with an extra capability of *capturing* the events from the beginning of the session. It saves the links you have followed, the passwords you have entered and the values you have filled in a form. Using one of the *Capture/Replay Tools* available off-the-shelf, about 20 test cases are prepared. The test cases are designed carefully considering the internal processes and functions of the system. With this test cases almost all the loops and program structures of the system are explored. While testing the system by a capture/replay tool, some breakpoints are set to inspect the values of the variables and some parameters to verify the proper functioning of the main and sub routines.