CHAPTER 1

INTRODUCTION

1.1 BOAT BORROWING MANAGEMENT SYSTEM

Boat Borrowing Management System is a stand alone application which is generally small or medium in size. It is used by user to manage the borrowing of boats using a computerized system where he/she can record various transactions like issue of boats, return of boats, addition of new boats, list of borrowers with their details etc. Boats maintenance modules are also included in this system which would keep track of borrowers that borrow the boats along with detailed descriptions about the boats a system contains. With this computerized system there will be no loss of boat record or borrower record which generally happens when a non-computerized system is used.

All these modules are able to help the user manage the system with more convenience and in a more efficient way as compared to systems which are not computerized.

1.1.1 PURPOSE

The purpose of this document is to build a software to manage the borrowing of boats. We must be able to provide information without the user having to write complicated queries. Everything must be done at the click of a button. Data about borrowers, boats issued, return dates, dock information, owners and brand of boats, etc. must be readily available.

1.1.2 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

- SRS -Software Requirement Specification.
- DB –Database.
- ER- Entity Relationship.

1.1.3 SCOPE

- A boat borrowing system that keeps track of all boats in various docks.
- The boat must also be categorized in different departments according to type.

- Contains a transaction list of all the boats borrowed, who borrowed said boat, date borrowed and expected date of return.
- Must keep details of all the people who have reserved/borrowed boat.
- Each customer can only rent out two boats at a time.

1.2 EXISTING SYSTEM

The existing system is a manual one. Different records are maintained for different transactions. When a new transaction takes place, the system enters the details of the transactions in a new file depending upon the type of the transaction. The system has to maintain different type of operation like keeping details of the customer i.e. regular customer, detail records of boats, keeping track of customer newly registered moreover financial transactions like income and expenditure for the period. The reports are generated time to time for various operations; these should also be produced to the higher authority in timely manner. The information regarding the system needs interaction and presentation at regular intervals of time.

1.3 PROPOSED SYSTEM AND STATEMENT OF PROBLEM

The current project keeps track of the boats issued, boats returned and the new boats added to the docks. It provides a user-friendly environment where the user can be serviced better. But the drawback of this system is that it does not provide the details about the number of boats need to be purchased in the next season and the total cost of boats to be purchased as per the demand.

CHAPTER 2

SOFTWARE REQUIREMENT SPECIFICATION

2.1 SOFTWARE REQUIREMENTS SPECIFICATION

A software requirements specification (SRS) is a description of a software system to be developed. It lays out functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide. Software requirements specification permits a rigorous assessment of requirements before design can begin and reduces later redesign. It should also provide a realistic basis for estimating product costs, risks, and schedules. Used appropriately, software requirements specifications can help prevent software project failure.

The software requirements specification document enlists enough and necessary requirements that are required for the project development. To derive the requirements, the developer needs to have clear and thorough understanding of the products to be developed or being developed. This is achieved and refined with detailed and continuous communications with the project team and customer till the completion of the software.

2.2 OPERATING ENVIRONMENT

2.2.1 Hardware Requirements

- Processor: Pentium core ,64-bit processor
- RAM: 4GB
- Hard Disk: 40GB
- Main memory: 512 MB RAM
- Hard Disk speed in RPM: 5400 RPM
- Keyboard: standard PS/2 keyboard
- Mouse: 2 or 3 button mouse
- Monitor: generic pnp monitor

2.2.2 Software Requirements

- Operating System: Windows 7 and later
- Programming Language: C#
- Backend Database: SQLite
- IDE used: Visual Studio

2.3 FUNCTIONAL REQUIREMENTS

The functional requirements include:

1. Database

Database implies that a single application should be able to operate transparently on data that is spread across a variety of different tables.

2. Client/Server System

- The term client/server refers primarily to an architecture or logical division of responsibilities, the client is the application (also known as the front-end), and the server is the DBMS (also known as the back-end).
- A client/server system is a distributed system in which, some sites are client sites and others are server sites.
- All the data resides at the server sites.
- All applications execute at the client sites.

3. User Interfaces

- Front-end software: C#
- Back-end software: SQLite

4. Hardware Interfaces

- Windows.
- Visual Studio

2.4 NON-FUNCTIONAL REQUIREMENTS

2.4.1 PERFORMANCE REQUIREMENTS

The steps involved to perform the implementation of boat borrowing system database are as listed below.

A) E-R Diagram

The E-R Diagram constitutes a technique for representing the logical structure of a database in a pictorial manner. This analysis is then used to organize data as a relation, normalizing relation and finally obtaining a relation database.

- ENTITIES: Which specify distinct real-world items in an application.
- PROPERTIES/ATTRIBUTES: Which specify properties of an entity and relationships.
- RELATIONSHIPS: Which connect entities and represent meaningful dependencies between them.

B) Normalization:

The basic objective of normalization is to reduce redundancy which means that information is to be stored only once. Storing information several times leads to wastage of storage space and increase in the total size of the data stored.

If a database is not properly designed it can give rise to modification anomalies. Modification anomalies arise when data is added to, changed or deleted from a database table. Similarly, in traditional databases as well as improperly designed relational databases, data redundancy can be a problem. These can be eliminated by normalizing a database.

Normalization is the process of breaking down a table into smaller tables. So that each table deals with a single theme. There are three different kinds of modifications of anomalies and formulated the first, second and third normal forms (3NF) is considered sufficient for most practical purposes. It should be considered only after a thorough analysis and complete understanding of its implications.

2.4.2 SAFETY REQUIREMENTS

If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was backed up to archival storage (typically tape) and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backend log, up to the time of failure.

2.4.3 SECURITY REQUIREMENTS

Security systems need database storage just like many other applications. However, the special requirements of the security market mean that vendors must choose their database partner carefully.

2.4.4 SOFTWARE QUALITY ATTRIBUTES

- Maintainability: There will be no maintained requirement for the software. The database is provided by the end user and therefore is maintained by this user.
- Portability: The system is developed for secured purpose, so it is can't be portable.
- Availability: This system will available only until the system on which it is install, is running.
- Scalability: Applicable.
- Correctness: The system should maintain the correct details of the boats.

CHAPTER 3

DESIGN

3.1 E-R DIAGRAM

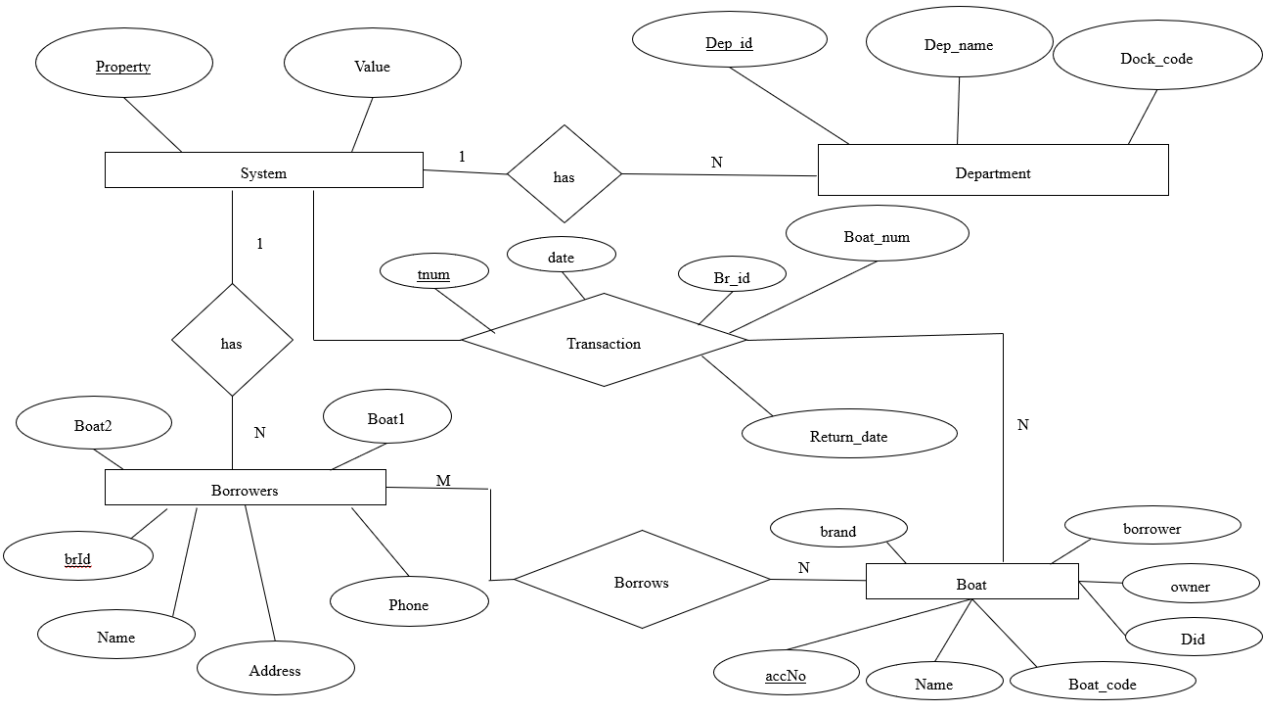

Fig. 3.1 ER Diagram of Database

3.2 SCHEMA DIAGRAM

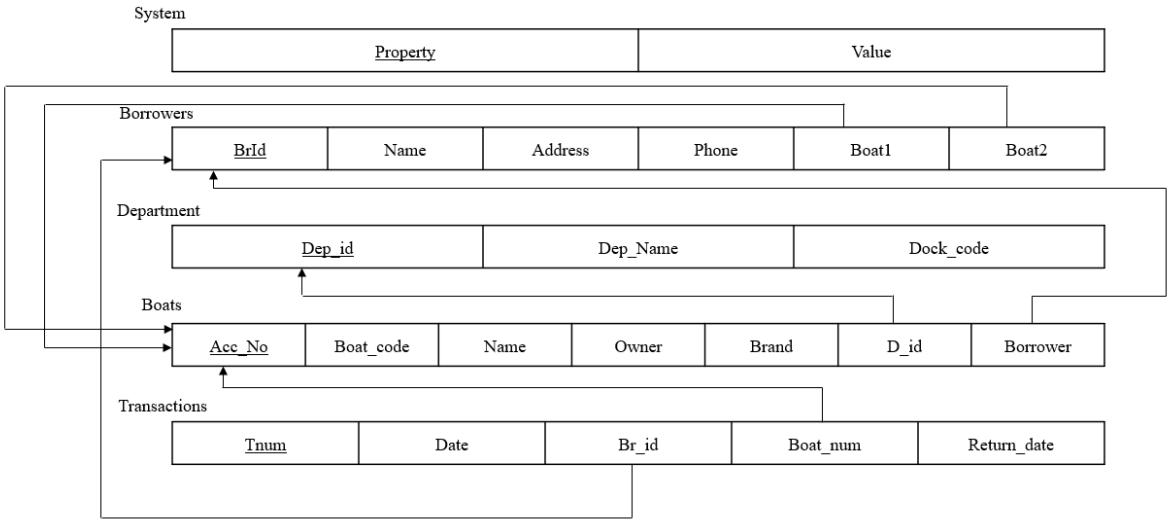

Fig. 3.2 Schema Diagram of Database

3.4 DATA FLOW DIAGRAM

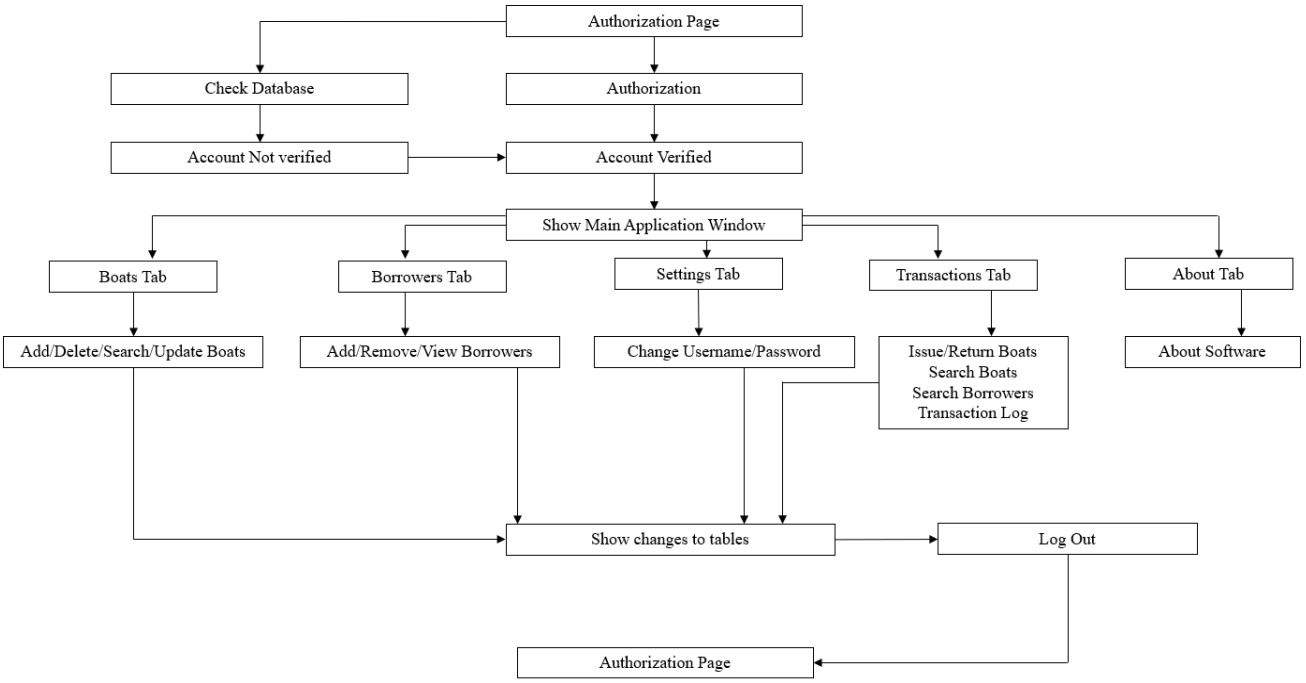

Fig. 3.3 Data Flow Diagram

CHAPTER 4

IMPLEMENTATION

4.1. IMPLEMENTATION

Implementation is the stage in the project where the theoretical design is turned into a working system which gives confidence on the new system for the users for effective and efficient work. It involves careful planning, investigation of the current system and its constraints on implementation, designs of method to achieve change over, an evaluation of change over methods. Apart from planning major task of preparing the implementation are education and training the user. The more complex the system being implemented, the more involved will be the system analysis and the design effort required for its implementation.

In our project the implementation starts from:

1. Creating architecture for the project.

2. Selecting the proper database.

3. Filling up the database.

4. Selecting the proper backend language and server.

5. Selecting the proper frontend language.

6. Creating user friendly frontend

4.2 PROGRAMMING LANGUAGE SELECTION

C#

C# was developed by Microsoft and is used in essentially all of their products. It is mainly used for developing desktop applications and, more recently, Windows 8/10 applications. It is also a part of .NET so it is used alongside languages like ASP in web development and apps. The term is sometimes spelled as C Sharp or C-Sharp. The term's # character derives its name from the musical sharp key, which denotes a one semitone pitch

increase. C# is pronounced "see sharp." C# improved and updated many C and C++ features, including the following:

- C# has a strict Boolean data variable type, such as bool, whereas C++ bool variable types may be returned as integers or pointers to avoid common programming errors.
- C# automatically manages inaccessible object memory using a garbage collector, which eliminates developer concerns and memory leaks.
- C# type is safer than C++ and has safe default conversions only (for example, integer widening), which are implemented during compile or runtime.

SQLite

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private.

SQLite is atomicity, consistency, isolation, durability (ACID) compliant. This embedded relational database management system is contained in a small C programming library and is an integral part of client-based applications. SQLite uses a dynamic SQL syntax and performs multitasking to do reads and writes at the same time. The reads and writes are done directly to ordinary disk files.

An SQLite library is called dynamically and application programs use SQLite functionality through simple function calls, reducing latency in database access. These programs store entire databases as single cross-platform files on host machines. This simple design is implemented by locking the entire database file during a write.

4.3 QUERIES FOR CREATION OF TABLES

1. Boats Table -

```
CREATE TABLE [dbo].[boats] (
    [accNo]    INT          NOT NULL,
    [boatcode] VARCHAR (50) NULL,
    [name]     VARCHAR (50) NULL,
```

```
    [owner]    VARCHAR (50) NULL,
    [brand]    VARCHAR (50) NULL,
    [dId]      INT          NULL,
    [borrower] INT          NULL,
    PRIMARY KEY CLUSTERED ([accNo] ASC),
    CONSTRAINT   [FK_boats_ToTable]   FOREIGN   KEY   ([dId])
REFERENCES [dbo].[department] ([dep_Id]),
    CONSTRAINT  [FK_boats_ToTable_1]  FOREIGN  KEY  ([borrower])
REFERENCES [dbo].[Borrowers] ([brId])
);
```

2. Borrowers Table -

```
CREATE TABLE [dbo].[Borrowers] (
    [brId]    INT          NOT NULL,
    [Name]    VARCHAR (30) NULL,
    [Address] VARCHAR (30) NULL,
    [Phone]   INT          NULL,
    [Boat1]   INT          NULL,
    [Boat2]   INT          NULL,
    PRIMARY KEY CLUSTERED ([brId] ASC),
    CONSTRAINT  [FK_Borrowers_ToTable]  FOREIGN  KEY  ([Boat1])
REFERENCES [dbo].[boats] ([accNo]),
    CONSTRAINT [FK_Borrowers_ToTable_1] FOREIGN KEY ([Boat2])
REFERENCES [dbo].[boats] ([accNo])
);
```

3. Departments Table -

```
CREATE TABLE [dbo].[department] (
    [dep_Id]   INT          NOT NULL,
    [dep_Name] VARCHAR (20) NULL,
    [dockCode] INT          NULL,
    PRIMARY KEY CLUSTERED ([dep_Id] ASC)
);
```

4.System Table -

```
CREATE TABLE [dbo].[systemTable] (
    [Property] VARCHAR (10) NOT NULL,
    [Value]    VARCHAR (20) NULL,
    PRIMARY KEY CLUSTERED ([Property] ASC)
);
```

5.Transactions Table –

```
CREATE TABLE [dbo].[transactions] (
    [Tnum]          INT          IDENTITY (1, 1) NOT NULL,
    [date]          DATE         NULL,
    [br_id]         INT          NULL,
    [boat_num]      INT          NULL,
    [return_date]   DATE         NULL,
    PRIMARY KEY CLUSTERED ([Tnum] ASC),
    CONSTRAINT [FK_transactions_ToTable] FOREIGN KEY ([br_id])
REFERENCES [dbo].[Borrowers] ([brId]),
    CONSTRAINT    [FK_transactions_ToTable_1]    FOREIGN    KEY
([boat_num]) REFERENCES [dbo].[boats] ([accNo])
);
```

4.4 DESCRIPTION OF TABLES

1. Boats Table

Field	Type	Null	Key	Default
accNo	int	NO	PRI	
name	varchar (50)	YES		
boatcode	varchar (50)	YES		
owner	varchar (50)	YES		
brand	varchar (50)	YES		
did	int	YES		
borrower	int	YES		

Table 4.1 Boats Table

2. Department Table

Field	Type	Null	Key	Default
Dep_id	int	NO	PRI	
Dep_name	varchar (20)	YES		
dockCode	int	YES		

Table 4.2 Department Table

3. System table

Field	Type	Null	Key	Default
property	varchar (10)	NO	PRI	
value	varchar (20)	YES		

Table 4.3 System Table

4. Transaction table

Field	Type	Null	Key	Default
tnum	int	NO	PRI	
date	date	YES		
br_id	Int	YES		
Boat_num	Int	YES		
return_date	date	YES		

Table 4.4 Transactions Table

5. Borrowers Table

Field	Type	Null	Key	Default
brId	int	NO	PRI	
name	varchar (30)	YES		
address	varchar (30)	YES		
phone	int	YES		
Boat1	int	YES		
Boat2	int	YES		

Table 4.5 Borrowers Table

4.5 TRIGGERS

We have used triggers on the transactions table to keep a transactions log which can be displayed at the click of a button. In the case of an issue of a boat, the borrower id, time and date of issue along with expected return date is entered into the transactions log. In the case of a return of a boat, we increment the number of transactions but do not keep track of whoever borrowed it because we will already have this data in our transactions table so it would be redundant to keep it in our transactions table as well.

- Transactions Log Issue Trigger
- Transactions Log Return Trigger

Transactions Log Issue Trigger:

```
CREATE TRIGGER Transaction_Log_Return_Trigger
     ON transactions
     instead of delete
     AS
     BEGIN
          declare @br_id int
          declare @boat_num int

          select @br_id = br_id from inserted
          select @boat_num = boat_num from inserted

          insert into Transactions_Log
          values('BORROWER   WITH   ID=   '+cast(@br_id   as
varchar(10)) +
          ' RETURNED THE BOOK WITH ACCNO=' +cast(@boat_num as
varchar(10))+
          'ON '+cast( GETDATE() as varchar(20)))
     END
```

Transactions Log Return Trigger:

```
CREATE TRIGGER Transaction_Log_Issue_Trigger
     ON transactions
     FOR INSERT
     AS
     BEGIN
          declare @br_id int
          declare @boat_num int

          select @br_id = br_id from inserted
          select @boat_num = boat_num from inserted

          insert into Transactions_Log
          values('BORROWER   WITH   ID=   '+cast(@br_id   as
varchar(10)) +
          ' BORROWED THE BOOK WITH ACCNO=' +cast(@boat_num as
varchar(10))+
          'ON '+cast( GETDATE() as varchar(20)))
     END
```

4.6 STORED PROCEDURES

In our project we have thirteen stored procedures for effective functioning. Stored procedures are the safest ways of manipulating data from the front end into the back end. These are our stored procedures –

1.BoatsAdd_SP

This is used to add more boats to the database by taking inputs from the front-end.

2.BoatsDelete_SP

This is used to delete boat entries in the database based on inputs from the front-end.

3.SearchBoat_SP

This is used to search for a particular boat in the database based on input from the front-end.

4.ShowAllBooksData_SP

This is used to display all boats in the database in the front-end.

5.TransactUpdateBoat1_SP

This is used to update the value of Boat1 in the database in the case of issue or return of the boat.

6.TransactUpdateBoat2_SP

This is used to update the value of Boat1 in the database in the case of issue or return of the boat.

7.TransactUpdateBorrower_SP

This is used to update the value of Borrower in the database in the case of issue or return of the boat.

8.Transactions_Insert_SP

This is used to update the transactions table in the database in the case of issue of a boat.

9.Transactions_Delete_SP

This is used to update the transactions table in the database in the case of return of a boat.

10.Borrowersadd_SP

This is used to add more borrowers to the borrowers table in the database.

11.Borrowersdelete_SP

This is used to remove borrowers from the borrowers table in the database.

12.Changeuserpass_SP

This is used to change the login credentials of a user in the database.

13.Showallborrowers_SP

This is used to show all the borrowers present currently in the database.

Code:

1.BoatsAdd_SP

```
CREATE PROCEDURE [dbo].BoatsAdd_SP
     @accNo int,
     @boatcode varchar(10),
     @name varchar(30),
     @owner varchar(30),
     @brand varchar(30),
     @dId int
AS
     insert   into   boats(accNo,boatcode,name,owner,brand,dId)
values(@accNo,@boatcode,@name,@owner,@brand,@dId)
RETURN 0
```

2.BoatsDelete_SP

```
CREATE PROCEDURE [dbo].BoatsDelete_SP
     @accNo int
AS
     delete from boats where accNo=@accNo
RETURN 0
```

3.SearchBoat_SP

```
CREATE PROCEDURE [dbo].SearchBoat_SP
     @accNo int
AS
     SELECT b.accNo, b.boatcode,   b.name, b.owner, b.brand,
d.dep_Id, d.dep_Name,d.dockCode from boats b,department d where
b.dId=d.dep_Id and b.accNo=@accNo
```

```
RETURN 0
```

4.ShowAllBooksData_SP

```
CREATE PROCEDURE [dbo].ShowAllBooksData_SP
AS
SELECT b.accNo, b.boatcode, b.name, b.owner, b.brand, d.dep_Id,
d.dep_Name,d.dockCode from boats b,department d where
b.dId=d.dep_Id
RETURN 0
```

5.Transact_Update_Boat1_SP

```
CREATE PROCEDURE [dbo].Transact_Update_Boat1_SP
     @brId int,
     @accNo int
AS
     update Borrowers set Boat1=@accNo where brId=@brId
RETURN 0
```

6. Transact_Update_Boat2_SP

```
CREATE PROCEDURE [dbo].Transact_Update_Boat2_SP
     @brId int,
     @accNo int
AS
     update Borrowers set Boat2=@accNo where brId=@brId
RETURN 0
```

7.Transact_Update_Borrower_SP

```
CREATE PROCEDURE [dbo].[Transact_Update_Borrower_SP]
     @brId int,
     @accNo int
AS
     update boats set borrower=@brId where accNo=@accNo
RETURN 0
```

8.Transactions_Delete_SP

```
CREATE PROCEDURE [dbo].Transactions_Delete_SP
     @br_id int,
     @boat_num int
AS
     delete from transactions where br_id=@br_id and boat_num=
@boat_num
RETURN 0
```

9.Transactions_Insert_SP

```
CREATE PROCEDURE [dbo].Transactions_Insert_SP
     @br_id int,
     @boat_num int
AS
     insert into transactions
     (date,br_id,boat_num,return_date)
     values
     (convert(date,getdate()),
     @br_id,
     @boat_num,
     DATEADD(WEEK,2,convert(date,getdate()))
     )
RETURN 0
```

10.Borrowersadd_SP

```
CREATE PROCEDURE [dbo].borrowersadd_SP
     @brId int,
     @Name varchar(30),
     @Address varchar(30),
     @Phone int
AS
     insert       into       Borrowers(brId,Name,Address,Phone)
values(@brId,@Name,@Address,@Phone)
RETURN 0
```

11.Borrowersdelete_SP

```
CREATE PROCEDURE [dbo].borrowersdelete_SP
     @brId int
AS
     delete from Borrowers where brId=@brId
RETURN 0
```

12.Changeuserpass_SP

```
CREATE PROCEDURE [dbo].changeuserpass_SP
     @username varchar(10),
     @password varchar(20)
AS
     update    systemTable    set    Value=@username    where
Property='UserName'
     update    systemTable    set    Value=@password    where
Property='Password'
RETURN 0
```

13.Showallborrowers_SP

```
CREATE PROCEDURE [dbo].showallborrowers_SP
AS
     SELECT * from Borrowers
RETURN 0
```

CHAPTER 5

RESULTS

We built a working project for boat borrowing management system. It is a stand-alone system. We keep track of boats issues and borrowed in our simple and user-friendly interface. It is very simply to find out boats that are available, boats that have been borrowed and who has borrowed set boats. The multiple tabs on the left-hand side allow users to easily procur information they require. Further scope would include perhaps adding a fine to borrowers who do not return boats on time. We are also in the process of developing an algorithm that will allow us to predict the demand of boats according to season because would give us helpful data. Here are some screenshots of our system –

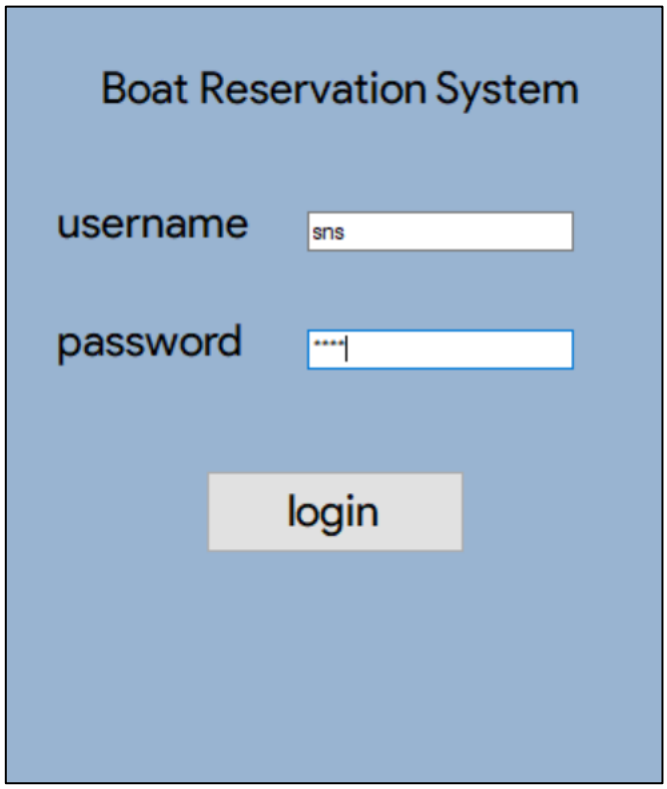

Fig. 5.1 Authorization Page

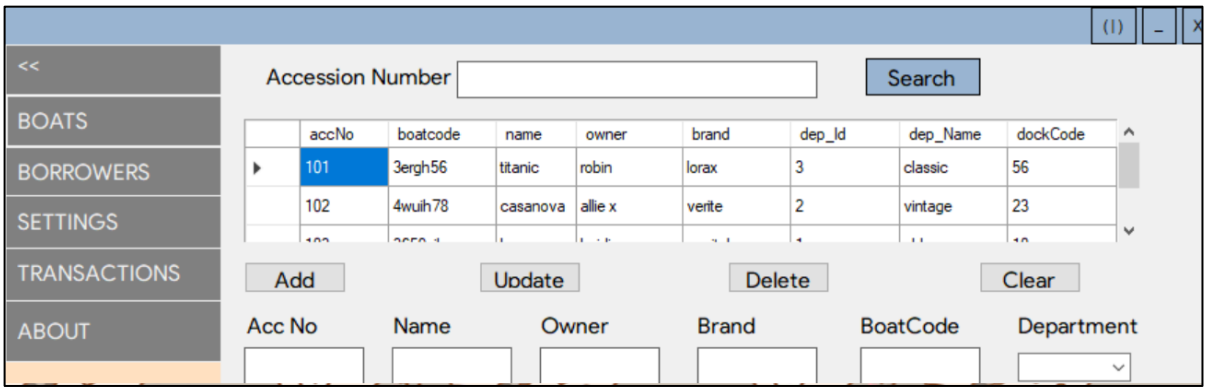

Fig. 5.2 Boats Tab

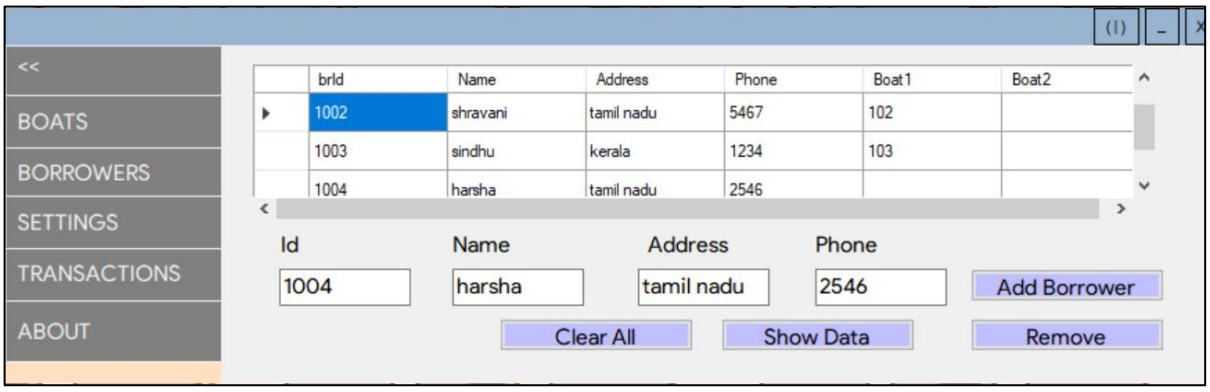

Fig. 5.3 Borrowers Tab

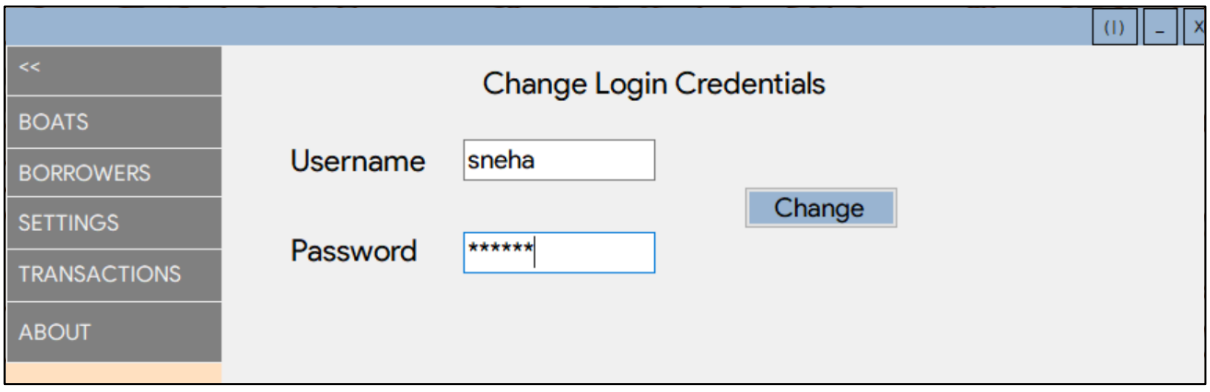

Fig. 5.4 Settings Tab

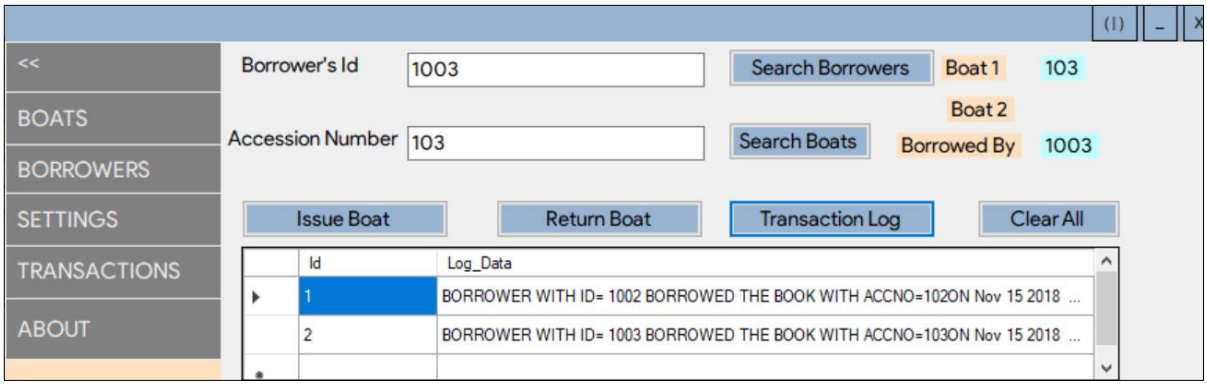

Fig. 5.5 Transactions Tab

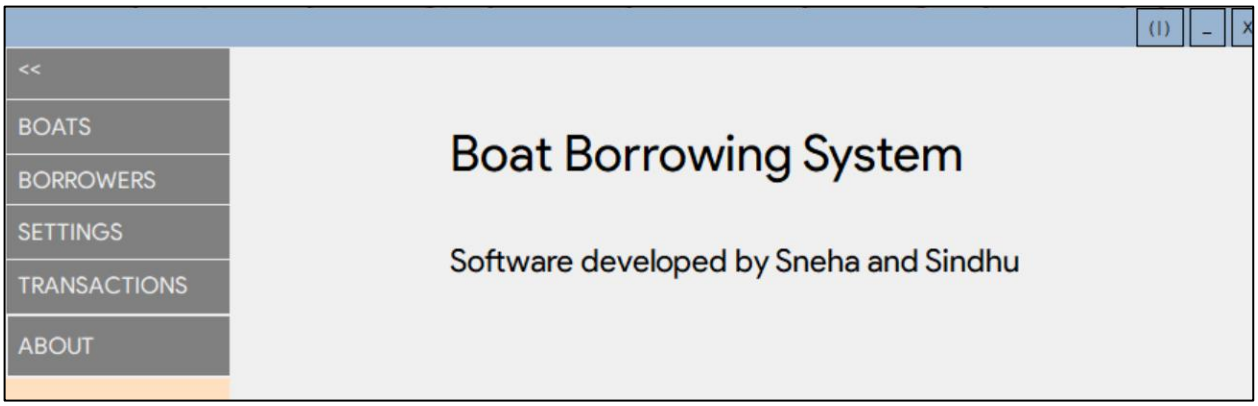

Fig. 5.6 About Tab

CHAPTER 6

REFERENCES

- https://docs.microsoft.com/en-gb/visualstudio/?view=vs-2017
- Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, Pearson, 7th Edition, 2017.
- Database management systems, Ramakrishnan, and Gehrke, McGraw Hill, 3rd Edition, 2014,.