**Acknowledgement**

We would like to give our sincere acknowledgement to everybody responsible for the successful completion of our project titled “**Digital Locker**”.

It gives us great pleasure to acknowledgement with thanks the assistance and contribution of many individual who have been actively involved at various stages of this project to make it a success. We wish to express our whole hearted thanks to Mr. Brijesh Singh for providing the lab facilities in our institute and complete support. Perfection is impossible to achieve, but hard work and dedication can achieve any goal. We have tried our level best to fulfill the requirements of the project, but We could not have achieved the goal without the guidance of Mr. Dharmendra. We feel indebted to Mr. Dharmendra, for providing us an opportunity to work under the guidance and for his constant support and encouragement. Our gratitude towards our guide Mr. Dharmendra for their keen interest and encouragement, which is received during the entire period of our project .We would also like to thank all our lecturers.

Finally we convey our acknowledgement to all our friends and family members who directly or indirectly associated with us in the successful completion of the project.

We thank one and all.

**Abstract**

*Digital locker is basically an electronic facility. This would prove useful to users in keeping their important document such as Pan card, Adhar card and passport etc. in the digital format. This project has been done with an Aim to provide online storage facilty to registered customers. Sharing documents with various private agences for different purposes is tedious task. Digital Locker has been developed to minimize the usage of physical documents and enable sharing of e-documents across agencies or at any place.*

*It is a repository of digital documents for each registered customer. Digital Locker helps its users to store documents, certificates etc. Digital Locker empowers the user with features like registration, login, upload documents, download uploaded documents and delete documents keeping the documents secured at server.*

**CHAPTER 1: INTRODUCTION**

**1.1 Background**

**1.2 Objectives**

**1.3 Purpose, Scope, and Applicability**

**1.3.1 Purpose**

**1.3.2 Scope**

**1.3.3 Applicability**

**1.4 Achievements**

**1.5 Organization of Report**

**CHAPTER 2: SURVEY OF TECHNOLOGIES**

**CHAPTER 3: REQUIREMENTS AND ANALYSIS**

**3.1 Problem Definition**

**3.2 Requirements Specification**

**3.3 Planning and Scheduling**

**3.4 Software and Hardware Requirements**

**3.5 Preliminary Product Description**

**3.5.1. Objectives**

**3.5.2. Functions of Application System**

**3.6 Conceptual Models**

**3.6.1. ER Diagram( Entity Relation Diagram):**

**3.6.2. ER Diagram Symbols, and Notations**

**3.6.3. DFD( Data Flow Diagram)**

**3.6.4. Use Case Diagram**

**3.6.5. Class Diagram**

**3.6.6. Sequence Diagram**

**CHAPTER 4: SYSTEM DESIGN**

**4.1 Basic Modules**

**4.2 Data Design**

**4.2.1 Schema Design**

**4.2.2 Data Integrity and Constraints**

**4.3 Procedural Design**

**4.3.1 Logic Diagrams**

**4.3.2 Data Structures**

**4.3.3 Algorithms Design**

**4.4 User interface design**

**4.5 Security Issues**

**4.5.1. Real-time issues.**

## 4.5.2. Data Validation.

**4.6 Test Cases Design**

**CHAPTER 5: IMPLEMENTATION AND TESTING**

**5.1 Implementation Approaches**

**5.2 Coding Details and Code Efficiency**

**5.2.1 Code Efficiency**

**5.3 Testing Approach**

**5.3.1 Unit Testing**

**5.3.2. Functional Testing**

**5.3.3 Integrated Testing**

**5.4 Modifications and Improvements**

**CHAPTER 6: RESULTS AND DISCUSSION**

**6.1 Test Reports**

**6.2 User Documentation**

**CHAPTER 7: CONCLUSIONS**

**7.1 Conclusion**

**7.2 Limitations of the System**

**7.3 Future Scope of the Project**

**REFERENCES**

**GLOSSARY**

**1. INTRODUCTION**

**1.1 Background**

Currently, in India, almost all of the government issued documents are in physical form across the country. This means every time a user needs to share the document with an agency to avail any service, an attested photo copy either in physical form or on scanned form is shared. Use of physical copies of document creates huge overhead in terms of manual verification, paper storage, manual audits, etc. incurring high cost and inconvenience. This creates problem for various agencies to verify the authenticity of these documents, thus, creating loopholes for usage of fake documents/certificates. Due to the nature of these documents not having a strong identity attached to it, anyone with same name can indeed misuse someone else’s document.

**1.2. Objective**

* The objective of the project is to develop a system that empowers the user to upload their documents online and keep them safe.
* Minimize the use of physical documents. Uploading your certificates like those related to birth, marriage, income and caste certificates on the Digital-locker will ensure that there is no need to carry them in the physical format when they are needed for educational or job applications.

**1.3. Purpose and Scope**

**1.3.1. Purpose**

Digital Locker is an online web application. It allows the registered users to store scanned or pdf formats of their documents. The Digital Locker allows the users to access their documents digitally with just one click of login to their respective account. The online locker gives the secure platform to upload documents at ease. The Registered user will have their profile created by the system as soon as user registered him/herself. Locker has been developed with an intent to reduce the hassle of a user’s day to day life. We all understand carrying documents can be very difficult at times especially when you have to carry bulk of documents which not only increases the chances of documents being misplaced but also a threat is already there. Digital locker will resolve this problem efficiently because after this user can just upload the documents and does not have to carry them at everywhere which will be a huge advancements over digitization.

**1.3.1. Scope**

The primary job of Digital Locker is to give store documents online securely. If user is not registered on platform. Then the user will first require to get registered. The Authentication page will have option to register or login depending on the existence of user in system. The main limitation of using the Digital locker is that only registered users are allowed to upload their documents only when they are registered on web application. If user is not registered he/she will get the error to get registered to be able to upload any documents. The upload option allows the user to upload documents i.e scanned images, pdf and word files.

* Issues covered:

|  |  |
| --- | --- |
| Efficiency: | Considering the significance of having documents such as registration certificate, Driving license these type of documents are frequently used so we have planned to design a system which will allow user to upload them online and keep a soft copy of it. |
| Security: | The Digital Locker is designed with an intent to keep user data safe secured server |
|  | No one will have the access apart from registered user because document view is only allowed for registered user. |
| Effort: | The Digital Locker reduces the effort |

**Table 1: Includes issues which has been covered**

* Functions

|  |  |
| --- | --- |
| * Login: | * Every registered will have the option of login to his/her account. * Dashboard page will show the files/documents which has been upload by user so |
| * Dashboard: | * Apart from Deleting and Uploading the files user will also have the option to download file. |

**Table 2: Functions**

Applicability:

The platform has the following applications:

* User can access their digital documents anytime, anywhere and share it online. This is convenient and time saving.
* It can reduce the administrative overhead of Government departments by minimizing the use of paper.
* Digital Locker makes it easier to validate the authenticity of documents as they are issued directly by the registered issuers.
* Self-uploaded documents can be digitally signed using the eSign facility (which is similar to the process of self-attestation).

**1.4. Achievements**

After concluding the project we have learnt below methodologies.

* Project Planning
* API
* Database Integration
* Digital Security
* Web bases technologies

Ultimate aim of the behind the digital locker implementation is to make it easy for users to upload their content online. This goal has certainly been achieved with the web based implementation. At present there is no digital locker(Web Based) with the web application that we have built this has capability of revolutionize the way of thinking of active users online.

**1.5. Organization of Report**

In the next chapters of this report we will take your through the technologies that we have used to complete this project. As there has been many technologies used in this completion of this project so indeed a thorough discussion is required. We have also a chapter which emphasis on how requirement analysis is important in this project, System design and data design is one of the key part of this project which is also discussed in the further sections. Testing is considered to be one of the important aspect of any designed system. This has been also considered as a matter of discussion.

**2. SURVEY OF TECHNOLOGIES**

**Angular**: Angular was introduced in October 2014 as a new framework to enable developers to build web applications for Web platforms and mobile. Since then the framework has evolved very much and has given lot of advantage to developers when it comes to write code. [[1]](https://paperpile.com/c/Rff4HW/HZQW)Angular is a structural framework for dynamic web apps. It lets us use HTML as your template language and lets us extend HTML's syntax to express your application's components clearly and succinctly. The framework is useful handling frontend of any web app which is being developed. The framework is used to design the Single page web applications.

Features:

The general features of Angular are as follows-

* Angular provides developers an options to write client side applications using JavaScript in a clean Model View Controller (MVC) way.
* Angular is open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache license version 2.0.

The core features of Angular are as follows-

* Data-binding: It is the automatic synchronization of data between model and view components.
* Scope: These are objects that refer to the model. They act as a glue between controller and view.
* Controller: These are JavaScript functions bound to a particular scope.
* Services: Angular comes with several built-in services such as $http to make a XMLHttpRequests. These are singleton objects which are instantiated only once in app
* Directives: Directives are markers on DOM elements such as elements, attributes, CSS, and more. These can be used to create custom HTML tags that serve as new, custom widgets. AngularJS has built-in directives such as ngBind, ngModel, etc.
* Templates: These are the rendered view with information from the controller and model. These can be a single file (such as index.html) or multiple views in one page using partials.
* Routing: It is concept of switching views.
* Dependency Injection: Angular has a built-in dependency injection subsystem that helps the developer to create, understand, and test the applications easily.

**PostgreSQL** :

PostgreSQL [[2]](https://paperpile.com/c/Rff4HW/HZQW)is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of [[2]](https://paperpile.com/c/Rff4HW/HZQW+KmkV)PostgreSQL date back to 1986 as part of the [POSTGRES](https://www.postgresql.org/docs/current/history.html) project at the University of California at Berkeley and has more than 30 years of active development on the core platform.

PostgreSQL has earned a strong reputation for its proven architecture, reliability, data integrity, robust feature set, extensibility, and the dedication of the open source community behind the software to consistently deliver performant and innovative solutions. PostgreSQL runs on [all major operating systems](https://www.postgresql.org/download/), has been [ACID](https://en.wikipedia.org/wiki/ACID)-compliant since 2001, and has powerful add-ons such as the popular [PostGIS](https://postgis.net/) geospatial database extender. It is no surprise that PostgreSQL has become the open source relational database of choice for many people and organisations.

[Getting started](https://www.postgresql.org/docs/current/tutorial.html) with using PostgreSQL has never been easier - pick a project you want to build, and let PostgreSQL safely and robustly store your data.

PostgreSQL comes with [many features](https://www.postgresql.org/about/featurematrix/) aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being [free and open source](https://www.postgresql.org/about/license/), PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, even write code from [different programming languages](https://www.postgresql.org/docs/current/xplang.html) without recompiling your database!

PostgreSQL tries to conform with the [SQL standard](https://www.postgresql.org/docs/current/features.html) where such conformance does not contradict traditional features or could lead to poor architectural decisions. Many of the features required by the SQL standard are supported, though sometimes with slightly differing syntax or function. Further moves towards conformance can be expected over time. As of the version 12 release in October 2019, PostgreSQL conforms to at least 160 of the 179 mandatory features for SQL:2016 Core conformance. As of this writing, no relational database meets full conformance with this standard.

Below is an inexhaustive list of various features found in PostgreSQL, with more being added in every [major release](https://www.postgresql.org/developer/roadmap/):

Data Types

* Primitives: Integer, Numeric, String, Boolean
* Structured: Date/Time, Array, Range, UUID
* Document: JSON/JSONB, XML, Key-value (Hstore)
* Geometry: Point, Line, Circle, Polygon
* Customizations: Composite, Custom Types
* Data Integrity
* UNIQUE, NOT NULL
* Primary Keys
* Foreign Keys
* Exclusion Constraints
* Explicit Locks, Advisory Locks
* Concurrency, Performance
* Indexing: B-tree, Multicolumn, Expressions, Partial
* Advanced Indexing: GiST, SP-Gist, KNN Gist, GIN, BRIN, Covering indexes, Bloom filters
* Sophisticated query planner / optimizer, index-only scans, multicolumn statistics
* Transactions, Nested Transactions (via savepoints)
* Multi-Version concurrency Control (MVCC)
* Parallelization of read queries and building B-tree indexes
* Table partitioning
* All transaction isolation levels defined in the SQL standard, including Serializable
* Just-in-time (JIT) compilation of expressions
* Reliability, Disaster Recovery
* Write-ahead Logging (WAL)
* Replication: Asynchronous, Synchronous, Logical
* Point-in-time-recovery (PITR), active standbys
* Tablespaces
* Security
* Authentication: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, Certificate, and more
* Robust access-control system
* Column and row-level security
* Multi-factor authentication with certificates and an additional method
* Extensibility
* Stored functions and procedures
* Procedural Languages: PL/PGSQL, Perl, Python (and many more)
* SQL/JSON path expressions
* Foreign data wrappers: connect to other databases or streams with a standard SQL interface
* Customizable storage interface for tables
* Many extensions that provide additional functionality, including PostGIS
* Internationalisation, Text Search
* Support for international character sets, e.g. through ICU collations
* Case-insensitive and accent-insensitive collations
* Full-text search
* There are many more features that you can discover in the PostgreSQL [documentation](https://www.postgresql.org/docs/). Additionally, PostgreSQL is highly extensible: many features, such as indexes, have defined APIs so that you can build out with PostgreSQL to solve your challenges.
* PostgreSQL has been proven to be highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate. There are active PostgreSQL clusters in production environments that manage many terabytes of data, and specialized systems that manage petabytes.

RDBMS Terminology:

|  |  |
| --- | --- |
| Database | A database is a collection of tables, with related data. |
| Table | A table is a matrix with data. A table in a database looks like a simple spreadsheet. |
| Column | One column (data element) contains data of one and the same kind, for example the column postcode. |
| Row | A row ( tuple, entry or record) is a group of related data, for example the data of one subscription |
| Redundancy | Storing data twice, redundantly to make the system faster |
| Primary Key | A primary key is unique. A key value can not occur twice in one table. With a key, you can only find one row. |
| Foreign Key | A foreign key is the linking pin between two tables. |
| Compound Key | A compound key (composite key) is a key that consists of multiple columns, because one column is not sufficiently unique |
| Index | An index in a database resembles an index at the back of a book. |
| Referential Integrity | Referential Integrity makes sure that a foreign key value always points to an existing row. |

**Table 3: Terms used in Relational Database**

A Relational DataBase Management System ([[4]](https://paperpile.com/c/Rff4HW/tyLF)RDBMS) is a software that :

* Enables you to implement a database with tables, columns and indexes.
* Guarantees the Referential Integrity between rows of various tables.
* Updates the indexes automatically.
* Interprets an SQL query and combines information from various tables.

**Spring Boot**: Spring Boot is an open source Java-based framework used to create a Micro Service. It is developed by Pivotal Team. It is easy to create a stand-alone and production ready spring applications using Spring Boot. Spring Boot contains a comprehensive infrastructure support for developing a micro service and enables you to develop enterprise-ready applications that you can “just run”

Spring Boot is a Framework from “The Spring Team” to ease the bootstrapping and development of new Spring Applications. It provides defaults for code and annotation configuration to quick start new Spring projects within no time. It follows “Opinionated Defaults Configuration” Approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

Advantages of [[5]](https://paperpile.com/c/Rff4HW/XBQg)Spring Boot:

* It is very easy to develop Spring Based applications with Java.
* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
* It provides Embedded HTTP servers like Tomcat. to develop and test our web applications very easily.

Maven: Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, [[6]](https://paperpile.com/c/Rff4HW/51qV)Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of the build related tasks.

**3. REQUIREMENTS AND ANALYSIS**

**3.1. Problem Definition**

Digital Locker is a web based application which gives the user access to upload his/her documents. So the online locker first require a authentication page because security is primary objective here as it’s pertaining to the documents which are going to be upload by user. Authentication page gives the option to login to account using registered email and password. The user is allowed to pass authentication only if he is a registered user with valid email and password, therefore if user is not registered on online locker he/she will have to get registered to pass the authentication. Upon the successfully attempt to login. User will be routed to his profile dashboard.

Dashboard feature aims to shows the previously uploaded items to user. They’ll appear in descending order such that the latest one appears at the top/first. The online lockers also aims to have download option for registered users which allows them to download the uploaded file whenever they need.

Locker also aims to have delete feature as often times user may want to delete the uploaded items.

The problem can be broken into sub problems as below:

* **Login**
* **Registration**
* **Upload**
* **Delete**

**3.2. Requirement Specification**

Requirements of the System: In context to the requirement of system. User need to have a valid email address and internet connection to be able to access the Digital locker, Since locker is intended to serve users on an online platform users will requirement the internet connection. Another requirement is that user need to be registered on portal which is one of the important aspect.

We have already mentioned the features that possess in the proposed system and how actions will take place on the system.

Let us understand the operation of exiting system and problems of the existing system now-

In the present system we have to bear the effort of carrying documents at every place. Often times we end up with bulks of document, Important Certificate which we don’t need eventually

This wastes lot of time from customers perspective which is a huge disadvantage of a current system

Some of the problems are listed below which are associated with existing system.

* Effort
* Efficiency
* Management
* Durability

**3.2 Planning and Scheduling**

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and then arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to.

* Break down the project tasks into smaller, manageable form
* Find out various tasks and correlate them
* Estimate time frame required for each task
* Divide time into work-units
* Assign adequate number of work-units for each task
* Calculate total time required for the project from start to finish

PERT- PERT (Program Evaluation and Review Technique) is one of the successful method of project evaluation. [[7]](https://paperpile.com/c/Rff4HW/8hoU)PERT was initially created by the US Navy in the late 1950s. At the core, PERT is all about management probabilities. Therefore, PERT involves in many simple statistical methods as well.

Same as most of other estimation techniques, PERT also breaks down the tasks into detailed activities.

In PERT a node represents each event. The activities are represented as arrows and they are drawn from one event to another, based on the sequence

Next, the Earliest Time (TE) and the Latest Time (TL) are figured for each activity and identify the slack time for each activity.

Slack time- is the amount of time a task can be delayed without impacting the completion date of your project. Also known as float. The slack time is calculated as: Slack Time = Latest Time-Earliest Time

The Three Chances

There are three estimation times involved in PERT; Optimistic Time Estimate (TOPT), Most Likely Time Estimate (TLIKELY), and Pessimistic Time Estimate (TPESS).

In PERT, these three estimate times are derived for each activity. This way, a range of time is given for each activity with the most probable value, TLIKELY.

Following are further details on each estimate:

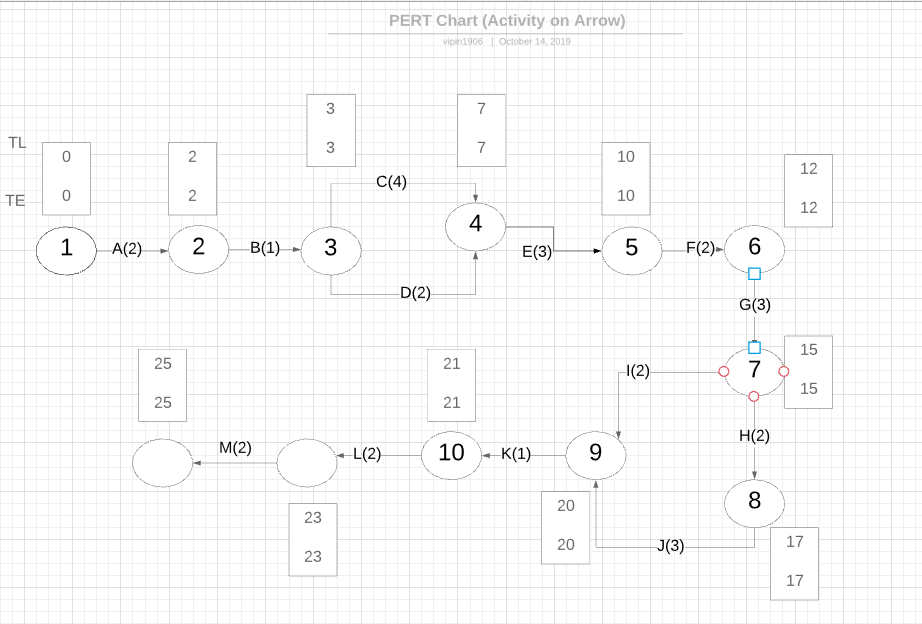
1. TOPT: This is the fastest time an activity can be completed. For this, the assumption is made that all the necessary resources are available and all predecessor activities are completed as planned.

2. TLIKELY: Most of the times, project managers are asked only to submit one estimate. In that case, this is the estimate that goes to the upper management.

3. TPESS: This is the maximum time required to complete an activity. In this case, it is assumed that many things go wrong related to the activity. A lot of rework and resource unavailability are assumed when this estimation is derived.

|  |  |  |  |
| --- | --- | --- | --- |
| Alias | Name of Actives | Predecessor/dependency(Alias) | TOPT(optimistic time in days) |
| A | Installation of Software’s | na | 2 |
| B | API to register user | A | 1 |
| C | API to login | B | 4 |
| D | UI components to register and login | B | 2 |
| E | API to parse AccessToken from request | DC | 3 |
| F | UI component to list file with icons | E | 2 |
| G | API to list all uploaded files | F | 3 |
| H | API to upload file to server | G | 2 |
| I | API/UI to download a file | G | 2 |
| J | Integrate UI for upload feature | H | 3 |
| K | API to delete file | IJ | 1 |
| L | API/UI to logout | K | 2 |
| M | Code Review /testing | L | 2 |

**Table 4: Activities**



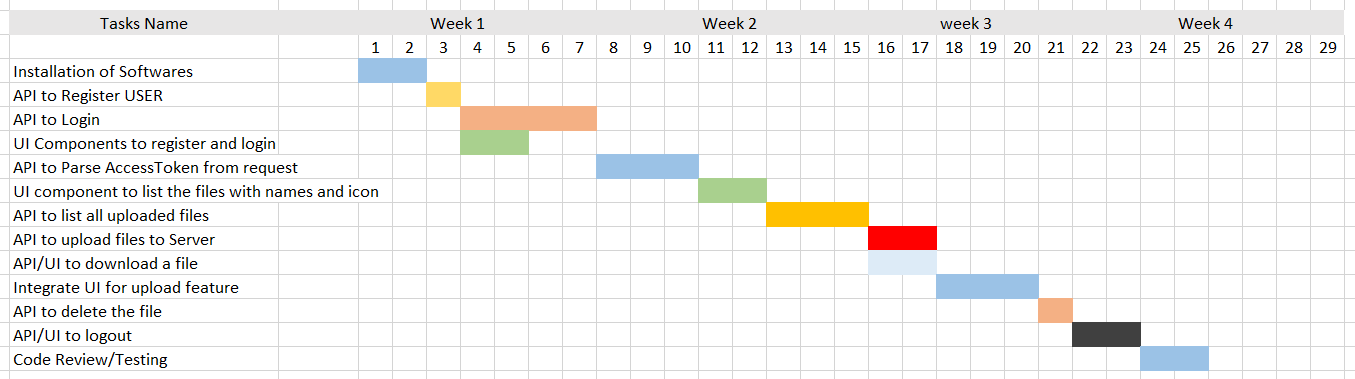
**Figure 1: Pert Chart**

GANT CHART- Gantt chart is a type of a bar chart that is used for illustrating project schedules. [[8]](https://paperpile.com/c/Rff4HW/Qe9F)Gantt charts can be used in any projects that involve effort, resources, milestones and deliveries. Gantt charts, the project manager can keep a track of the individual tasks as well as of the overall project progression.

In addition to tracking the progression of the tasks, Gantt charts can also be used for tracking the utilization of the resources in the project. These resources can be human resources as well as materials used.

Advantages & Disadvantages:

* The ability to grasp the overall status of a project and its tasks at once is the key advantage in using a Gantt chart tool. Therefore, upper management or the sponsors of the project can make informed decisions just by looking at the Gantt chart tool.
* The software-based Gantt charts are able to show the task dependencies in a project schedule. This helps to identify and maintain the critical path of a project schedule.
* Gantt chart tools can be used as the single entity for managing small projects. For small projects, no other documentation may be required; but for large projects, the Gantt chart tool should be supported by other means of documentation.
* For large projects, the information displayed in Gantt charts may not be sufficient for decision making.
* Although Gantt charts accurately represent the cost, time and scope aspects of a project, it does not elaborate on the project size or size of the work elements. Therefore, the magnitude of constraints and issues can be easily misunderstood.



**Figure 2: Gant Chart**

**3.4. Software and Hardware Requirements**

**3.4.1. Hardware Requirements**

* P4 Processor (CPU) with 2 gigahertz (GHz) frequency or above
* A minimum of 2 GB of RAM
* Monitor Resolution 1024 X 768 or higher
* A minimum of 20 GB of available space on the hard disk
* Internet Connection Broadband (high-speed) Internet connection with a speed of 4 Mbps or higher
* Keyboard and a Microsoft Mouse or some other compatible pointing device

**3.4.2. Operating System**

* Windows 7, Windows 8 or Windows 10
* Mac OSX 10.8, 10.9, 10.10 or 10.11

**3.4.3. Browsers**

* Chrome\* 36+
* Edge\* 20+
* Mozilla Firefox 31+
* Internet Explorer 11+ (Windows only)
* Safari 6+ (MacOS only)

**3.4.4. Software Requirements**

* JDK 1.8
* Maven Build tool
* Node JS with NPM installer
* Git / Github
* Eclipse / Intellij for development
* Spring Boot Rest APIs
* Angular 2
* PostgreSQL

**3.5. Preliminary Product Description**

**3.5.1. Objectives**

* Eliminate need for the users to maintain hard copy of important documents.
* Eliminate need for the users to produce (in hard format) documents, while applying for services.
* Provide secure and consented access of documents to user agencies.
* Reduce administrative burden, service fulfillment time, and costs by enabling paperless transactions.
* Ensure all the documents issued to the users are available to him/her anywhere anytime, in a standard format which can be shared with any other department.
* Provide an open, interoperable, multi-provider architecture to ensure departments and states have flexibility to use best document repository for their purposes.
* Provide an architecture that can support well-structured future documents as well as a mechanism to digitize older documents that may not have machine readable formats.
* Provide a default portal and mobile application for residents to view their documents in a consolidated way.

**3.5.2. Functions of Application System:**

The functions of Application can be decomposed into multiple modules to have more clarity with respect to each function which is being part of the application.

* Login: Proposed system will have a login where user needs to enter email and password to get access of his account.
* Registration: Proposes system will have registration section as well only for those who have not been registered on application.
* Upload : This feature will only be accessible to the registered user after successful login.
* Delete: A registered user will be allowed to delete the previously upload document.

**3.6. Conceptual Models**

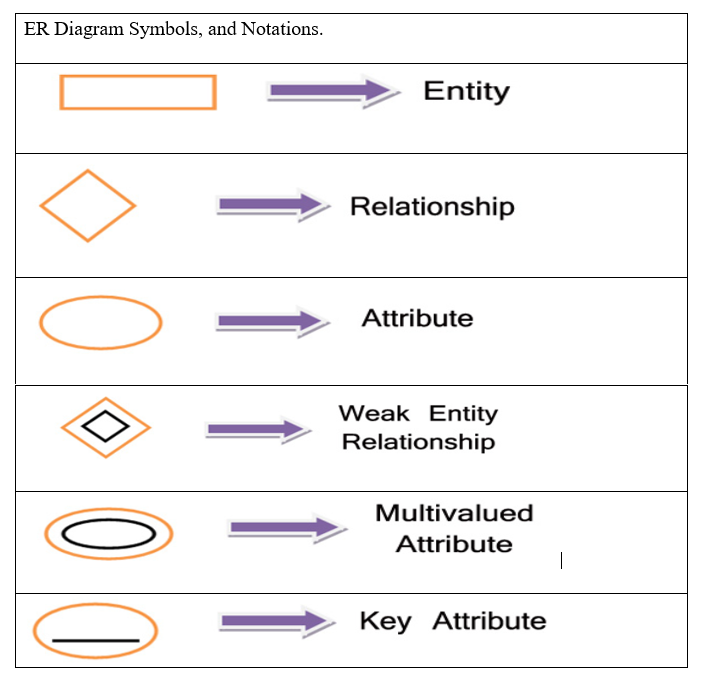
**3.6.1. ER Diagram( Entity Relation Diagram):**

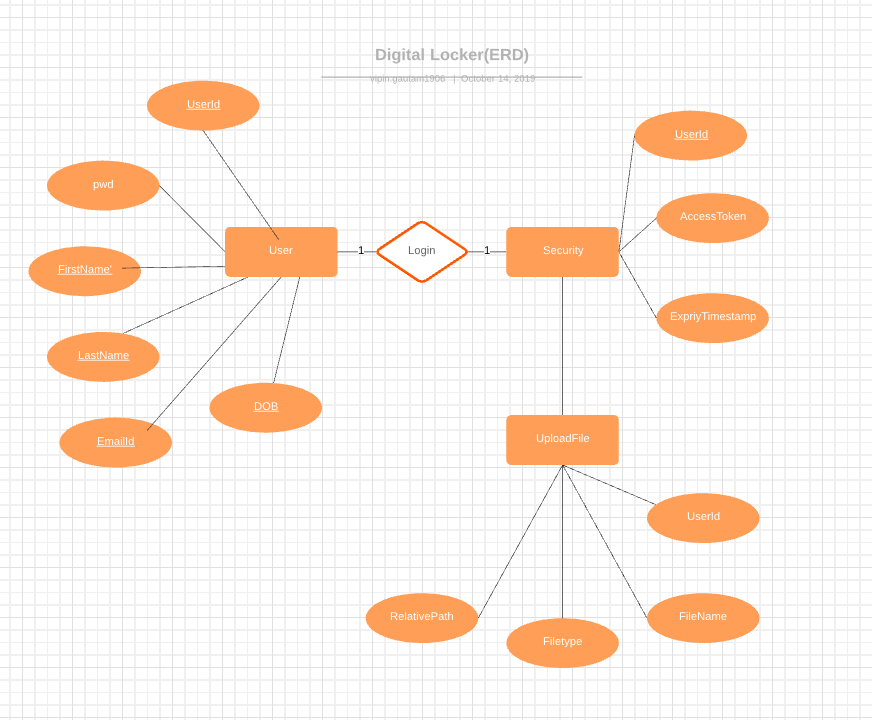
The ER or ([[8,9]](https://paperpile.com/c/Rff4HW/Qe9F+dDWq)Entity Relational Model) is a high-level conceptual data model diagram. Entity-Relation model is based on the notion of real-world entities and the relationship between them. ER modeling helps you to analyze data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modeling before implementing your database.

E-R diagrams are used to model real-world objects like a person, a car, a company etc. and the relation between these real-world objects. An e-r diagram has following features:

* E-R diagrams are used to represent E-R model in a database, which makes them easy to be converted into relations (tables).
* E-R diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
* E-R diagrams require no technical knowledge & no hardware support.
* These diagrams are very easy to understand and easy to create even by a naive user.
* It gives a standard solution of visualizing the data logically.

**3.6.2. ER Diagram Symbols, and Notations:**

**Table 5: Notations**



**Figure 3: ER Diagram**

**3.6.3. DFD( Data Flow Diagram):**

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. Specific operations based on the data can be represented by a flowchart.

DFD Components:

DFD can represent Source, destination, storage and flow of data using the following set of components -



**Figure 4: Notations in Data Flow Diagram**

Entities - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.

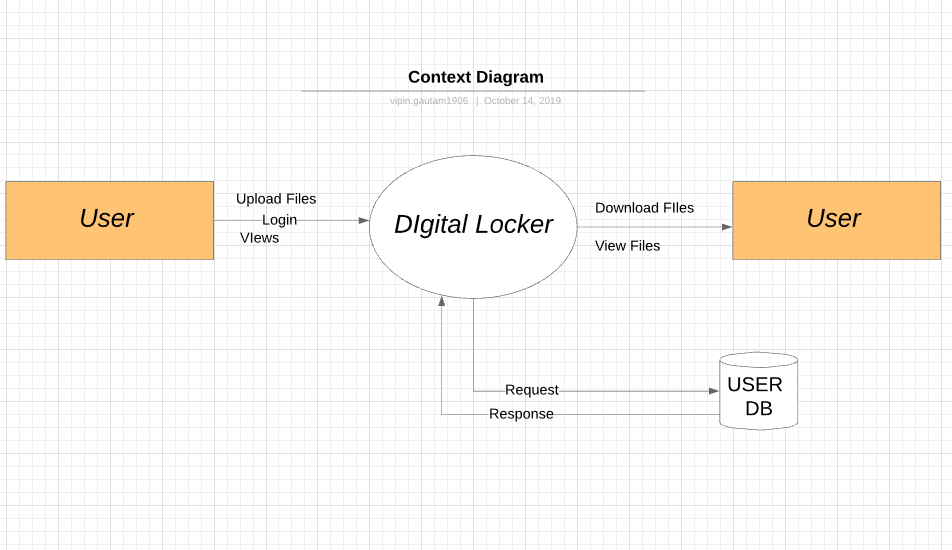
Process - Activities and action taken on the data are represented by Circle or Round-edged rectangles.

Data Storage - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

Data Flow - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

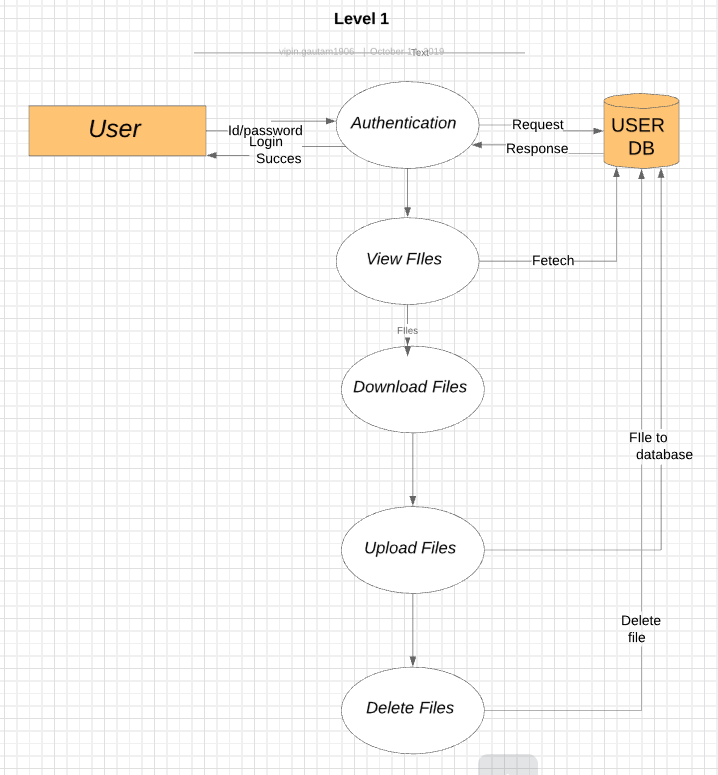
Levels of DFD

Level 0 - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



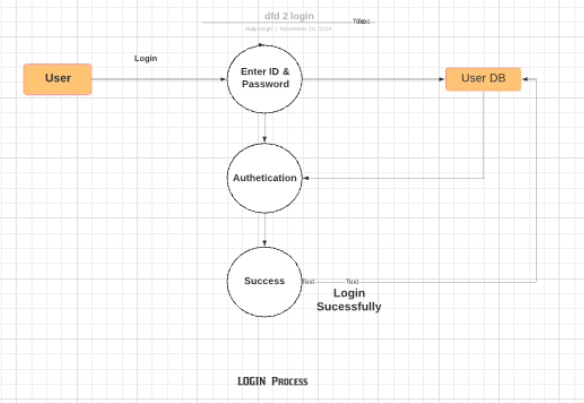
**Figure 5: Context Level DFD**

Level 1 - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.

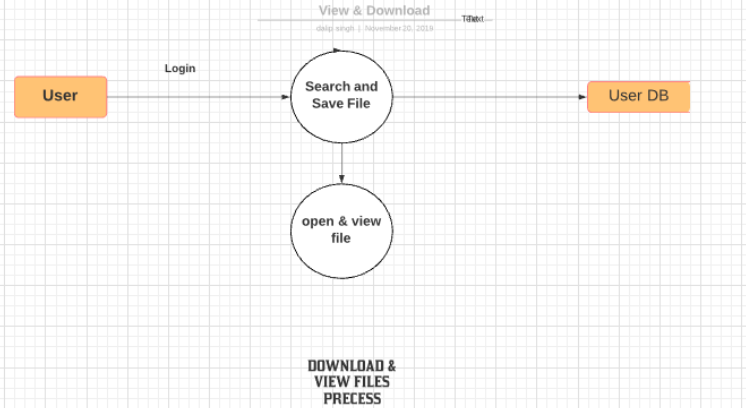


**Figure 6: Level 1 DFD**

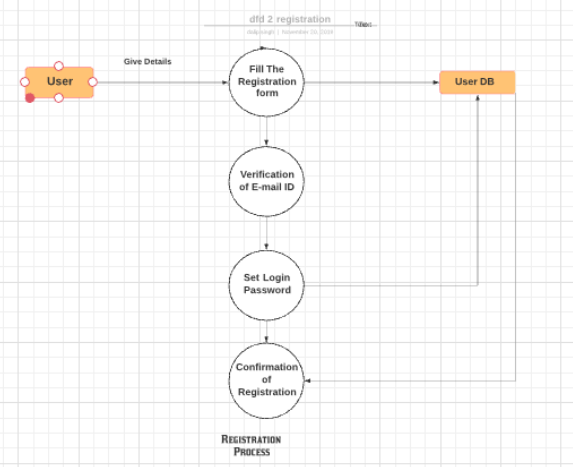
Level 2 - At this level, DFD shows how data flows inside the modules mentioned in Level 1 Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.



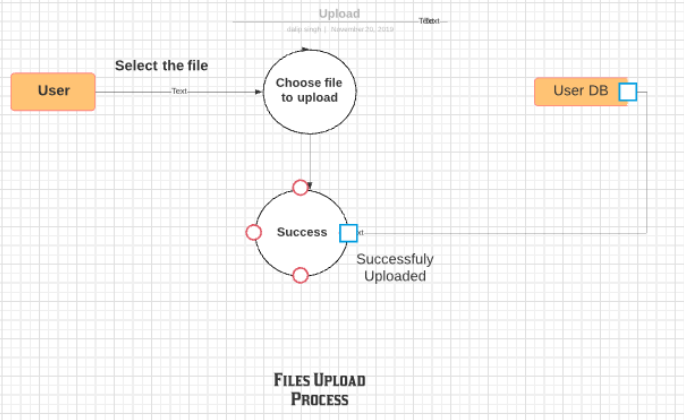
**Figure 7: Level 2 DFD**



**Figure 8: Level 2 DFD**



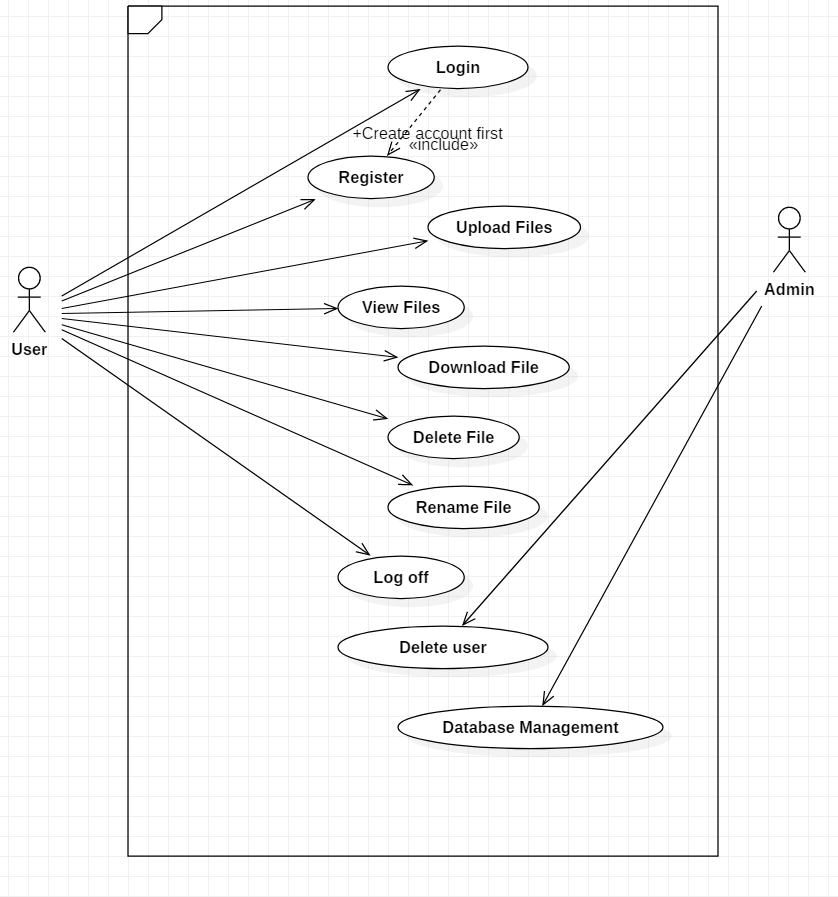
**Figure 9: Level 2 DFD**



**Figure 10: Level 2 DFD**

**3.6.4. Use Case Diagram:**

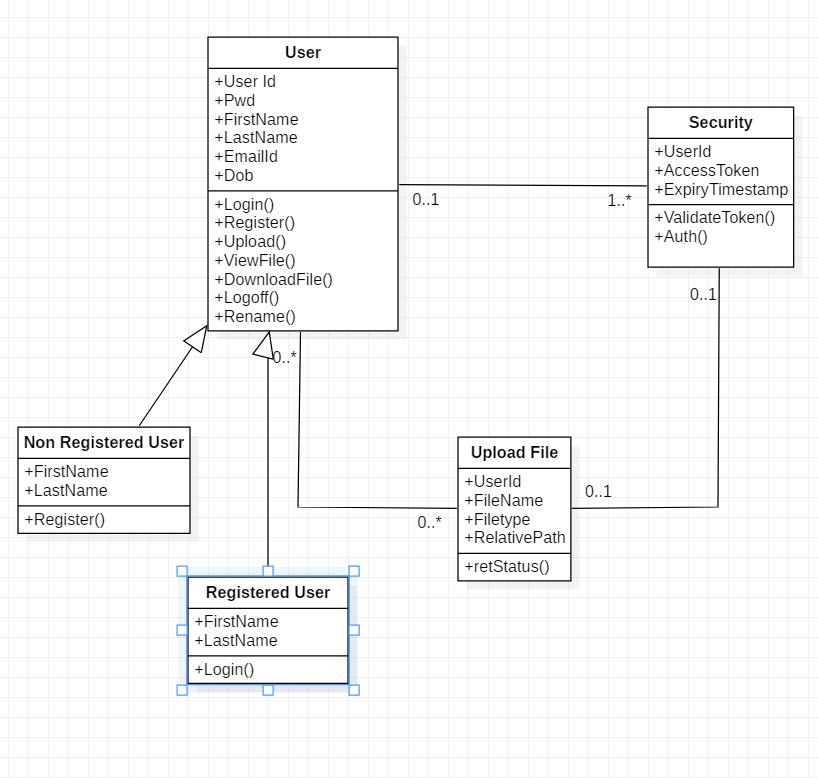
A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system



**Figure 11: Use Case Diagram**

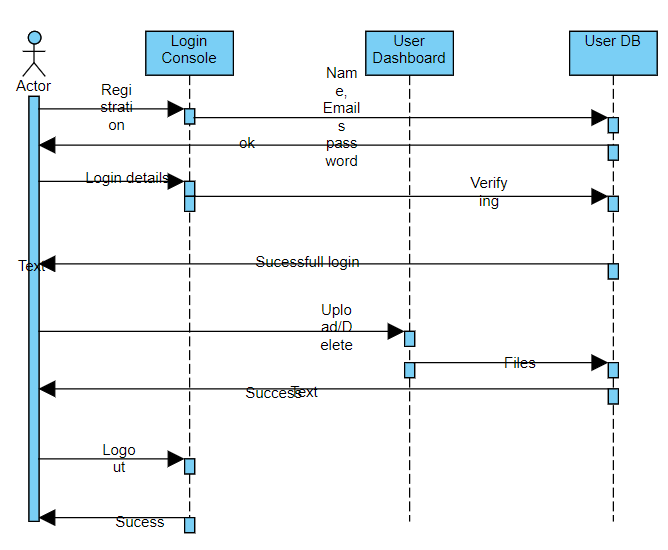
**3.6.5. Class Diagram:**

Class diagrams are the main building blocks of every object oriented methods. The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, so as the class diagram has appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in its context. It describes various kinds of objects and the static relationship in between them.



**Figure 12: Class Diagram**

**3.6.6. Sequence Diagram:**



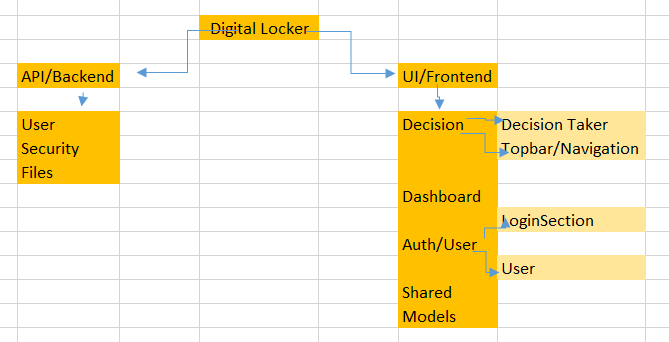
**Figure 13: Sequence Diagram**

**4. SYSTEM DESIGN**

**4.1. Basic Modules**

We’ve used the divide and conquer technique here before designing any system it is very important to have a abstract structure in mind as in what needs to be done and how we can approach the system.

In our project our Aim is to design Digital Locker. Mostly all application comprises frontend and backend. Frontend is the content that user see what something with which he/she can interact whereas Backend is something who is responsible for handling all those request in backend which user is not able to see but it’s core of any web application. So why not to have a complete different module for both of these. We have followed the same approach first and foremost we have divided the entire application in two parts UI(Frontend) and API(application programing interface)backend. We shall discuss both in more detail. We will layout the abstract structure of the built application.



**Figure 14: App Tree**

We will start the discussion on each module now.

UI Module is tightly held by the angular tree. Which we have divided in 4 major modules. Shared and Models are the parts of the entire UI module the apparent reason is that they hold some services or common data which is being used by either users on dashboard or may be to list data on some other part. For the simplicity we can call them component as well. Each component in the able abstraction has their own role and somewhat related to other component as well thus we can say they all are bounded by the UI component as a whole.

Lets start our discussion with Decision Component.

Decision Module

The Decision component is further divided in two parts. Decision Taker and Topbar, Decision Taker is solely responsible as to which component will get trigger on each user interaction whereas Topbar is just the depiction of it which lets the user interaction which component he may choose to go.

**4.2. Data Design**

**4.2.1. Schema Design:**

Database schema is the skeleton of database. It is designed when the database doesn't exist at all. Once the database is operational, it is very difficult to make any changes to it. A database schema does not contain any data or information.

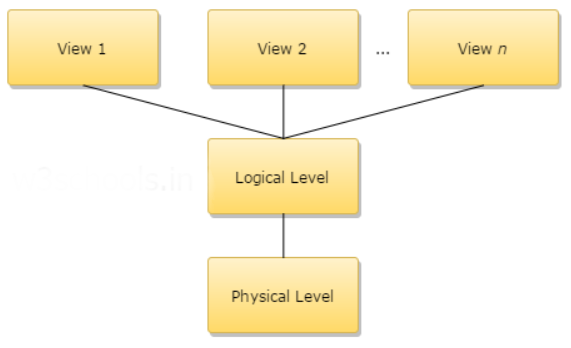
A database instance is a state of operational database with data at any given time. It contains a snapshot of the database. Database instances tend to change with time. A DBMS ensures that its every instance (state) is in a valid state, by diligently following all the validations, constraints, and conditions that the database designers have imposed.

A database schema can be divided broadly into two categories

* Physical Database Schema: This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
* Logical Database Schema: This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

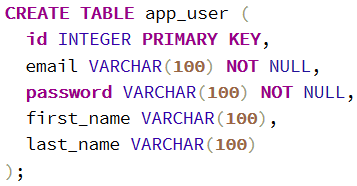
A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

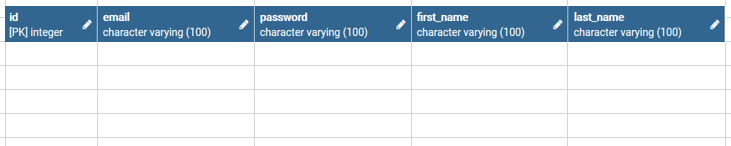
A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams.



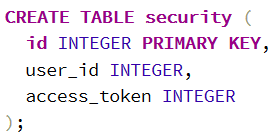
**Figure 15: Schema Architecture**

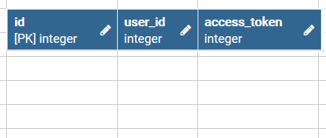
App User Table-



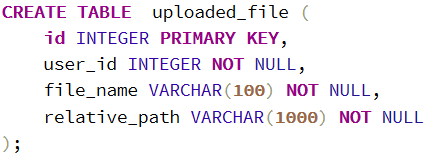


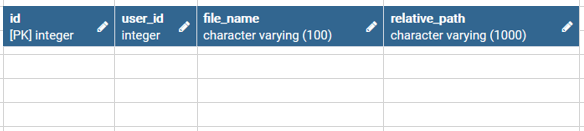
Security Table-





Upload File Table-



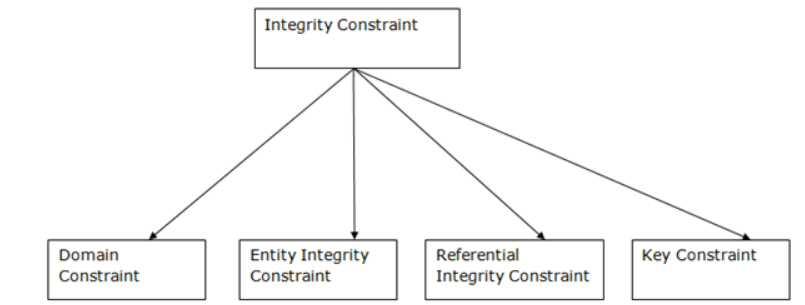


**4.2.2. Data Integrity and Constraints:**

Integrity Constraints

* Integrity constraints are a set of rules. It is used to maintain the quality of information.
* Integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected.
* Thus, integrity constraint is used to guard against accidental damage to the database.

## Types of Integrity Constraint



**Figure 16: Types of Constraints**

Explanation about the constraints and their implementation in our project:

1. Domain constraints

* Domain constraints can be defined as the definition of a valid set of values for an attribute.
* The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.
* In our project we have ensured that all the attributes are completely enforced with domain constraint. Here if user tries to enter any value which is not permissible in domain won’t be allowed. We have given snapshot of the logical schema as well physical schema for that.

### 2. Entity integrity constraints

* The entity integrity constraint states that primary key value can't be null.
* This is because the primary key value is used to identify individual rows in relation and if the primary key has a null value, then we can't identify those rows.
* A table can contain a null value other than the primary key field.

In app\_user, security and upload\_file table all 3 satisfies entity integrity constraint because we have considered Id as primary key.

That clearly means under no circumstances it can be kept null.

### 3. Referential Integrity Constraints

* A referential integrity constraint is specified between two tables.
* In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.

### 4. Key constraints

* Keys are the entity set that is used to identify an entity within its entity set uniquely.
* An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

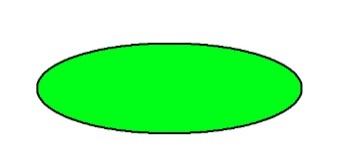
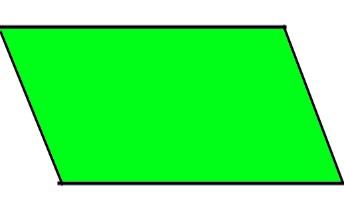
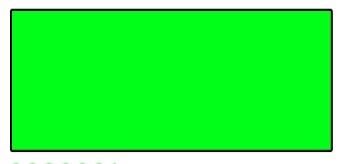
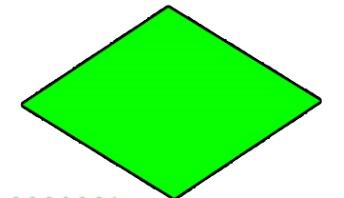
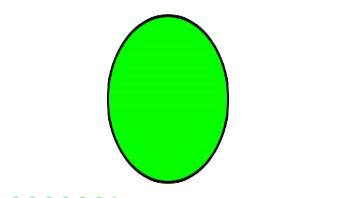
In all the relations there exist on key and which holds the unique value for relation identification.

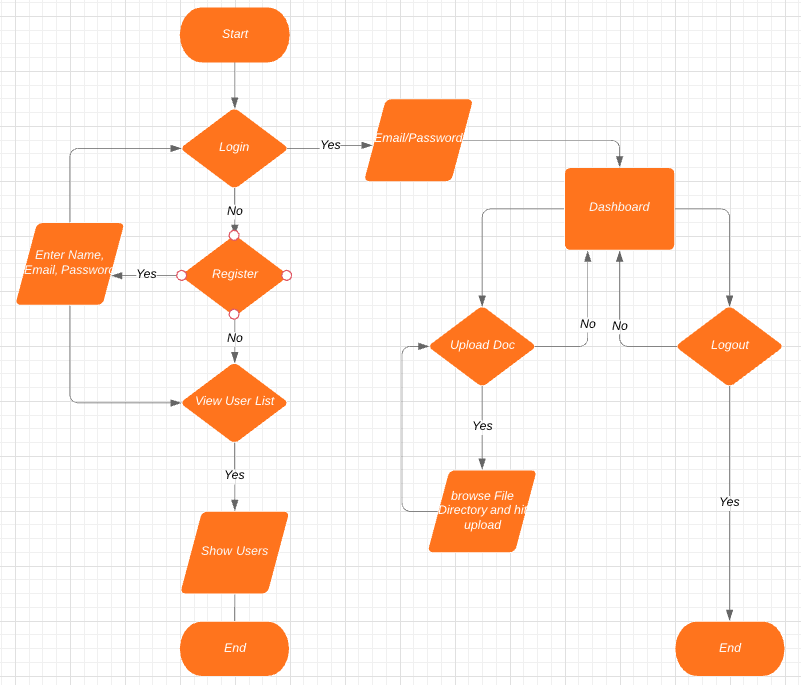
**4.3. Procedural Design**

**4.3.1. Logic Diagrams**

Flowchart is a graphical representation of an algorithm. It makes use of symbols which are connected among them to indicate the flow of information and processing.  
The process of drawing a flowchart for an algorithm is known as “flowcharting”.

Basic Symbols used in Flowchart Designs

1. Terminal: The oval symbol indicates Start, Stop and Halt in a program’s logic flow. A pause/halt is generally used in a program logic under some error conditions. Terminal is the first and last symbols in the flowchart.  
   
2. Input/Output: A parallelogram denotes any function of input/output type. Program instructions that take input from input devices and display output on output devices are indicated with parallelogram in a flowchart.  
   
3. Processing: A box represents arithmetic instructions. All arithmetic processes such as adding, subtracting, multiplication and division are indicated by action or process symbol.  
   
4. Decision Diamond symbol represents a decision point. Decision based operations such as yes/no question or true/false are indicated by diamond in flowchart.  
   
5. Connectors: Whenever flowchart becomes complex or it spreads over more than one page, it is useful to use connectors to avoid any confusions. It is represented by a circle.  
   
6. Flow lines: Flow lines indicate the exact sequence in which instructions are executed. Arrows represent the direction of flow of control and relationship among different symbols of flowchart.



**Figure 17: Control Flow Chart Diagram**

**4.3.2. Data Structures:**

Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have some data which has, player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is of integer data type.

We can organize this data as a record like Player record, which will have both player's name and age in it. Now we can collect and store player's records in a file or database as a data structure. For example: "Dhoni" 30, "Gambhir" 31, "Sehwag" 33

If you are aware of Object Oriented programming concepts, then a class also does the same thing, it collects different type of data under one single entity. The only difference being, data structures provides for techniques to access and manipulate data efficiently.

In simple language, Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

In our project we have used Array List.

Let us dive deeper and explain this Data Structure used in course project.

Array List-

* ArrayList is a part of [collection framework](https://www.geeksforgeeks.org/collections-in-java-2/) and is present in java.util package. It provides us dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed.
* ArrayList inherits AbstractList class and implements List interface.
* ArrayList is initialized by a size, however the size can increase if collection grows or shrunk if objects are removed from the collection.
* Java ArrayList allows us to randomly access the list.
* ArrayList can not be used for primitive types, like int, char, etc. We need a wrapper class for such cases (see [this](https://www.geeksforgeeks.org/array-vs-arraylist-in-java/) for details).
* ArrayList in Java can be seen as similar to [vector in C++](http://quiz.geeksforgeeks.org/vector-sequence-containers-the-c-standard-template-library-stl-set-1/).

Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non synchronized.
* Java ArrayList allows random access because array works at the index basis.
* In Java ArrayList class, manipulation is slow because a lot of shifting needs to occur if any element is removed from the array list.

declaration for java.util.ArrayList class.

public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable

**4.3.3. Algorithms Design**

What is Algorithm?

Consider how you use a computer in a typical day. For example, you start working on a report, and once you have completed a paragraph, you perform a spell check. You open up a spreadsheet application to do some financial projections to see if you can afford a new car loan. You use a web browser to search online for a kind of car you want to buy.

You may not think about this very consciously, but all of these operations performed by your computer consist of algorithms. An algorithm is a well-defined procedure that allows a computer to solve a problem. Another way to describe an algorithm is a sequence of unambiguous instructions. The use of the term 'unambiguous' indicates that there is no room for subjective interpretation. Every time you ask your computer to carry out the same algorithm, it will do it in exactly the same manner with the exact same result.

Consider the earlier examples again. Spell checking uses algorithms. Financial calculations use algorithms. A search engine uses algorithms. In fact, it is difficult to think of a task performed by your computer that does not use algorithms.

**Algorithms Devised In our project – Digital Locker.**

Main objective in digital locker is to give upload functionality to user after successful registration/login if he/she is not registered then we can say registration is mandatory process. So we need to check the Authenticity of each user who wishes to login so we should match each combination of user email Id and Password with the entries stored in our Database.

Let us explain the Algorithm we have made to follow Security/Authentication process.

**Security:**

Step1: Register with Name, Email Id and Password.

Step2: Login

Step3: If Combination of Email ID and Password matches with data in database.

Step4: Attach accessKey/Access Token

Step5: For each user access to secured URL check If accessKey exisits in System’

If True

Then Request will be processed to URL

Else

Access will be rejected;

Step6: follow step 2,3,4,5 for each user.

After successful login users allowed to view dashboard where they have access to uploaded files let see how this upload mechanism has been implemented.

**Upload:**

Step1: Click Upload Button.

Step2: Browse the file and select to upload

Step3: Server will select the file.

Step4: Server will get the multpart request.

Step5: Move the file to separate directory in server, user will have their corresponding files using relative path to user directory

**Delete:**

Step1: Select the file wish to delete.

Step2: Click Delete Button.

Step3: File removed from Server Directory.

Step: Entry from Database has been removed by server.

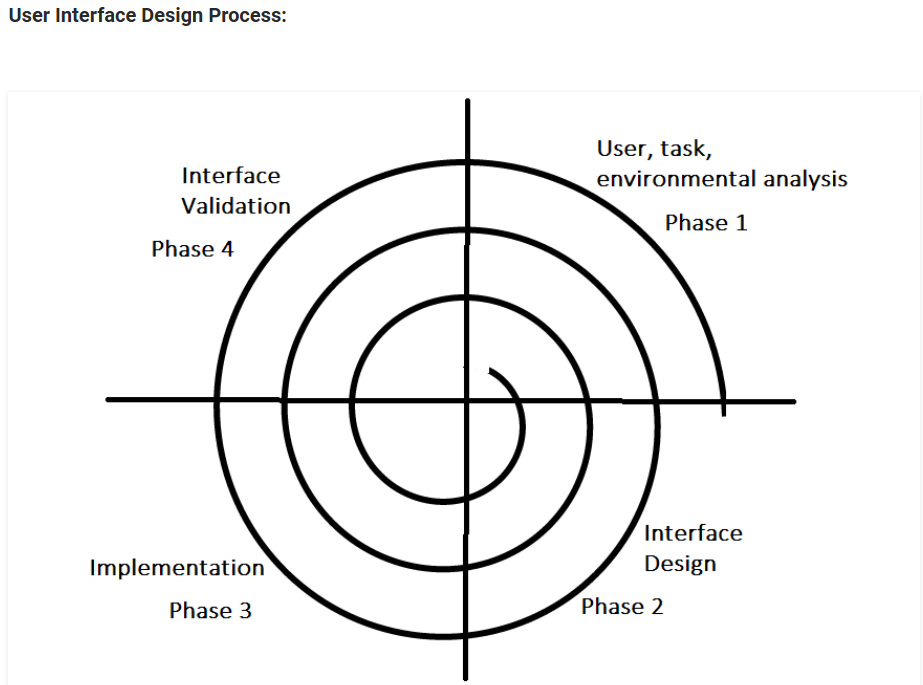
**4.4. User interface design**

User interface is the front-end application view to which user interacts in order to use the software. The software becomes more popular if its user interface is:

* Attractive
* Simple to use
* Responsive in short time
* Clear to understand
* Consistent on all interface screens

There are two types of User Interface:

1. Command Line Interface: Command Line Interface provides a command prompt, where the user types the command and feeds to the system. The user needs to remember the syntax of the command and its use.
2. Graphical User Interface: Graphical User Interface provides the simple interactive interface to interact with the system. GUI can be a combination of both hardware and software. Using GUI, user interprets the software.



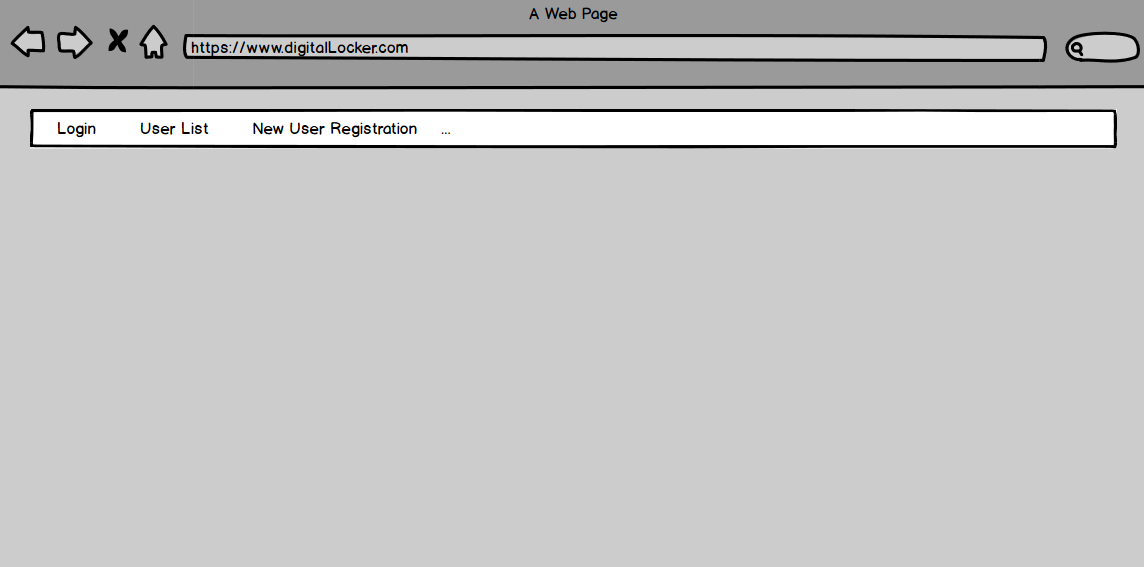
**Figure 18: Model for UI Design(Based on Waterfall Model)**

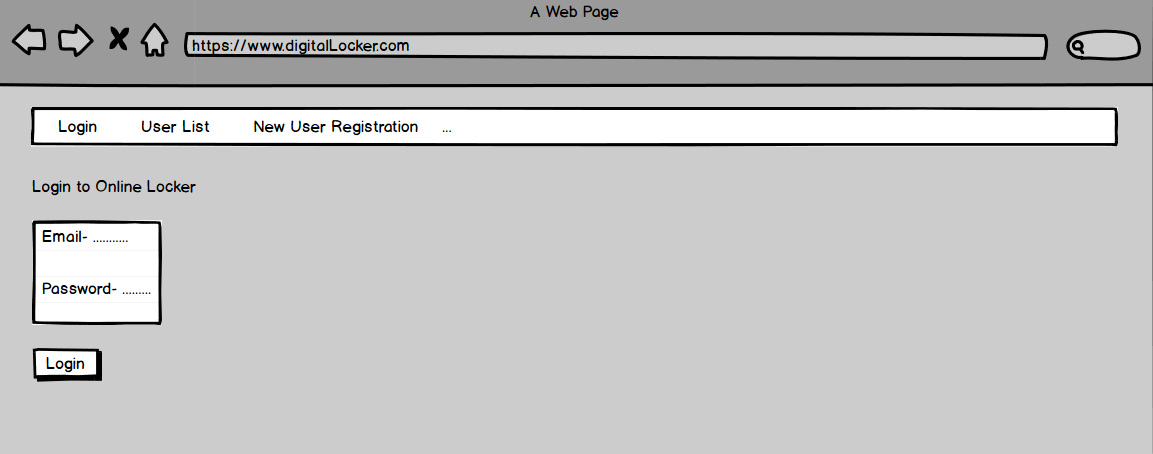
The analysis and design process of a user interface is iterative and can be represented by a spiral model. The analysis and design process of user interface consists of four framework activities.

1. User, task, environmental analysis, and modeling: Initially, the focus is based on the profile of users who will interact with the system, i.e. understanding, skill and knowledge, type of user, etc, based on the user’s profile users are made into categories. From each category requirements are gathered. Based on the requirements developer understand how to develop the interface. Once all the requirements are gathered a detailed analysis is conducted. In the analysis part, the tasks that the user performs to establish the goals of the system are identified, described and elaborated. The analysis of the user environment focuses on the physical work environment. Among the questions to be asked are:
   * Where will the interface be located physically?
   * Will the user be sitting, standing, or performing other tasks unrelated to the interface?
   * Does the interface hardware accommodate space, light, or noise constraints?
   * Are there special human factors considerations driven by environmental factors?
2. Interface Design: The goal of this phase is to define the set of interface objects and actions i.e. Control mechanisms that enable the user to perform desired tasks. Indicate how these control mechanisms affect the system. Specify the action sequence of tasks and subtasks, also called a user scenario. Indicate the state of the system when the user performs a particular task. Always follow the three golden rules stated by Theo Mandel. Design issues such as response time, command and action structure, error handling, and help facilities are considered as the design model is refined. This phase serves as the foundation for the implementation phase.
3. Interface construction and implementation: The implementation activity begins with the creation of prototype (model) that enables usage scenarios to be evaluated. As iterative design process continues a User Interface toolkit that allows the creation of windows, menus, device interaction, error messages, commands, and many other elements of an interactive environment can be used for completing the construction of an interface.

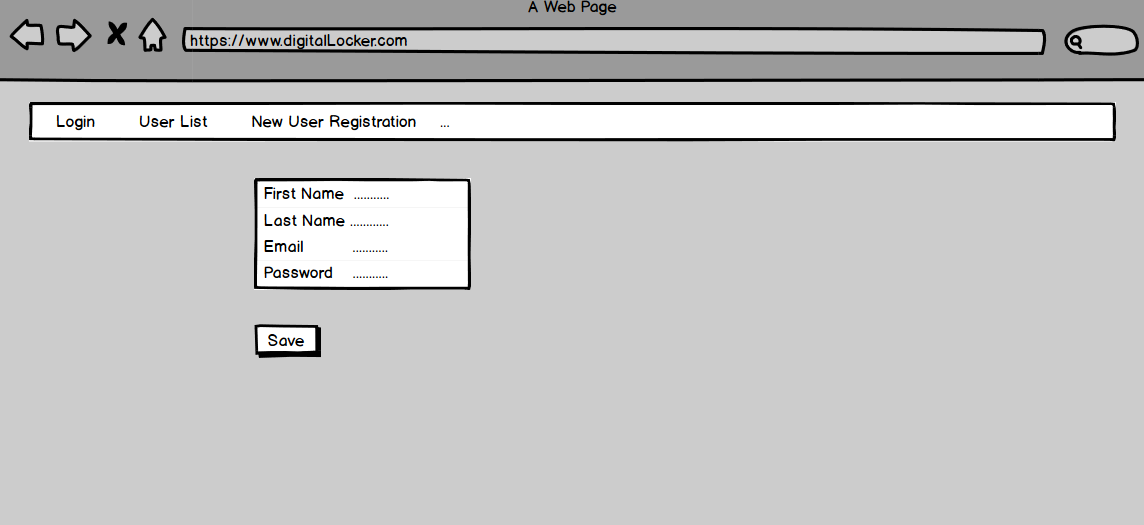
In our course project before implementing the application we’ve tried to map the component and user interfaces as how they should look when viewed by user.

In Digital Locker, our primary aim is to give full control to the user who will be using our application so the best way to give full control to user is to provide a navigation bar at the top.

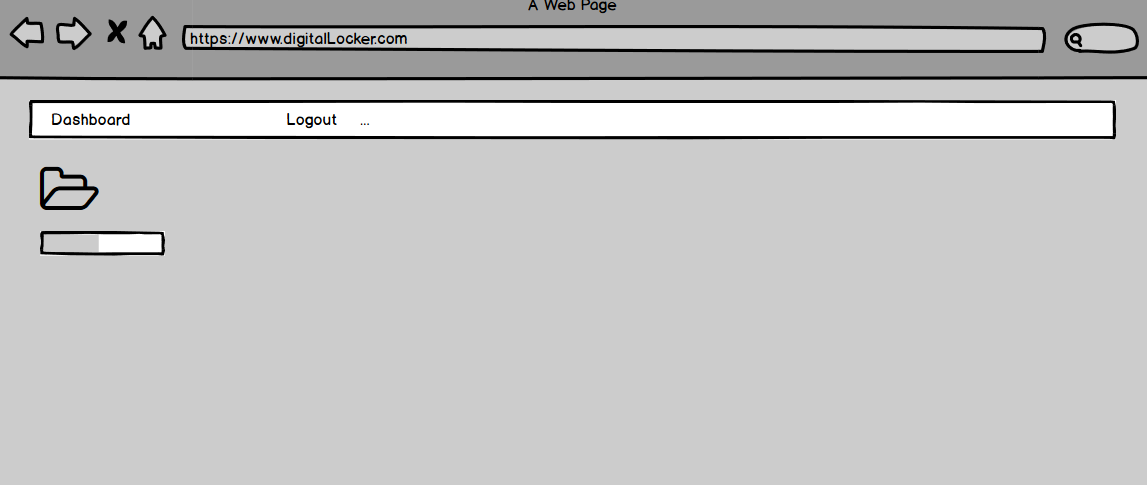
**Figure 18: Navigation Bar Sketch**



**Figure 19: Login Page Sketch**



**Figure 20: Registration Page Sketch**



**Figure 21: User Dashboard Sketch**

**4.5. Security Issues**

Security and [data protection](https://www.gasystems.com.au/security/data-protection/), are stringently regulated. Although the law struggles to keep up with the constant changes of an evolving digital world, there are regulations in force which demand certain standards from any business with an online component. Users across the globe expect their privacy to be taken seriously and modern commerce must reflect this wish.

**4.5.1. Real-time issues**

Form validation is required to prevent web form abuse by malicious users. Improper validation of form data is one of the main causes of security vulnerabilities. It exposes your website to attacks such as header injections, cross-site scripting, and SQL injections.

1. header injection attacks can be used to send email spam from your web server.
2. cross-site scripting may allow an attacker to post any data to web app.
3. SQL injection may corrupt your database backend.

Validation of form data is based on the data type you specify for each field when [creating a form](https://formsmarts.com/form-builder). For each data type, only a specific set of characters are allowed, and precise validation rules apply. During form submission, the user will be invited to modify any input fields found to contain invalid data. It is important that form owners choose the correct data type for each field. Selecting an incorrect data type will not pose a security risk, but it may prevent form users from entering legitimate data.

## 4.5.2. Data Validation

## Data validation is the process of ensuring that user input is clean, correct, and useful.

## Typical validation tasks are:

* Has the user filled in all required fields?
* Has the user entered a valid date?
* Has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

Digital Locker has two core sections which are vulnerable to security threats we have used Angular based form validation to ensure that there is no spurious access to registered user databases.

Two sections are mentioned as below.

Registration Form and Login Form- In both no user is allowed to login without updating correct fields in the form. Thus bypassing the login and registration is impossible. Here every user who is not registered on database will have to go through the registration and when he tries to login he is assigned a access token so no two users can have this access token. In the Login section both Email id and Password are mandatory fields similarly in the registration form All fields are mandatory and we have also taken care of email id field as user must use the valid email id whilst creating his account on the platform. After the user is logged in the server is made such a way that it pull the directory of user who is logged in currently so no chance of data threat. This is implemented based on access token which is different for each user and created at the time of login so when user logout this access token is destroyed completely so if at a time 100 users are logged in then there will be 100 unique access token which will make sure they are working on separate directories of their own without any conflict of data.

Database has been designed with keeping the security constraints in mind so there is no chance of any type of anomaly in the Database as well.

**4.6. Test Cases Design**

Feature Outline: Credential manipulation inside application.

Scenario Outline: Login into the application:

Given: there exists a user with <email> and <password> in database

When: user performs login with <email> and <password>

Then: an accessToken will be generated for that user

Scenario Outline: Login fails for a user:

Given: there exists a user with <email> and <password>

When: user performs login with <email> and <randomPassword>

Then: login will fail telling email/password doesn't exists

Scenario Outline: Logout from the application:

Given: there exists an already generated <accessToken>

When: user performs logout with that <accessToken>

Then: security record from database for that token will be removed

Scenario Outline: Logout from the application:

Given: there exists an invalid <accessToken>

When: user performs logout with that <accessToken>

Then: logout fails telling it doesn't exists

Feature Outline: User manipulation inside application.

Scenario Outline: Creating a user:

Given: user create a user object by fillings its fields <email>,

<password>, <firstName>, <lastName>

When: user saves that object

Then: user object will be saved in database with a new id

Scenario Outline: Retrieving all users:

Given: there are two users in the system

When: user performs GET all users

Then: two users object will be returned

Feature Outline: File Manipulation inside application.

Scenario Outline: Uploading a file:

Given: there exists a user with <email> and <password> in database

When: user performs login with <email> and <password>

Then: <accessToken> will be generated for him for future requests.

Given: there exists a Multipart <newFile> to Upload

And: User adds the <accessToken> in request header "AccessToken"

When: User uploads the <newFile>

Then: file will be uploaded in server's temp folder

And: UploadedFile also be saved in database.

Scenario Outline: downloading a file:

Given: there exists a user with <email> and <password> in database

When: user performs login with <email> and <password>

Then: <accessToken> will be generated for him for future requests.

Given: there exists a UploadedFile with <id>

And: User adds the <accessToken> in request header "AccessToken"

When: User downloads the file with <id>

Then: blob file object will be saved to reponse

Scenario Outline: Deleting a file:

Given: there exists a user with <email> and <password> in database

When: user performs login with <email> and <password>

Then: <accessToken> will be generated for him for future requests.

Given: there exists a UploadedFile with <id>

And: User adds the <accessToken> in request header "AccessToken"

When: User deletes the file with <id>

Then: file will be removed from server's directory

And: UploadedFile will be removed from database

Scenario Outline: Retrieving all files:

Given: there exists a user with <email> and <password> in database

When: user performs login with <email> and <password>

Then: <accessToken> will be generated for him for future requests.

Given: there exists multiple UploadedFiles

And: User adds the <accessToken> in request header "AccessToken"

When: User performs GET all files

Then: Multiple UploadedFile list will be returned

**5. IMPLEMENTATION AND TESTING**

**5.1. Implementation Approaches**

Throughout the project we have used Structured Programming for implementation

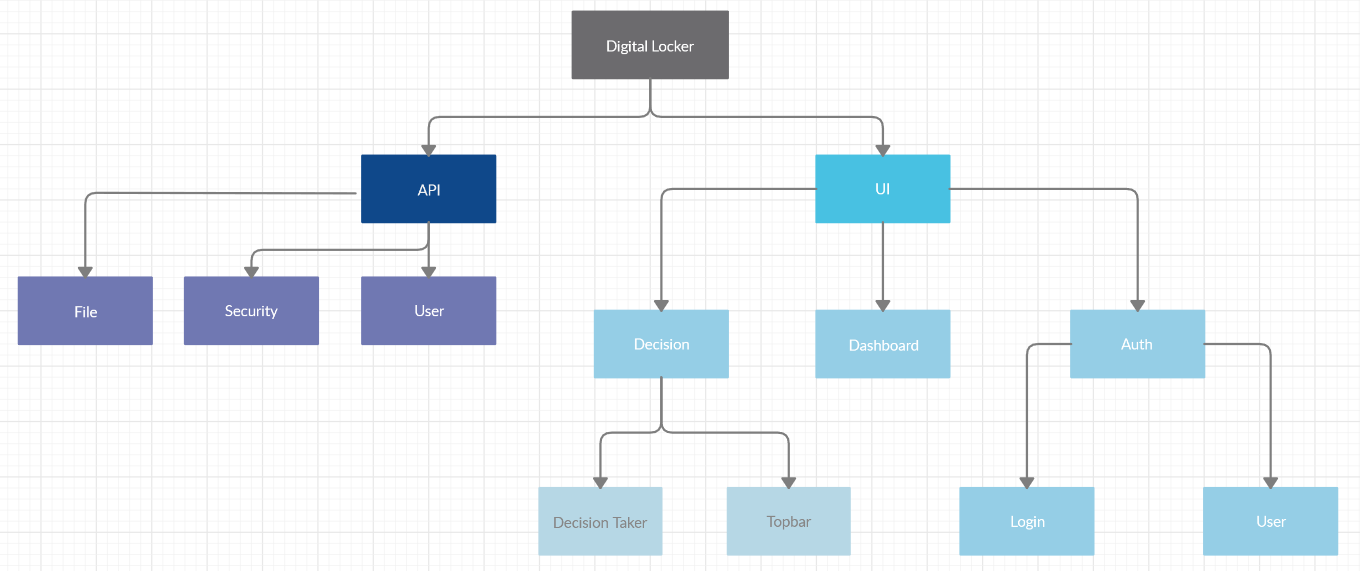
Let us explain more about structured programming.

In the process of coding, the lines of code keep multiplying, thus, size of the software increases. Gradually, it becomes next to impossible to remember the flow of program. If one forgets how software and its underlying programs, files, procedures are constructed it then becomes very difficult to share, debug and modify the program. The solution to this is structured programming. It encourages the developer to use subroutines and loops instead of using simple jumps in the code, thereby bringing clarity in the code and improving its efficiency Structured programming also helps programmer to reduce coding time and organize code properly.

Structured programming states how the program shall be coded. Structured programming uses three main concepts:

* Top-down analysis - A software is always made to perform some rational work. This rational work is known as problem in the software parlance. Thus it is very important that we understand how to solve the problem. Under top-down analysis, the problem is broken down into small pieces where each one has some significance. Each problem is individually solved and steps are clearly stated about how to solve the problem.
* Modular Programming - While programming, the code is broken down into smaller group of instructions. These groups are known as modules, subprograms or subroutines. Modular programming based on the understanding of top-down analysis. It discourages jumps using ‘goto’ statements in the program, which often makes the program flow non-traceable. Jumps are prohibited and modular format is encouraged in structured programming.
* Structured Coding - In reference with top-down analysis, structured coding sub-divides the modules into further smaller units of code in the order of their execution. Structured programming uses control structure, which controls the flow of the program, whereas structured coding uses control structure to organize its instructions in definable patterns.

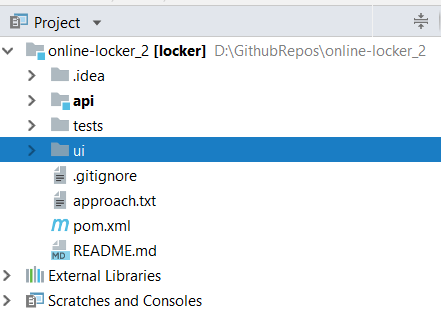
**5.2. Coding Details and Code Efficiency**



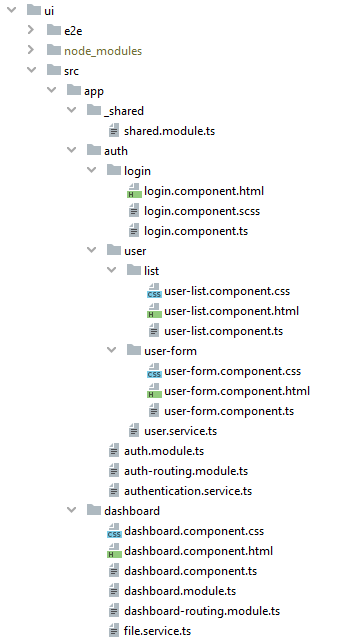
**Figure 22: App Tree**

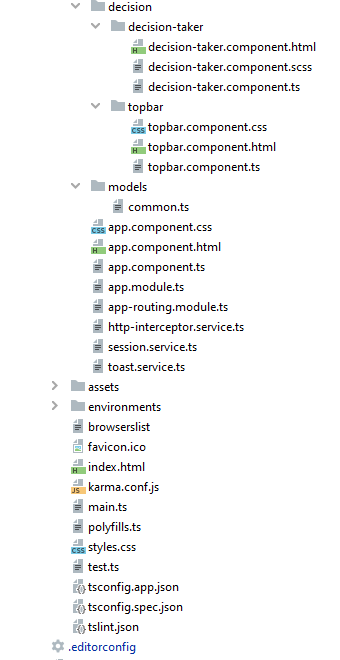
Digital Locker has been divided into multiple components to achieve high level of modulation.  
Let us explain about each component in more details along with the source code( Components may have some shared services i.e services used by others components).

Project Directory:



UI/Frontend.





Login Component HTML

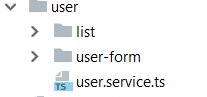
<**div class="navbar-brand"**>  
 <**a class="navbar-item"**>  
 <**img src="assets/lock.jpg" width="650px"**>  
 </**a**>  
</**div**>  
<**div class="ui-g"**>  
 <**div class="ui-g-6"**>  
 <**form [formGroup]="loginForm" \*ngIf="loginForm"**>  
  
 <**div class="ui-g"**>  
 <**div class="ui-g-12"**>  
 <**H4**>Login To Online Locker</**H4**>  
 </**div**>  
 <**div class="ui-g-12"**>  
 <**input pInputText  
 type="text"  
 placeholder="Enter Email"  
 formControlName="email"  
 required  
 pattern="^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$"** />  
 </**div**>  
 <**div class="ui-g-12"**>  
 <**input pPassword  
 type="password"  
 placeholder="Enter Password"  
 formControlName="password"  
 minlength="8"  
 required** />  
 </**div**>  
 <**div class="ui-g-12"**>  
 <**button pButton  
 label="Login"  
 (click)="doLogin()"  
 [disabled]="!loginForm.valid"**>  
 </**button**>  
 </**div**>  
 </**div**>  
 </**form**>  
 </**div**>  
</**div**>

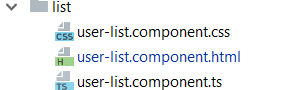
Login Component CSS

.**ui-g-6**{  
 **padding**: 8**px**;  
 **background**: **#9370DB**;  
 **margin**: 3**px**;  
 **font-weight**: **bold**;  
 **font-size**: 14**px**;  
 **border-radius**: 2**px**;  
 **box-shadow**: 0 9**px** 4**px** -6**px #a0a0a0**;  
 **border**: 1**px solid #c5c5c5**;  
 **cursor**: **pointer**;  
 **color**: **white**;  
}

Login Component Typscript file

**import** { Component, OnInit } **from '@angular/core'**;  
**import** { FormControl, FormGroup } **from '@angular/forms'**;  
**import** { AuthenticationService } **from '../authentication.service'**;  
**import** { SessionService } **from '../../session.service'**;  
**import** { Router } **from '@angular/router'**;  
  
@Component({  
 **selector**: **'app-login'**,  
 **templateUrl**: **'./login.component.html'**,  
 **styleUrls**: [**'./login.component.scss'**]  
})  
**export class** LoginComponent **implements** OnInit {  
 **loginForm**: FormGroup;  
 **constructor**(  
 **private authService**: AuthenticationService,  
 **private sessionService**: SessionService,  
 **private router**: Router  
 ) {}  
 ngOnInit() {  
 **this**.**loginForm** = **new** FormGroup({  
 **email**: **new** FormControl(**''**),  
 **password**: **new** FormControl(**''**),  
 });  
 }  
 doLogin() {  
 **const** val = **this**.**loginForm**.**value**;  
 **this**.**authService**.doLogin(val.**email**, val.**password**).subscribe((e) => {  
 **this**.**sessionService**.saveAllTokens(e);  
 **this**.**router**.navigate([**'dashboard'**]);  
 }, (err) => {  
 ***console***.error(**'some ex occured'** + err);  
  
 });  
 }  
 go(url: **string**) {  
 **this**.**router**.navigate([url]);  
 }  
}





List Component CSS

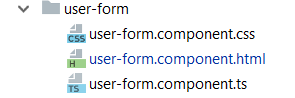
.**user-list-item** {  
 **padding**: 8**px**;  
 **background**: **#9370DB**;  
 **margin**: 3**px**;  
 **font-weight**: **bold**;  
 **font-size**: 16**px**;  
 **border-radius**: 2**px**;  
 **box-shadow**: 0 9**px** 4**px** -6**px #a0a0a0**;  
 **border**: 1**px solid #c5c5c5**;  
 **cursor**: **pointer**;  
 **color**: **white**;  
}

List Component HTML

<**div class="navbar-brand"**>  
 <**a class="navbar-item"**>  
 <**img src="assets/lock.jpg" width="800px"**>  
 </**a**>  
</**div**>  
<**div class="ui-g-12"**>  
 <**div class="ui-g-8" \*ngFor="let user of users; let idx = index"**>  
 <**div class="ui-g user-list-item"**>  
 <**div class="ui-g-2"**>{{ idx+1 }}.</**div**>  
 <**div class="ui-g-4"**>{{ user.email }}</**div**>  
 <**div class="ui-g-4"**>{{ user.fullName }}</**div**>  
 </**div**>  
 </**div**>  
</**div**>

List Component Typescript

**import** { Component, OnInit } **from '@angular/core'**;  
**import** { UserService } **from '../user.service'**;  
**import** { User } **from "../../../models/common"**;  
  
@Component({  
 **selector**: **'user-list'**,  
 **templateUrl**: **'./user-list.component.html'**,  
 **styleUrls**: [**'./user-list.component.css'**]  
})  
**export class** UserListComponent **implements** OnInit {  
  
 **users**: User[] = [];  
  
 **constructor**(**private userService**: UserService) { }  
  
 ngOnInit() {  
 **this**.**userService**.getAll().subscribe((r) => {  
 **this**.**users** = r.map(e => **new** User(e));  
 });  
 }  
}



User-Form Component CSS

.**form-cont** {  
 **padding**: 8**px**;  
 **background**: **#9370DB**;  
 **margin**: 3**px**;  
 **font-weight**: **bold**;  
 **font-size**: 16**px**;  
 **border-radius**: 2**px**;  
 **box-shadow**: 0 9**px** 4**px** -6**px #a0a0a0**;  
 **border**: 1**px solid #c5c5c5**;  
 **cursor**: **pointer**;  
 **color**: **white**;  
}

User-Form Component HTML

<**div class="navbar-brand"**>  
 <**a class="navbar-item"**>  
 <**img src="assets/lock.jpg" width="800px"**>  
 </**a**>  
</**div**>  
<**form [formGroup]="userForm" \*ngIf="!!userForm"**>  
 <**div class="ui-lg-7 ui-md-5 ui-g-7 ui-sm-12 form-cont"**>  
 <**div class="ui-g-4"**>  
 <**label**>First Name</**label**>  
 </**div**>  
 <**div class="ui-g-8"**>  
 <**input pInputText  
 type="text"  
 formControlName="firstName"  
 placeholder="First Name"  
 minlength="3"  
 required  
 pattern="[A-Za-z]+"** />  
 </**div**>  
 <**div class="ui-g-4"**>  
 <**label**>Last Name</**label**>  
 </**div**>  
 <**div class="ui-g-8"**>  
 <**input pInputText  
 type="text"  
 formControlName="lastName"  
 placeholder="Last Name"  
 minlength="3"  
 required  
 pattern="[A-Za-z]+"**/>  
  
 </**div**>  
 <**div class="ui-g-4"**>  
 <**label**>Email</**label**>  
 </**div**>  
 <**div class="ui-g-8"**>  
 <**input pInputText  
 type="text"  
 formControlName="email"  
 placeholder="Enter Email"  
 required  
 pattern="^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$"** />  
 </**div**>  
 <**div class="ui-g-4"**>  
 <**label**>Password</**label**>  
 </**div**>  
 <**div class="ui-g-8"**>  
 <**input pInputText  
 type="password"  
 formControlName="password"  
 placeholder="Enter Password"  
 minlength="8"  
 required**/>  
 </**div**>  
  
 <**div class="ui-g-12"**>  
 <**button pButton label="Save" (click)="saveUser()" [disabled]="!userForm.valid"**></**button**>  
 </**div**>  
 </**div**>  
</**form**>

User-Form Component Typescript

**import** {Component, OnInit} **from '@angular/core'**;  
**import** {Router} **from '@angular/router'**;  
**import** {FormBuilder, FormControl, FormGroup} **from '@angular/forms'**;  
**import** {User} **from '../../../models/common'**;  
**import** {UserService} **from '../user.service'**;  
**import** {ToastService} **from '../../../toast.service'**;  
  
@Component({  
 **selector**: **'user-form'**,  
 **templateUrl**: **'./user-form.component.html'**,  
 **styleUrls**: [**'./user-form.component.css'**]  
})  
**export class** UserFormComponent **implements** OnInit {  
 **user**: User = **new** User();  
 **userForm**: FormGroup;  
 **constructor**(  
 **private router**: Router,  
 **private userService**: UserService,  
 **private fb**: FormBuilder,  
 **private toastService**: ToastService  
 ) {}  
 ngOnInit() {  
 **this**.loadForm();  
 }  
 **private** loadForm() {  
 **this**.**userForm** = **this**.**fb**.group({  
 **email**: **new** FormControl(),  
 **password**: **new** FormControl(),  
 **firstName**: **new** FormControl(),  
 **lastName**: **new** FormControl()  
 });  
 }  
 saveUser() {  
 **const** tempUser = **new** User(**this**.**userForm**.**value**);  
 **this**.**userService**.save(tempUser).subscribe(r => {  
 **this**.**toastService**.*success*(**'User has been created successfully'**);  
 **this**.**router**.navigate([**'login'**]);  
 });}}

User Service

**import** { Injectable } **from '@angular/core'**;  
**import** { HttpClient } **from '@angular/common/http'**;  
**import** { Observable } **from 'rxjs'**;  
**import** { User } **from "../../models/common"**;  
  
@Injectable()  
**export class** UserService {  
  
 **constructor**(**private httpClient**: HttpClient) {  
 }  
  
 getAll(): Observable<**any**> {  
 **return this**.**httpClient**.get(**'/public/users'**);  
 }  
  
 save(user: User): Observable<**any**> {  
 **return this**.**httpClient**.post(**'/public/users'**, user);  
 }  
}

Auth Routing Module

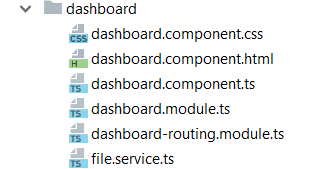
**import** { NgModule } **from '@angular/core'**;  
**import** { SharedModule } **from '../\_shared/shared.module'**;  
**import** { LoginComponent } **from './login/login.component'**;  
**import** { AuthenticationService } **from './authentication.service'**;  
**import** { AuthRoutingModule } **from './auth-routing.module'**;  
**import** { UserFormComponent } **from "./user/user-form/user-form.component"**;  
**import** { UserListComponent } **from "./user/list/user-list.component"**;  
**import** { UserService } **from "./user/user.service"**;  
  
@NgModule({  
 **declarations**: [  
 LoginComponent,  
 UserFormComponent,  
 UserListComponent,  
 ],  
 **imports**: [  
 SharedModule,  
 AuthRoutingModule,  
 ],  
 **providers**: [  
 AuthenticationService,  
 UserService,  
 ]  
})  
**export class** AuthModule {  
}

Auth-Routing Module

**import** { NgModule } **from '@angular/core'**;  
**import** { RouterModule, Routes } **from '@angular/router'**;  
**import** { LoginComponent } **from './login/login.component'**;  
**import** { UserFormComponent } **from "./user/user-form/user-form.component"**;  
**import** { UserListComponent } **from "./user/list/user-list.component"**;  
  
**const** routes: Routes = [  
 {**path**: **'login'**, **component**: LoginComponent},  
 {**path**: **'users'**, **component**: UserListComponent},  
 {**path**: **'users/add'**, **component**: UserFormComponent}  
];  
  
@NgModule({  
 **imports**: [RouterModule.*forRoot*(routes)],  
 **exports**: [RouterModule]  
})  
**export class** AuthRoutingModule {  
}

Authentication Service

**import** { Injectable } **from '@angular/core'**;  
**import** { HttpClient, HttpHeaders } **from '@angular/common/http'**;  
**import** { Observable } **from 'rxjs'**;  
**import** { SessionService } **from "../session.service"**;  
  
@Injectable({  
 **providedIn**: **'root'**})  
**export class** AuthenticationService {  
  
 **constructor**(**private httpClient**: HttpClient,  
 **private sessionService**: SessionService) {  
 }  
  
 doLogin(email: **string**, password: **string**): Observable<**any**> {  
 **const** oauthTokenUrl = **'/public/credentials/login'**;  
 **const** body = {  
 **email**: **`**${email}**`**,  
 **password**: **`**${password}**`** };  
 **const** httpHeaders: HttpHeaders = **new** HttpHeaders()  
 .set(**'Content-Type'**, **'application/json'**);  
 **return this**.**httpClient**.post(oauthTokenUrl, body, {**headers**: httpHeaders});  
 }  
  
 doLogout(): Observable<**any**> {  
 **return this**.**httpClient**.delete(**'/public/credentials/logout/'** +  
 **this**.**sessionService**.getAccessToken());  
 }  
}



Dashboard Component CSS

.**file-item** {  
 **padding**: 8**px**;  
 **background**: **#9370DB**;  
 **margin**: 3**px**;  
 **font-weight**: **bold**;  
 **font-size**: 14**px**;  
 **border-radius**: 2**px**;  
 **box-shadow**: 0 9**px** 4**px** -6**px #a0a0a0**;  
 **border**: 1**px solid #c5c5c5**;  
 **cursor**: **pointer**;  
 **color**: **white**;  
}  
**button**{  
 **border-radius**: 0**px**;  
 **background**: **#9370DB** ;  
 **font-size**: 18**px**;  
 **color**: **white**;  
}  
  
.**container**{  
 **background-image**: **url**(**"user.png"**);  
 **background-repeat**: **no-repeat**;  
}

Dashboard Component HTML

<**div class="container"**>  
  
 <**div class="ui-g" style="padding**: 25**px"**>  
 <**p-progressSpinner \*ngIf="fileUploading"  
 [style]="{width: '50px', height: '50px'}"  
 strokeWidth="4" animationDuration=".5s"**>  
 </**p-progressSpinner**>  
  
 <**input type="file" (change)="fileEvent(*$event*)"**>  
 </**div**>  
  
 <**div class="ui-g"**>  
 <**div class="ui-g-6" \*ngFor="let *uf* of uploadedFiles; let *idx* = *index*"**>  
 <**div class="ui-g file-item"**>  
 <**div class="ui-g-1"**>{{ ***idx***+1 }}.</**div**>  
 <**div class="ui-g-6"**>{{ ***uf***.**fileName** }}</**div**>  
  
 <**div class="ui-g-2"**>  
 <**button pButton type="button" label="Save" (click)="downloadFile(*uf*.id, *uf*.fileName)"  
 class="ui-button-success" icon="pi pi-download"**>  
 </**button**>  
 </**div**>  
 <**div class="ui-g-2"**>  
 <**button pButton type="button" label="Delete" (click)="removeFile(*uf*.id)"  
 class="ui-button-danger" icon="pi pi-trash"**>  
 </**button**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
  
</**div**>

Dashboard Component Typescript

**import** { Component, OnInit } **from '@angular/core'**;  
**import** { UploadedFile } **from '../models/common'**;  
**import** { FileService } **from './file.service'**;  
**import** {ToastService} **from "../toast.service"**;  
  
@Component({  
 **selector**: **'dashboard'**,  
 **templateUrl**: **'./dashboard.component.html'**,  
 **styleUrls**: [**'./dashboard.component.css'**]  
})  
**export class** DashboardComponent **implements** OnInit {  
 **uploadedFiles**: UploadedFile[] = [];  
 **fileUploading**: **boolean** = **false**;  
 **constructor**(  
 **private fileService**: FileService,  
 **private toastService**: ToastService) { }  
 ngOnInit() {  
 **this**.**fileService**.listAll().subscribe((r) => {  
 **this**.**uploadedFiles** = r.map(e => **new** UploadedFile(e));  
 });  
 }  
 downloadFile(fileId: **string**, filename: **string**) {  
 **this**.**fileService**.download(fileId).subscribe(res => {  
 **var** url = ***window***.**URL**.*createObjectURL*(res);  
 **var** a = ***document***.createElement(**'a'**);  
 ***document***.**body**.appendChild(a);  
 a.setAttribute(**'style'**, **'display: none'**);  
 a.**href** = url;  
 res.**filename** = filename;  
 a.**download** = res.**filename**;  
 a.click();  
 ***window***.**URL**.*revokeObjectURL*(url);  
 a.remove();  
 }, error => {  
 ***console***.log(**'download error:'**, JSON.stringify(error));  
 }, () => {  
 ***console***.log(**'Completed file download.'**);  
 });  
 }  
  
 **public** fileEvent($event) {  
 **this**.**fileUploading** = **true**;  
 **const** fileSelected: File = $event.**target**.**files**[0];  
 **this**.**fileService**.upload(fileSelected).subscribe((response) => {  
  
 **this**.**toastService**.*success*(**'Uploading'**, **'File has been uploaded successfully'**);  
  
 **this**.**fileService**.listAll().subscribe((r) => {  
 **this**.**uploadedFiles** = r.map(e => **new** UploadedFile(e));  
 });  
 **return** response;  
 }, e => ***console***.log(e), () => {  
 **this**.**fileUploading** = **false**;  
 });  
 }  
  
 removeFile(fileId: **string**) {  
  
 **this**.**fileService**.remove(fileId).subscribe(res => {  
 **this**.**toastService**.*success*(**'Deleting'**, **'File has been deleted successfully'**);  
  
 **this**.**fileService**.listAll().subscribe((r) => {  
 **this**.**uploadedFiles** = r.map(e => **new** UploadedFile(e));  
 });  
 });  
 }  
}

Dashboard Module Typescript

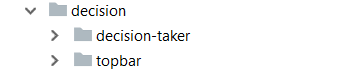
**import** { NgModule } **from '@angular/core'**;  
**import** { SharedModule } **from '../\_shared/shared.module'**;  
**import** { DashboardRoutingModule } **from './dashboard-routing.module'**;  
**import** {DashboardComponent} **from "./dashboard.component"**;  
**import** {FileService} **from "./file.service"**;  
  
@NgModule({  
 **declarations**: [  
 DashboardComponent,  
 ],  
 **imports**: [  
 SharedModule,  
 DashboardRoutingModule,  
 ],  
 **providers**: [  
 FileService,  
 ]  
})  
**export class** DashboardModule {  
}

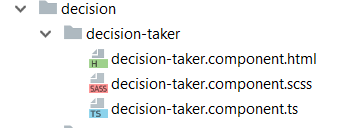
Dashboard Routing TypeScript

**import** { NgModule } **from '@angular/core'**;  
**import** { RouterModule, Routes } **from '@angular/router'**;  
**import** { DashboardComponent } **from "./dashboard.component"**;  
  
**const** routes: Routes = [  
 {**path**: **'dashboard'**, **component**: DashboardComponent}  
];  
  
@NgModule({  
 **imports**: [RouterModule.*forRoot*(routes)],  
 **exports**: [RouterModule]  
})  
**export class** DashboardRoutingModule {  
}

File Service

**import** { Injectable } **from '@angular/core'**;  
**import** { HttpClient } **from '@angular/common/http'**;  
**import** { Observable } **from 'rxjs'**;  
  
@Injectable()  
**export class** FileService {  
  
 **constructor**(**private httpClient**: HttpClient) {  
 }  
  
 listAll(): Observable<**any**> {  
 **return this**.**httpClient**.get(**'/secured/files'**);  
 }  
  
 upload(fileToUpload: File) {  
 **const** formData = **new *FormData***();  
 formData.append(**'file'**, fileToUpload, fileToUpload.**name**);  
 **return this**.**httpClient**.post(**'/secured/files/upload'**, formData);  
 }  
  
 download(fileId: **string**): Observable<**any**> {  
 **return this**.**httpClient**.get(**`/secured/files/download/**${fileId}**`**, {  
 **responseType**: **'blob'** });  
 }  
 remove(fileId: **string**): Observable<**any**> {  
 **return this**.**httpClient**.delete(**`/secured/files/**${fileId}**`**);  
 }  
}



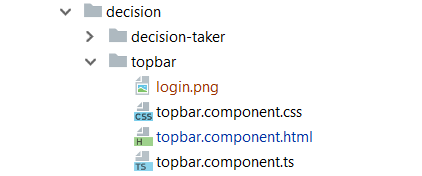


Decision Taker Component HTML

<**div class="overall"**>  
 <**div class="topbar-leftsidebar"**>  
 <**ng-container \*ngIf="shouldBehaveAsAuthenticated"**>  
 <**app-topbar**></**app-topbar**>  
 </**ng-container**>  
  
 <**ng-container \*ngIf="!shouldBehaveAsAuthenticated"**>  
 <**app-topbar [isAuthenticated]="false"**></**app-topbar**>  
 </**ng-container**>  
  
 </**div**>  
  
 <**div class="main-component-loader"**>  
 <**router-outlet**></**router-outlet**>  
 </**div**>  
</**div**>  
<**p-toast**></**p-toast**>

Decision Taker Component TypeScript

**import** {Component} **from '@angular/core'**;  
**import** {SessionService} **from '../../session.service'**;  
**import** {Event, NavigationStart, Router} **from '@angular/router'**;  
  
@Component({  
 **selector**: **'decision-taker'**,  
 **templateUrl**: **'./decision-taker.component.html'**,  
 **styleUrls**: [**'./decision-taker.component.scss'**]  
})  
**export class** DecisionTakerComponent {  
  
 **shouldBehaveAsAuthenticated**: **boolean**;  
  
 **private noAuthNeededRoutes**: **string**[] = [  
 **'/login'**,  
 **'/users'**,  
 **'/error'** ];  
  
 **constructor**(  
 **private sessionService**: SessionService,  
 **private router**: Router,  
 ) {  
 **this**.**shouldBehaveAsAuthenticated** = **false**;  
  
 **this**.**router**.**events**.subscribe((ev: Event) => {  
 **if** (ev **instanceof** NavigationStart) {  
 **this**.takeDecision(ev);  
 }  
 });  
 }  
  
 **private** noAuthUrlExists(currUrl: **string**): **boolean** {  
 **return** (**this**.**noAuthNeededRoutes** .filter(r => currUrl.startsWith(r)).**length**) > 0;  
 }  
  
 **private** takeDecision(routeEv: NavigationStart) {  
 **const** currentUrl = routeEv.**url**;  
  
 **if** (!**this**.**sessionService**.isAuthenticated()) {  
 **if** (!**this**.noAuthUrlExists(currentUrl)) {  
 **this**.**router**.navigate([**'login'**]);  
 }  
 **this**.**shouldBehaveAsAuthenticated** = **false**;  
 } **else** {  
 **if** (**this**.noAuthUrlExists(currentUrl)) {  
 **this**.**shouldBehaveAsAuthenticated** = **false**;  
 } **else** {  
 **if** (currentUrl === **'/'**) {  
 **this**.**router**.navigate([**'dashboard'**]);  
 }  
 **this**.**shouldBehaveAsAuthenticated** = **true**;  
 }  
 }  
 }  
}



Topbar Component CSS

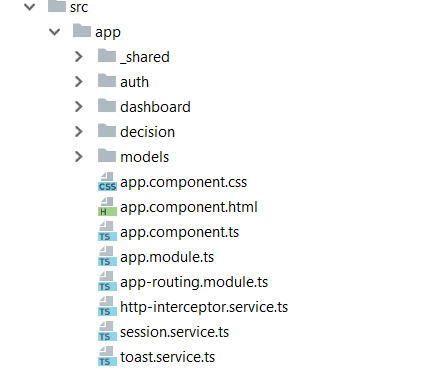
**button**{  
 **border-radius**: 0**px**;  
 **background**: **#9370DB** ;  
 **font-size**: 18**px**;  
 **color**: **white**;  
 **padding**: 10**px** 20**px**;  
}  
.**mynavbar**{  
 **background**: **#9370DB**;  
}

Topbar Component HTML

<**div class="mynavbar"**>  
 <**div \*ngIf="isAuthenticated"**>  
 <**button type="button" class="btn btn-secondary" (click)="go('dashboard')"**>Dashboard</**button**>  
 <**button type="button" class="btn btn-secondary pull-right" (click)="doLogout()"**>Logout</**button**>  
 </**div**>  
</**div**>  
<**div class="mynavbar"**>  
 <**div \*ngIf="!isAuthenticated"**>  
  
 <**div class="navbar-brand"**>  
 <**a class="navbar-item"**>  
 <**img src="assets/login.png" width="50px"**>  
 </**a**>  
 </**div**>  
 <**button type="button" class="btn btn-secondary" (click)="go('users/add')"**>Registration Here</**button**>  
 <**button type="button" class="btn btn-secondary" (click)="go('users')"**>Users List</**button**>  
 <**button type="button" class="btn btn-secondary pull-right" (click)="go('login')"**>Login</**button**>  
  
  
 </**div**>  
</**div**>  
<**hr**>

Topbar Component TypeScript

**import** {Component, Input} **from '@angular/core'**;  
**import** { Router } **from '@angular/router'**;  
**import** { AuthenticationService } **from '../../auth/authentication.service'**;  
**import** { SessionService } **from '../../session.service'**;  
  
@Component({  
 **selector**: **'app-topbar'**,  
 **templateUrl**: **'./topbar.component.html'**,  
 **styleUrls**: [**'./topbar.component.css'**]  
})  
**export class** TopbarComponent {  
  
 @Input() **isAuthenticated**: **boolean** = **true**;  
  
 **constructor**(  
 **private router**: Router,  
 **private authService**: AuthenticationService,  
 **private sessionService**: SessionService,  
 ) {  
 }  
  
 go(url: **string**) {  
 **this**.**router**.navigate([url]);  
 }  
  
 doLogout() {  
 **this**.**authService**.doLogout().subscribe((e) => {  
 **this**.**sessionService**.deleteTokens();  
 **this**.**router**.navigate([**'login'**]);  
 });  
 }  
}



Models

App Component HTML

<**div class="container"**>  
 <**decision-taker**></**decision-taker**>  
</**div**>

App Component Typescript

**import** { Component } **from '@angular/core'**;  
  
@Component({  
 **selector**: **'app-root'**,  
 **templateUrl**: **'./app.component.html'**,  
 **styleUrls**: [**'./app.component.css'**]  
})  
**export class** AppComponent {  
}

App Module

**import** { BrowserModule } **from '@angular/platform-browser'**;  
**import** { NgModule } **from '@angular/core'**;  
  
**import** { AppComponent } **from './app.component'**;  
**import** { BrowserAnimationsModule } **from '@angular/platform-browser/animations'**;  
**import** { DecisionTakerComponent } **from './decision/decision-taker/decision-taker.component'**;  
**import** { TopbarComponent } **from './decision/topbar/topbar.component'**;  
**import** { AuthModule } **from './auth/auth.module'**;  
**import** { AppRoutingModule } **from './app-routing.module'**;  
**import** { SessionService } **from './session.service'**;  
**import** { SharedModule } **from './\_shared/shared.module'**;  
**import** { HttpInterceptorService } **from './http-interceptor.service'**;  
**import** { HTTP\_INTERCEPTORS, HttpClientModule } **from '@angular/common/http'**;  
**import** { ToastService } **from './toast.service'**;  
**import** { MessageService } **from 'primeng/api'**;  
**import** { ToastModule } **from 'primeng/toast'**;  
**import** { DashboardModule } **from './dashboard/dashboard.module'**;  
  
@NgModule({  
  
 **declarations**: [  
 AppComponent,  
 DecisionTakerComponent,  
 TopbarComponent,  
 ],  
 **imports**: [  
 BrowserModule,  
 BrowserAnimationsModule,  
 HttpClientModule,  
  
 SharedModule,  
 AuthModule,  
 DashboardModule,  
 AppRoutingModule,  
  
 ToastModule,  
 ],  
 **providers**: [  
 SessionService,  
 HttpInterceptorService,  
 MessageService,  
 ToastService,  
 {  
 **provide**: HTTP\_INTERCEPTORS,  
 **useClass**: HttpInterceptorService,  
 **multi**: **true** }  
 ],  
 **bootstrap**: [AppComponent]  
})  
**export class** AppModule {  
}

App Routing

**import** { NgModule } **from '@angular/core'**;  
**import** { Routes, RouterModule } **from '@angular/router'**;  
  
**const** routes: Routes = [];  
  
@NgModule({  
 **imports**: [RouterModule.*forRoot*(routes)],  
 **exports**: [RouterModule]  
})  
**export class** AppRoutingModule {  
}

HTTP interseptor service

**import** { Injectable } **from '@angular/core'**;  
**import** { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } **from '@angular/common/http'**;  
**import** { Observable } **from 'rxjs'**;  
**import** { SessionService } **from './session.service'**;  
  
@Injectable({  
 **providedIn**: **'root'**})  
**export class** HttpInterceptorService **implements** HttpInterceptor {  
  
 **constructor**(**private sessionService**: SessionService) {}  
  
 intercept(req: HttpRequest<**any**>, next: HttpHandler): Observable<HttpEvent<**any**>> {  
 **if** (!req.**url**.startsWith(**'/public/'**)) {  
 req = req.clone({  
 **setHeaders**: { **AccessToken**: **this**.**sessionService**.getAccessToken() }  
 });  
 }  
 **return** next.handle(req);  
 }  
}

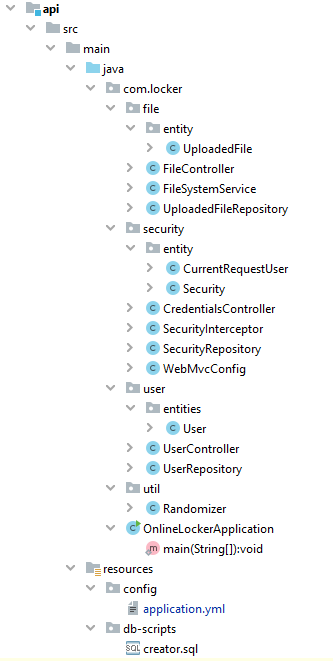
Session Service

**import** { Injectable } **from '@angular/core'**;  
**import** { HttpClient } **from '@angular/common/http'**;  
  
@Injectable({  
 **providedIn**: **'root'**})  
**export class** SessionService {  
  
 **constructor**(**private httpClient**: HttpClient) {  
 }  
  
 saveAllTokens(e) {  
 ***localStorage***.setItem(**'access\_token'**, e.**accessToken**);  
 }  
  
 isAuthenticated(): **boolean** {  
 **return** !!***localStorage***.getItem(**'access\_token'**);  
 }  
  
 getAccessToken() {  
 **return *localStorage***.getItem(**'access\_token'**);  
 }  
  
 deleteTokens() {  
 ***localStorage***.removeItem(**'access\_token'**);  
 }  
}

Toast Service

**import** { Injectable } **from '@angular/core'**;  
**import** { MessageService } **from 'primeng/api'**;  
  
@Injectable({  
 **providedIn**: **'root'**})  
**export class** ToastService {  
  
 **constructor**(**private messageService**: MessageService) {  
 }  
  
 *success* = (summary: **string**, detail?: **string**) =>  
 **this**.**messageService**.add({ **severity**: **'success'**, summary, detail })  
  
 *info* = (summary: **string**, detail?: **string**) =>  
 **this**.**messageService**.add({ **severity**: **'info'**, summary, detail })  
  
 *warn* = (summary: **string**, detail?: **string**) =>  
 **this**.**messageService**.add({ **severity**: **'warn'**, summary, detail })  
  
 *error* = (summary: **string**, detail?: **string**) =>  
 **this**.**messageService**.add({ **severity**: **'error'**, summary, detail })  
}

API/Backend.



Upload File

**package** com.locker.file.entity;  
  
**public class** UploadedFile {  
  
 **public** Integer id;  
 **public** Integer userId;  
 **public** String fileName;  
 **public** String relativePath;  
  
 **public** UploadedFile() {  
 }  
  
 **public** UploadedFile(Object[] u) {  
 **this**.id = (Integer) u[0];  
 **this**.userId = (Integer) u[1];  
 **this**.fileName = (String) u[2];  
 **this**.relativePath = (String) u[3];  
 }  
}

File Controller

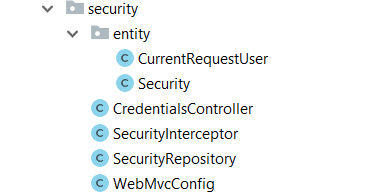
**package** com.locker.file;  
  
**import** com.locker.file.entity.UploadedFile;  
**import** com.locker.security.entity.CurrentRequestUser;  
**import** com.locker.util.Randomizer;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.core.io.Resource;  
**import** org.springframework.http.HttpHeaders;  
**import** org.springframework.http.MediaType;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
**import** org.springframework.web.multipart.MultipartFile;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.transaction.Transactional;  
**import** java.io.IOException;  
**import** java.nio.file.Path;  
**import** java.util.List;  
  
*/\*\*  
 \* FileController is a file manipulator endpoints container. the endpoints are  
 \* designed in a way that will help user to upload a MULTIPART file via post request,  
 \* download a uploaded file by its id, view all the uploaded files list.  
 \*  
 \* The controller internally uses two fields - uploadedFileRepository and fileSystemService.  
 \*/*@RestController  
@RequestMapping(**"/secured/files"**)  
**public class** FileController {  
  
 @Autowired  
 **private** UploadedFileRepository **uploadedFileRepository**;  
  
 @Autowired  
 **private** FileSystemService **fileSystemService**;  
  
 */\*\*  
 \* this method will return all the uploaded\_file tables entries for a current  
 \* user. Internally it calls repository class with currentRequest userId.  
 \*  
 \** ***@return*** *a list of UploadedFile entity records from database.  
 \*/* @GetMapping  
 @org.springframework.transaction.annotation.Transactional(readOnly = **true**)  
 **public** List<UploadedFile> getAllFiles() {  
 **return uploadedFileRepository**.getAll(CurrentRequestUser.*securedUser*.**userId**);  
 }  
  
 */\*\*  
 \* An Uploader resource endpoint, which saves file in server's temp  
 \* folder as it is. it then creates an entry in uploaded\_file table  
 \* with that file absolute path.  
 \** ***@param file*** *a MultipartFile sent via post request, which will be  
 \* written to directory.  
 \** ***@return*** *a saved UploadedFile entity object.  
 \*/* @PostMapping(**"/upload"**)  
 @Transactional  
 **public** UploadedFile upload(@RequestParam(**"file"**) MultipartFile file) {  
  
 **if** (file.isEmpty()) {  
 **throw new** RuntimeException(**"Please provide a file to upload"**);  
 }  
  
 Path storedFilePath = fileSystemService.write(file);  
  
 UploadedFile up = **new** UploadedFile();  
 up.id = Randomizer.generateInt();  
 up.userId = CurrentRequestUser.securedUser.userId;  
 up.relativePath = storedFilePath.toAbsolutePath().toString();  
 up.fileName = storedFilePath.getFileName().toString();  
  
 uploadedFileRepository.save(up);  
  
 **return** up;  
 }  
  
 */\*\*  
 \* A Downloader resource endpoint, it first fetches the UploadedFile entity  
 \* object first based on currently user and fileId. If record doesn't exists  
 \* then Application exception will be thrown.  
 \* If record exists, then Spring resource object will be fetched from  
 \* fileSystem by that file name. and returned to UI with its contentType  
 \* set in the response.  
 \*  
 \** ***@param*** *fileId the uploaded\_file table id to search in db. only if it exists  
 \* for current userId then it will be returned.  
 \** ***@param*** *request httpServlet request to to pull mimeType for particular  
 \* spring file resource.  
 \*  
 \** ***@return*** *Spring Resource object for given fileId.  
 \*/* @GetMapping(**"/download/{fileId}"**)  
 @org.springframework.transaction.annotation.Transactional(readOnly = **true**)  
 **public** ResponseEntity<Resource> download(@PathVariable Integer fileId,  
 HttpServletRequest request) {  
  
 *// first loading db object with fileId.* UploadedFile fileDBObject = uploadedFileRepository.getByIdAndUserId(  
 fileId, CurrentRequestUser.securedUser.userId);  
  
 *// then preparing its spring resource object from fileSystem* Resource resource = fileSystemService.loadFileAsResource(  
 fileDBObject.fileName);  
  
 *// Try to determine file's content type* String contentType = **null**;  
 **try** {  
 contentType = request.getServletContext().getMimeType(  
 resource.getFile().getAbsolutePath());  
 } **catch** (IOException ex) {  
 **throw new** RuntimeException(**"Could not determine file type."**);  
 }  
 *// Fallback to the default content type if type could not be determined* **if** (contentType == **null**) {  
 contentType = **"application/octet-stream"**;  
 }  
  
 **return** ResponseEntity.ok()  
 .contentType(MediaType.parseMediaType(contentType))  
 .header(HttpHeaders.CONTENT\_DISPOSITION,  
 **"attachment; filename=\""** + resource.getFilename() + **"\""**)  
 .body(resource);  
 }  
  
 */\*\*  
 \* This endpoint deletes the file from server's temp directory and  
 \* then removes from database as well.  
 \** ***@param*** *fileId  
 \*/* @DeleteMapping(**"/{fileId}"**)  
 @Transactional  
 **public void** remove(@PathVariable Integer fileId) {  
  
 UploadedFile fileDBObject = uploadedFileRepository.getByIdAndUserId(  
 fileId, CurrentRequestUser.securedUser.userId);  
  
 fileSystemService.removeFileAsResource(fileDBObject.fileName);  
  
 uploadedFileRepository.deleteById(fileId);  
 }  
}

File System Service

**package** com.locker.file;  
  
**import** org.springframework.core.io.Resource;  
**import** org.springframework.core.io.UrlResource;  
**import** org.springframework.stereotype.Service;  
**import** org.springframework.web.multipart.MultipartFile;  
  
**import** java.io.IOException;  
**import** java.net.MalformedURLException;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
  
*/\*\*  
 \* A fileSystem manipulator class. this class takes a spring multipart file  
 \* and persist that in server's temp folder.  
 \* It loads a file based on its file name.  
 \* and lastly it removes a file too based on its name.  
 \*/*@Service  
**public class** FileSystemService {  
  
 **private static** String *UPLOADED\_FOLDER* = **"C://temp//"**;  
 **private** Path **fileStorageLocation**;  
  
 **public** Path write(MultipartFile file) {  
 Path storedFilePath = **null**;  
 **try** {  
  
 *// Get the file and save it somewhere* **byte**[] bytes = file.getBytes();  
 storedFilePath = Paths.*get*(*UPLOADED\_FOLDER* + file  
 .getOriginalFilename());  
 Files.*write*(storedFilePath, bytes);  
  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** storedFilePath;  
 }  
  
 **public** Resource loadFileAsResource(String fileName) {  
 **this**.**fileStorageLocation** = Paths.*get*(*UPLOADED\_FOLDER*).toAbsolutePath()  
 .normalize();  
 **try** {  
  
 Path filePath = **this**.**fileStorageLocation**.resolve(fileName).normalize();  
 Resource resource = **new** UrlResource(filePath.toUri());  
 **if** (resource.exists()) {  
 **return** resource;  
 } **else** {  
 **throw new** RuntimeException(**"File not found "** + fileName);  
 }  
 } **catch** (MalformedURLException ex) {  
 **throw new** RuntimeException(**"File not found "** + fileName, ex);  
 }  
 }  
  
 **public void** removeFileAsResource(String fileName) {  
 **this**.**fileStorageLocation** = Paths.*get*(*UPLOADED\_FOLDER*).toAbsolutePath()  
 .normalize();  
 **try** {  
 Path filePath = **this**.**fileStorageLocation**.resolve(fileName).normalize();  
 Resource resource = **new** UrlResource(filePath.toUri());  
 **if** (resource.exists()) {  
 **boolean** status = resource.getFile().delete();  
 **if** (!status)  
 **throw new** RuntimeException(**"File "** + fileName +  
 **" could not be removed"**);  
 } **else** {  
 **throw new** RuntimeException(**"File not found "** + fileName);  
 }  
 } **catch** (MalformedURLException ex) {  
 **throw new** RuntimeException(**"File not found "** + fileName, ex);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
}

Upload File Repository

**package** com.locker.file;  
  
**import** com.locker.file.entity.UploadedFile;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Repository;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.Query;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \* A uploaded\_file database repository class. helps in manipulating uploaded\_file table.  
 \*/*@Repository  
**public class** UploadedFileRepository {  
  
 @Autowired  
 **private** EntityManager **entityManager**;  
  
 **public** List<UploadedFile> getAll(Integer userId) {  
  
 Query q = **entityManager**.createNativeQuery(**"SELECT** *\** **FROM uploaded\_file"** +  
 **" WHERE user\_id = ?"**)  
 .setParameter(1, userId);  
 List<Object[]> uploadedFilesRes = q.getResultList();  
  
 List<UploadedFile> allUploadedFiles = **new** ArrayList<UploadedFile>();  
 **for** (Object[] uf : uploadedFilesRes) {  
 UploadedFile u = **new** UploadedFile(uf);  
 allUploadedFiles.add(u);  
 }  
 **return** allUploadedFiles;  
  
 }  
  
 **public void** save(UploadedFile up) {  
 **entityManager**.createNativeQuery(**"INSERT INTO uploaded\_file"** +  
 **" (id, user\_id, file\_name, relative\_path) VALUES (?, ?, ?, ?)"**)  
 .setParameter(1, up.**id**)  
 .setParameter(2, up.**userId**)  
 .setParameter(3, up.**fileName**)  
 .setParameter(4, up.**relativePath**)  
 .executeUpdate();  
 }  
  
 **public** UploadedFile getByIdAndUserId(Integer fileId, Integer userId) {  
 Query q = **entityManager**.createNativeQuery(**"SELECT** *\** **FROM uploaded\_file"** +  
 **" WHERE id = ? AND user\_id = ?"**)  
 .setParameter(1, fileId)  
 .setParameter(2, userId);  
 List<Object[]> uploadedFilesRes = q.getResultList();  
  
 **if** (uploadedFilesRes.isEmpty())  
 **throw new** RuntimeException(**"Uploaded file Id "** + fileId +  
 **" doesn't exists for this user"**);  
  
 **return new** UploadedFile(uploadedFilesRes.get(0));  
 }  
  
 **public void** deleteById(Integer fileId) {  
 **entityManager**.createNativeQuery(**"DELETE FROM uploaded\_file WHERE id = ?"**)  
 .setParameter(1, fileId)  
 .executeUpdate();  
 }  
}



Current Request User

**package** com.locker.security.entity;  
  
**public class** CurrentRequestUser {  
  
 **public static** Security *securedUser*;  
}

Security

**package** com.locker.security.entity;  
  
**public class** Security {  
  
 **public** Integer **id**;  
 **public** Integer **userId**;  
 **public** Integer **accessToken**;  
  
 **public** Security() {}  
  
 **public** Security(Object[] u) {  
 **this**.**id** = (Integer) u[0];  
 **this**.**userId** = (Integer) u[1];  
 **this**.**accessToken** = (Integer) u[2];  
 }  
}

Credentials Controller

**package** com.locker.security;  
  
**import** com.locker.security.entity.Security;  
**import** com.locker.user.UserRepository;  
**import** com.locker.user.entities.User;  
**import** com.locker.util.Randomizer;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** javax.transaction.Transactional;  
  
*/\*\*  
 \* CredentialsController helps in doing loging and logout in the application.  
 \* Given the user's email and password, this controller asks repository  
 \* to see if that emaild/password combination exits. if not then simply  
 \* either email or password is wrong.  
 \* <p>  
 \* If a combination exists in the system, check its already  
 \* generated security entity record.  
 \* if there is already a security record generated in db, then just return  
 \* it otherwise create a new one. and then return security object.  
 \*/*@RestController  
@RequestMapping(**"/public/credentials"**)  
**public class** CredentialsController {  
  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 @Autowired  
 **private** SecurityRepository **securityRepository**;  
  
 */\*\*  
 \* this method takes email/password fields from user object, and checks its  
 \* existence in db for security  
 \** ***@param user*** *UI should pass email and password in this object.  
 \** ***@return*** *saved/fetched security object from db.  
 \*/* @PostMapping(**"/login"**)  
 @Transactional  
 **public** ResponseEntity<Security> doLogin(@RequestBody User user) {  
  
 */\* first check whether this email/password combination exists.  
 if yes. then insert a record in security table. and return that too.  
 otherwise tell user that email/password is invalid.  
 \*/* User loginUser = **userRepository**.getByEmailAndPassword(  
 user.**email**, user.**password**);  
  
 Security security = **securityRepository**.getByUserId(loginUser.**id**);  
 **if** (security != **null**)  
 **return** ResponseEntity.*ok*(security);  
  
 Security sec = **new** Security();  
 sec.**id** = Randomizer.*generateInt*();  
 sec.**userId** = loginUser.**id**;  
 sec.**accessToken** = Randomizer.*generateInt*();  
  
 **securityRepository**.save(sec);  
  
 **return** ResponseEntity.*ok*(sec);  
 }  
  
 */\*\*  
 \* this method simply removes security table row for given  
 \* tokenId via repository call.  
 \** ***@param securityTokenId*** *\*/* @DeleteMapping(**"/logout/{securityTokenId}"**)  
 @Transactional  
 **public void** doLogout(@PathVariable Integer securityTokenId) {  
 **securityRepository**.deleteById(securityTokenId);  
 }  
}

Security Interceptor

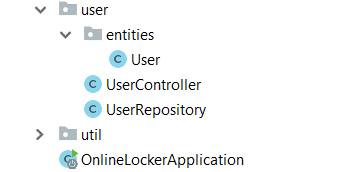
**package** com.locker.security;  
  
**import** com.locker.security.entity.CurrentRequestUser;  
**import** com.locker.security.entity.Security;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Service;  
**import** org.springframework.web.servlet.handler.HandlerInterceptorAdapter;  
  
**import** javax.servlet.http.HttpServletRequest;  
**import** javax.servlet.http.HttpServletResponse;  
  
*/\*\*  
 \* A Crucial Service helps in securing the endpoints before they actually hit.  
 \* this class extends {****@link*** *HandlerInterceptorAdapter } class which has  
 \* several methods.  
 \* we have Override preHandle() method here, and checks if the path of request  
 \* starts with /secured. if yes then check for the access token in that.  
 \* and do authorization there.  
 \*/*@Service  
**public class** SecurityInterceptor **extends** HandlerInterceptorAdapter {  
  
 @Autowired  
 **private** SecurityRepository **securityRepository**;  
  
 @Override  
 @org.springframework.transaction.annotation.Transactional(readOnly = **true**)  
 **public boolean** preHandle(HttpServletRequest request,  
 HttpServletResponse response,  
 Object handler) **throws** Exception {  
  
 *// if address starts from /secured. only then check for access tokens.  
 // otherwise user is accessing a public url. no problem.  
 // so if secured. then set the user context for this request. in  
 // Secured class. checkForSecurity in the request* **if** (request.getServletPath().startsWith(**"/secured"**)) {  
 String accessToken = request.getHeader(**"AccessToken"**);  
  
 **if** (accessToken == **null** || accessToken.isEmpty())  
 **throw new** RuntimeException(**"AccessToken header is missing"**);  
  
 */\* fetch from db that actual object.\*/* Security security = **securityRepository**.findByAccessToken(  
 Integer.*valueOf*(accessToken));  
 CurrentRequestUser.*securedUser* = security;  
 }  
  
 **return true**;  
 }  
}

Security Repository

**package** com.locker.security;  
  
**import** com.locker.security.entity.Security;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Repository;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.Query;  
**import** java.util.List;  
  
*/\*\*  
 \* SecurityRepository do interaction with security table in database.  
 \*/*@Repository  
**public class** SecurityRepository {  
  
 @Autowired  
 **private** EntityManager **entityManager**;  
  
 **public void** save(Security sec) {  
 **entityManager**.createNativeQuery(**"INSERT INTO security"** +  
 **" (id, user\_id, access\_token) VALUES (?, ?, ?)"**)  
 .setParameter(1, sec.**id**)  
 .setParameter(2, sec.**userId**)  
 .setParameter(3, sec.**accessToken**)  
 .executeUpdate();  
 }  
  
 **public** Security getByUserId(Integer userId) {  
 Query q = **entityManager**.createNativeQuery(**"SELECT id, user\_id,"** +  
 **" access\_token FROM security WHERE user\_id = ?"**)  
 .setParameter(1, userId);  
 List<Object[]> securities = q.getResultList();  
 **if** (securities.isEmpty())  
 **return null**;  
 **return new** Security(securities.get(0));  
 }  
  
 */\*\*  
 \* delete the security object by its id.  
 \** ***@param id*** *\*/* **public void** deleteById(Integer id) {  
 **entityManager**.createNativeQuery(**"DELETE FROM security WHERE id = ?"**)  
 .setParameter(1, id)  
 .executeUpdate();  
 }  
  
 */\*\*  
 \* Checks the existence of accessToken in the security table and return  
 \* that object too.  
 \** ***@param accessToken*** *\** ***@return*** *fetched security entity  
 \*/* **public** Security findByAccessToken(Integer accessToken) {  
 Query q = **entityManager**.createNativeQuery(**"SELECT** *\** **FROM security"** +  
 **" WHERE access\_token = ?"**)  
 .setParameter(1, Integer.*valueOf*(accessToken));  
  
 List<Object[]> appUsers = q.getResultList();  
  
 **if** (appUsers.isEmpty())  
 **throw new** RuntimeException(**"AccessToken doesn't exists in db."**);  
  
 **return new** Security(appUsers.get(0));  
 }  
}

MVC Configs

**package** com.locker.security;  
  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.context.annotation.Configuration;  
**import** org.springframework.web.servlet.HandlerInterceptor;  
**import** org.springframework.web.servlet.config.annotation.InterceptorRegistry;  
**import** org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;  
  
*/\*\*  
 \* Spring Interceptor registering class. we have used that to register our Security  
 \* Interceptor here.  
 \* our interceptor will be injected in this yourInjectedInterceptor field  
 \* and we will put that in register via addInterceptor method  
 \*/*@Configuration  
**public class** WebMvcConfig **extends** WebMvcConfigurerAdapter {  
  
 @Autowired  
 HandlerInterceptor **yourInjectedInterceptor**;  
  
 @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 registry.addInterceptor(**yourInjectedInterceptor**);  
 }  
}



User

**package** com.locker.user.entities;  
  
*/\*\*  
 \* Database app\_user Entity class  
 \*/***public class** User {  
  
 **public** Integer **id**;  
  
 **public** String **email**;  
 **public** String **password**;  
 **public** String **firstName**;  
 **public** String **lastName**;  
  
 **public** User() {}  
  
 **public** User(Object[] u) {  
 **this**.**id** = (Integer) u[0];  
 **this**.**email** = (String) u[1];  
 **this**.**password** = (String) u[2];  
 **this**.**firstName** = (String) u[3];  
 **this**.**lastName** = (String) u[4];  
 }  
}

User Controller

**package** com.locker.user;  
  
**import** com.locker.user.entities.User;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.http.ResponseEntity;  
**import** org.springframework.web.bind.annotation.\*;  
  
**import** javax.transaction.Transactional;  
**import** java.util.List;  
  
*/\*\*  
 \* UserController helps a person to create a new user and fetch all  
 \* users from the system.  
 \*/*@RestController  
@RequestMapping(**"/public/users"**)  
**public class** UserController {  
  
 @Autowired  
 **private** UserRepository **userRepository**;  
  
 */\*\*  
 \* userRepository call will be made to pull all the users in the system.  
 \** ***@return*** *\*/* @GetMapping  
 @org.springframework.transaction.annotation.Transactional(readOnly = **true**)  
 **public** List<User> fetchAllUsers() {  
 **return userRepository**.getAll();  
 }  
  
 */\*\*  
 \* this endpoint helps to create a new user based on given emaild,  
 \* password, firstName, lastName.  
 \** ***@param user*** *\** ***@return*** *the created user entity object with id associated with that.  
 \*/* @PostMapping  
 @Transactional  
 **public** ResponseEntity<User> saveUser(@RequestBody User user) {  
 User savedUser = **userRepository**.save(user);  
 **return** ResponseEntity.*ok*(savedUser);  
 }  
}

User Repository

**package** com.locker.user;  
  
**import** com.locker.user.entities.User;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.stereotype.Repository;  
  
**import** javax.persistence.EntityManager;  
**import** javax.persistence.Query;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Random;  
  
*/\*\*  
 \* UserRepository is solely designed for database operation based on given request.  
 \* person can pull all users, save a user, or search a user by its email and password.  
 \*/*@Repository  
**public class** UserRepository {  
  
 */\*\*  
 \* database query executor class helps in interacting with the database.  
 \*/* @Autowired  
 **private** EntityManager **entityManager**;  
  
 */\*\*  
 \* this method do fetch all the app\_user table records and create its  
 \* User object and add them all in a list, and finally return that list.  
 \** ***@return*** *\*/* **public** List<User> getAll() {  
  
 Query q = **entityManager**.createNativeQuery(**"SELECT** *\** **FROM app\_user"**);  
 List<Object[]> appUsers = q.getResultList();  
  
 List<User> allUsers = **new** ArrayList<User>(); *// add all db users in this list.* **for** (Object[] a : appUsers) {  
 User u = **new** User(a);  
 allUsers.add(u);  
 }  
 **return** allUsers;  
 }  
  
 */\*\*  
 \* this method will given user object via insert statement in database. and  
 \* returned the saved entity  
 \** ***@param user*** *\** ***@return*** *\*/* **public** User save(User user) {  
 Integer newUserId = **new** Random().nextInt();  
 user.**id** = newUserId;  
  
 **int** resultResponse = **entityManager**.createNativeQuery(**"INSERT INTO app\_user"** +  
 **" (id, email, password, first\_name, last\_name) VALUES (?, ?, ?, ?, ?)"**)  
 .setParameter(1, user.**id**)  
 .setParameter(2, user.**email**)  
 .setParameter(3, user.**password**)  
 .setParameter(4, user.**firstName**)  
 .setParameter(5, user.**lastName**)  
 .executeUpdate();  
 System.***out***.println(**" insert db response "** + resultResponse);  
 **return** user;  
 }  
  
 */\*\*  
 \* this method checks the presence of email/password combination in app\_user table.  
 \* if there is a combination exists it returns the user object. else throw exception.  
 \** ***@param email*** *\** ***@param password*** *\** ***@return*** *fetched user object.  
 \*/* **public** User getByEmailAndPassword(String email, String password) {  
  
 Query q = **entityManager**.createNativeQuery(**"SELECT** *\** **FROM app\_user"** +  
 **" WHERE email=? AND password=?"**)  
 .setParameter(1, email)  
 .setParameter(2, password);  
  
 List<Object[]> appUsers = q.getResultList();  
  
 **if** (appUsers.isEmpty())  
 **throw new** RuntimeException(**"User with this email/password doesn't exists"**);  
  
 **return new** User(appUsers.get(0));  
 }  
}

Util

**package** com.locker;  
  
**import** org.springframework.boot.SpringApplication;  
**import** org.springframework.boot.autoconfigure.SpringBootApplication;  
  
*/\*\*  
 \* the Spring-boot api launcher class.  
 \*/*@SpringBootApplication  
**public class** OnlineLockerApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(OnlineLockerApplication.**class**, args);  
 }  
}

Config

**server:  
 port:** 5000  
  
  
**spring:** *# database confs* **datasource:  
 url:** jdbc:postgresql://192.168.56.101:5432/postgres  
 **platform:** postgres  
 **username:** postgres  
 **password:** postgres  
  
 *# JPA sql generations confs* **jpa:  
 show-sql: true  
 generate-ddl: true  
 hibernate:  
 ddl-auto:** none  
  
 *# Multipart confs* **http:  
 multipart:  
 enabled: true  
 file-size-threshold:** 100KB  
 **max-request-size:** 215MB  
 **max-file-size:** 200MB

DB Scripts

**select** *\** **from** app\_user;  
  
**CREATE TABLE** app\_user (  
 **id INTEGER PRIMARY KEY**,  
 **email VARCHAR**(100) **NOT NULL**,  
 **password VARCHAR**(100) **NOT NULL**,  
 **first\_name VARCHAR**(100),  
 **last\_name VARCHAR**(100)  
);  
  
**CREATE TABLE** security (  
 **id INTEGER PRIMARY KEY**,  
 **user\_id INTEGER**,  
 **access\_token INTEGER**,  
 **FOREIGN KEY** (**user\_id**) **REFERENCES** app\_user (**id**)  
);  
  
**CREATE TABLE** uploaded\_file (  
 **id INTEGER PRIMARY KEY**,  
 **user\_id INTEGER NOT NULL**,  
 **file\_name VARCHAR**(100) **NOT NULL**,  
 **relative\_path VARCHAR**(1000) **NOT NULL**,  
 **FOREIGN KEY** (**user\_id**) **REFERENCES** app\_user (**id**)  
);  
  
**insert into** app\_user (**id**, **email**, **password**, **first\_name**, **last\_name**) **values** (22, **'vipin@gmail.com'**, **'password'**, **'vipin'**, **'gautam'**),  
 (33, **'shivam@gmail.com'**, **'password'**, **'shivam'**, **'sharma'**),  
 (44, **'dalip@gmail.com'**, **'password'**, **'dalip'**, **'singh'**);

**5.3. Testing Approaches**

# **5.3.1. Unit Testing**

Unit Testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

## Unit Testing Tasks

* Unit Test Plan
  + Prepare
  + Review
  + Rework
  + Baseline
* Unit Test Cases/Scripts
  + Prepares
  + Review
  + Rework
  + Baseline
* Unit Test Perform

## Unit Testing Benefits

* Unit testing increases confidence in changing/ maintaining code. If good unit tests are written and if they are run every time any code is changed, we will be able to promptly catch any defects introduced due to the change. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
* Codes are more reusable. In order to make unit testing possible, codes need to be modular. This means that codes are easier to reuse.
* Development is faster. How? If you do not have unit testing in place, you write your code and perform that fuzzy ‘developer test’ (You set some breakpoints, fire up the GUI, provide a few inputs that hopefully hit your code and hope that you are all set.) But, if you have unit testing in place, you write the test, write the code and run the test. Writing tests takes time but the time is compensated by the less amount of time it takes to run the tests; You need not fire up the GUI and provide all those inputs. And, of course, unit tests are more reliable than ‘developer tests’. Development is faster in the long run too. How? The effort required to find and fix defects found during unit testing is very less in comparison to the effort required to fix defects found during system testing or acceptance testing.
* The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels. Compare the cost (time, effort, destruction, humiliation) of a defect detected during acceptance testing or when the software is live.
* Debugging is easy. When a test fails, only the latest changes need to be debugged. With testing at higher levels, changes made over the span of several days/weeks/months need to be scanned.
* Codes are more reliable. Why? I think there is no need to explain this to a sane person.

**5.3.2. Functional Testing**

Functional Testing is a testing technique that is used to test the features/functionality of the system or Software, should cover all the scenarios including failure paths and boundary cases.

User Acceptance Testing: User acceptance testing is defined as a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This testing happens in the final phase of testing before moving the software application to the Market or Production environment.

The main purpose of this testing is to validate the end to end business flow. It does NOT focus on cosmetic errors, Spelling mistakes or System testing. This testing is carried out in a separate testing environment with production like data setup. It is a kind of black box testing where two or more end users will be involved.

**5.3.3. Integration Testing**

Integration testing is the process of testing the interface between two software units or module. It’s focus on determining the correctness of the interface. The purpose of the integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed

Integration test approaches –  
There are four types of integration testing approaches. Those approaches are the following:

1. Big-Bang Integration Testing –  
It is the simplest integration testing approach, where all the modules are combining and verifying the functionality after the completion of individual module testing. In simple words, all the modules of the system are simply put together and tested. This approach is practicable only for very small systems. If once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. So, debugging errors reported during big bang integration testing are very expensive to fix.

Advantages:

* It is convenient for small systems.

Disadvantages:

* There will be quite a lot of delay because you would have to wait for all the modules to be integrated.
* High risk critical modules are not isolated and tested on priority since all modules are tested at once.

2. Bottom-Up Integration Testing –  
In bottom-up testing, each module at lower levels is tested with higher modules until all modules are tested. The primary purpose of this integration testing is, each subsystem is to test the interfaces among various modules making up the subsystem. This integration testing uses test drivers to drive and pass appropriate data to the lower level modules.

Advantages:

* In bottom-up testing, no stubs are required.
* A principle advantage of this integration testing is that several disjoint subsystems can be tested simultaneously.

Disadvantages:

* Driver modules must be produced.
* In this testing, the complexity that occurs when the system is made up of a large number of small subsystem.

3. Top-Down Integration Testing –  
Top-down integration testing technique used in order to simulate the behaviour of the lower-level modules that are not yet integrated.In this integration testing, testing takes place from top to bottom. First high-level modules are tested and then low-level modules and finally integrating the low-level modules to a high level to ensure the system is working as intended.

Advantages:

* Separately debugged module.
* Few or no drivers needed.
* It is more stable and accurate at the aggregate level.

Disadvantages:

* Needs many Stubs.
* Modules at lower level are tested inadequately.

4. Mixed Integration Testing –  
A mixed integration testing is also called sandwiched integration testing. A mixed integration testing follows a combination of top down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level module have been coded and unit tested. In bottom-up approach, testing can start only after the bottom level modules are ready. This sandwich or mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. A mixed integration testing is also called sandwiched integration testing.

Advantages:

* Mixed approach is useful for very large projects having several sub projects.
* This Sandwich approach overcomes this shortcoming of the top-down and bottom-up approaches.

Disadvantages:

* For mixed integration testing, require very high cost because one part has Top-down approach while another part has bottom-up approach.
* This integration testing cannot be used for smaller system with huge interdependence between different modules.

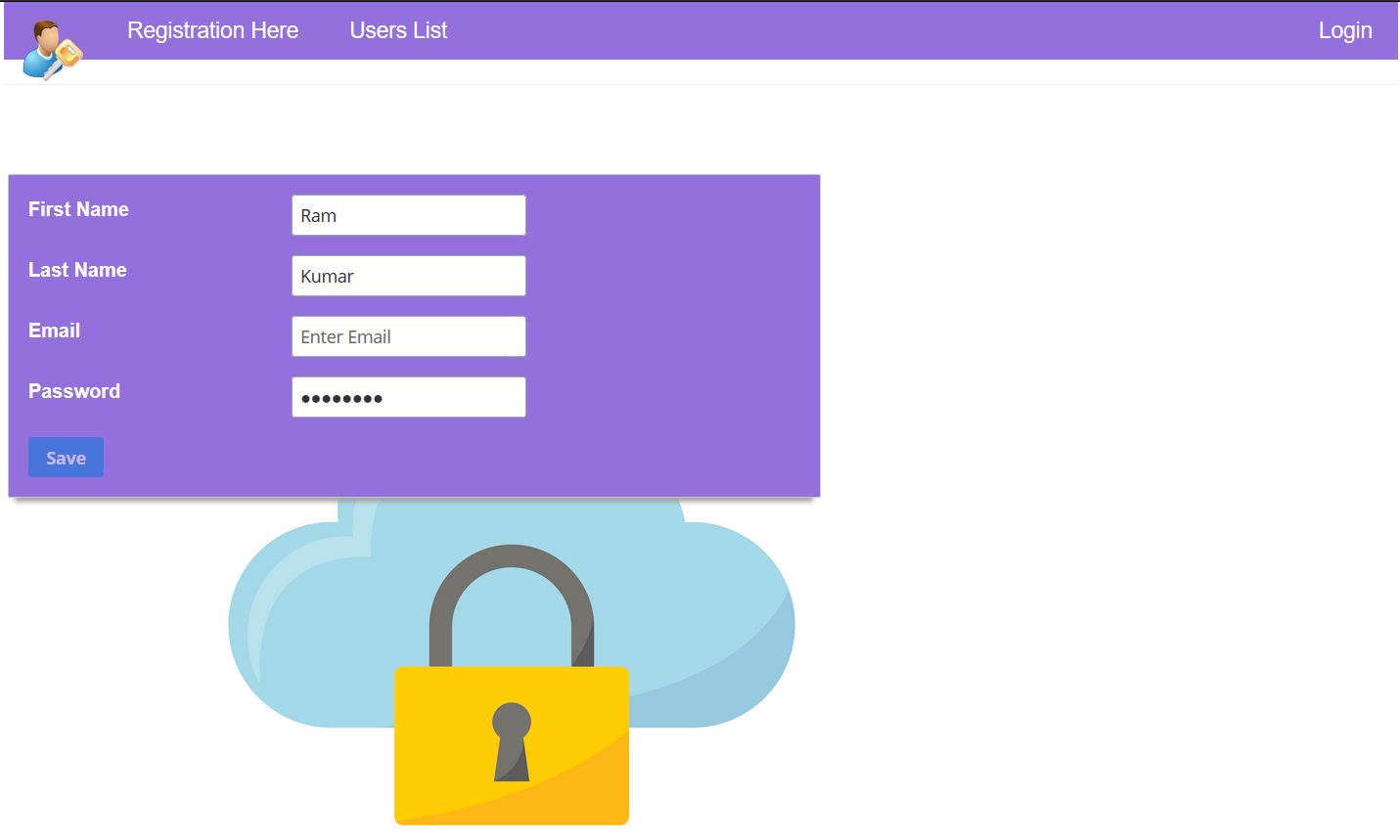
**6. RESULTS AND DISCUSSION**

**6.1. Test Reports**

*Registration Form*

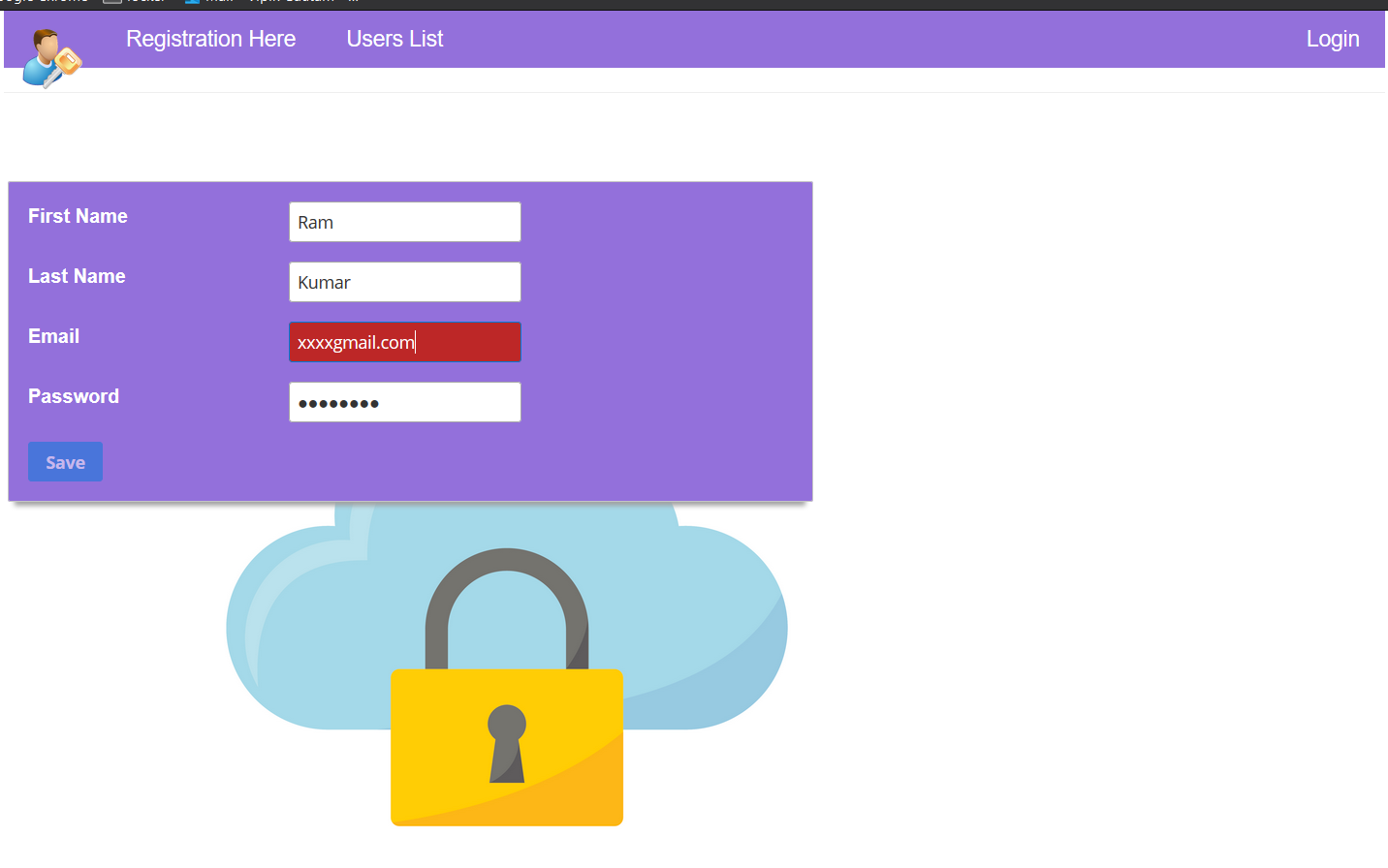
* Registration form only allows to register if below conditions are met
* First Name – Should be at least 3 Char.
* Last Name- Should be at least 3 Char.
* Email- valid email address consists of an email prefix and an email domain, both in acceptable formats. The prefix appears to the left of the @ symbol.
* Password- Minimum 8 letter.

*Below are the results when user attempted to bypass the form validation.*



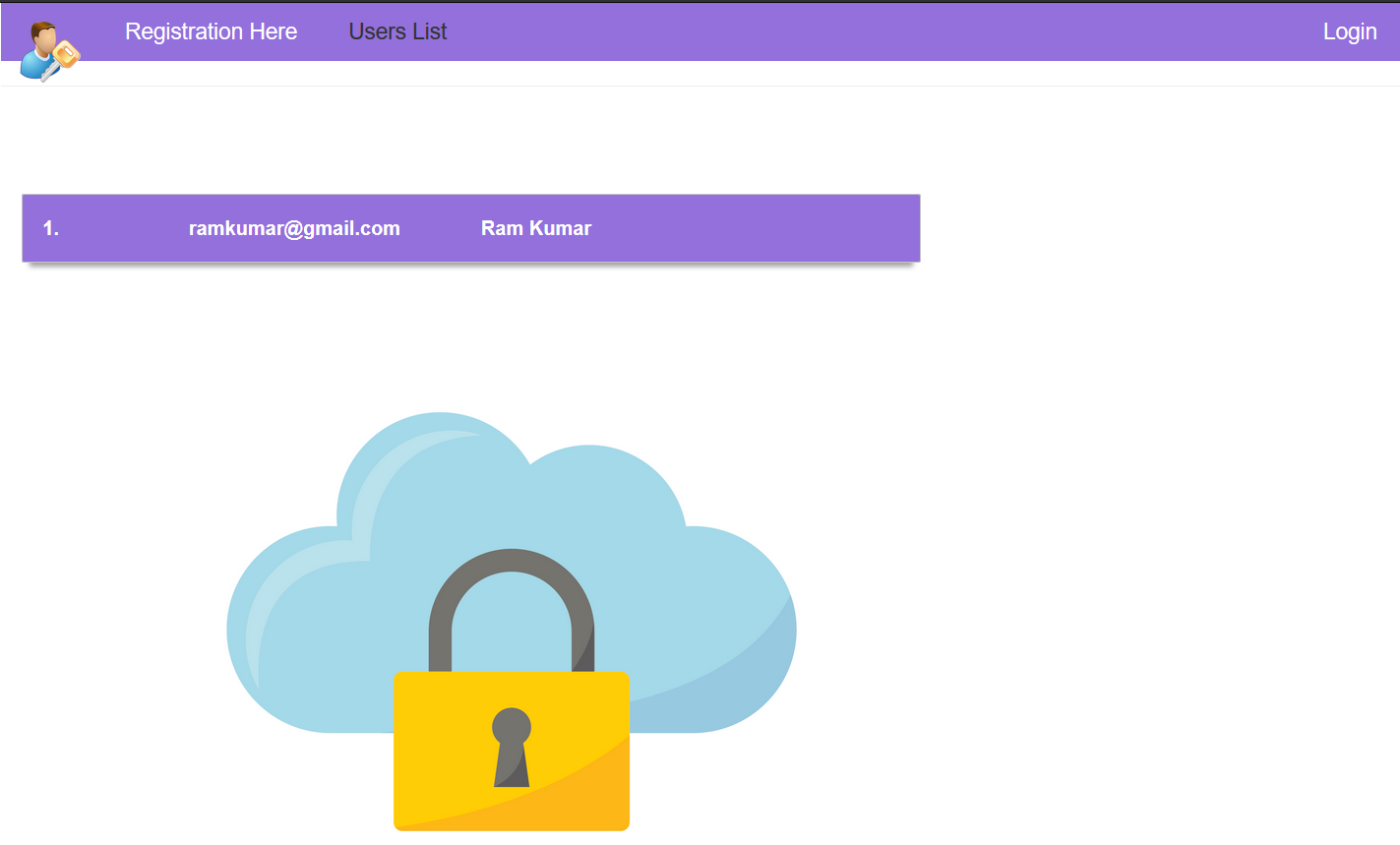
*Disabled Save button So user is unable to save without entering the email.*

*Result when tried to register with invalid email.*



*Save button is still disabled because of having error in the email field.*

*Result when entered all valid values.*

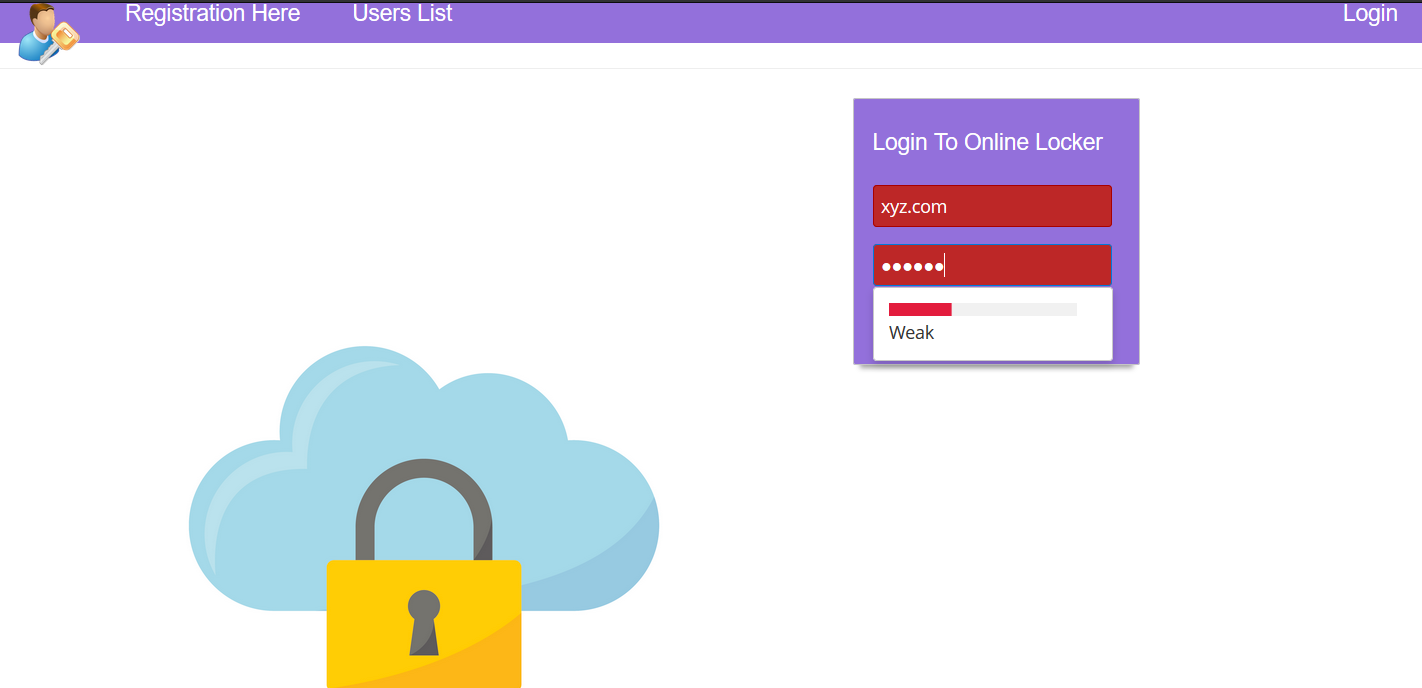


*User Ram successfully logged in.*

*Login Form*

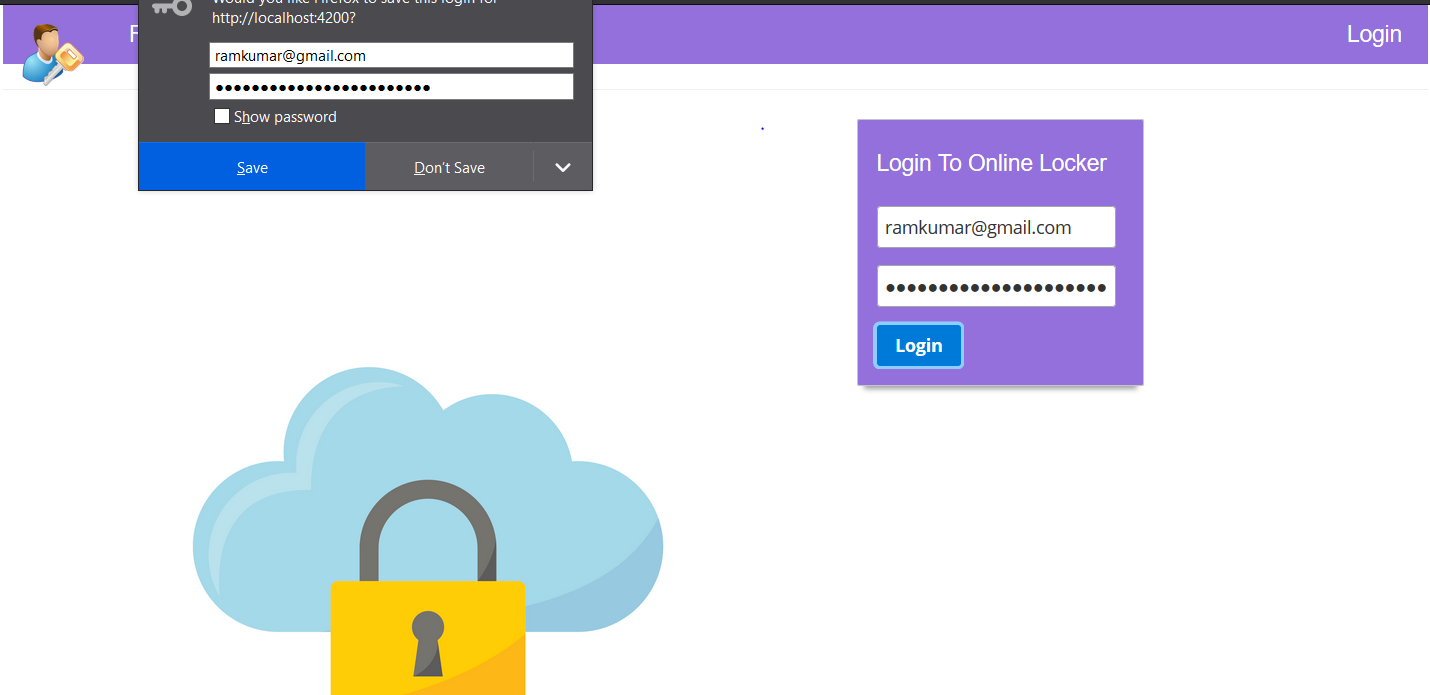
* Email- valid email address consists of an email prefix and an email domain, both in acceptable formats. The prefix appears to the left of the @ symbol.
* Password- Minimum 8 letter.
* User Type- Should be a registered user.

*Result when attempted to login with and invalid email and password(Less than 8 letter).*



*Result-*

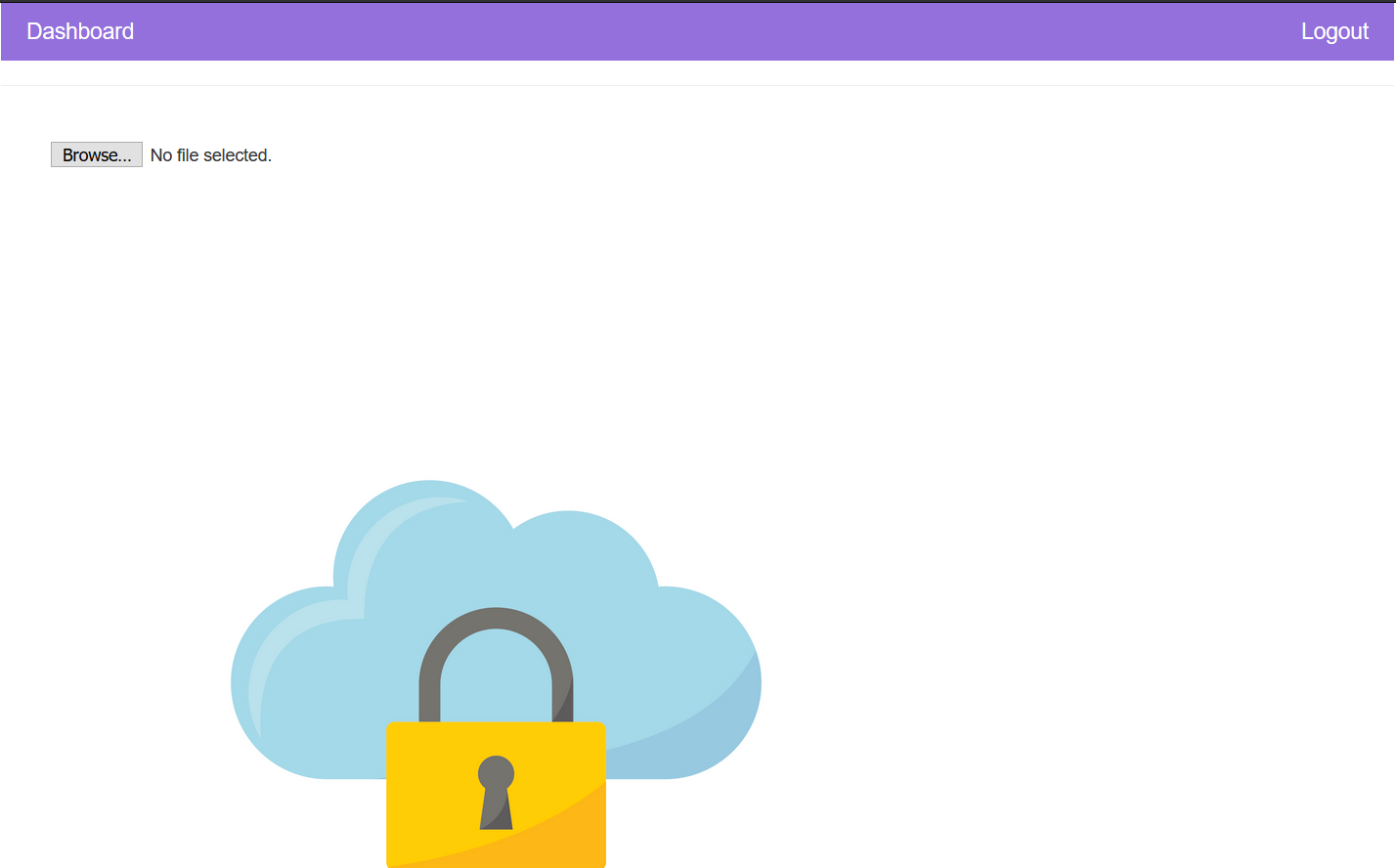
*When registered user Ram attempted to login with his valid email which he registered before that to with invalid email.*



*Ram is not routed to his account.*

*Result-*

*When Ram tried to login with valid email and password.*



*Ram has been logged in.*

**6.2. User Documentation**

Digital Locker is a web based Application where users are allowed to register and create there account and store there documents i.e data digitally, thus it provides a service and this ensures that they have readily access of their database on the go.

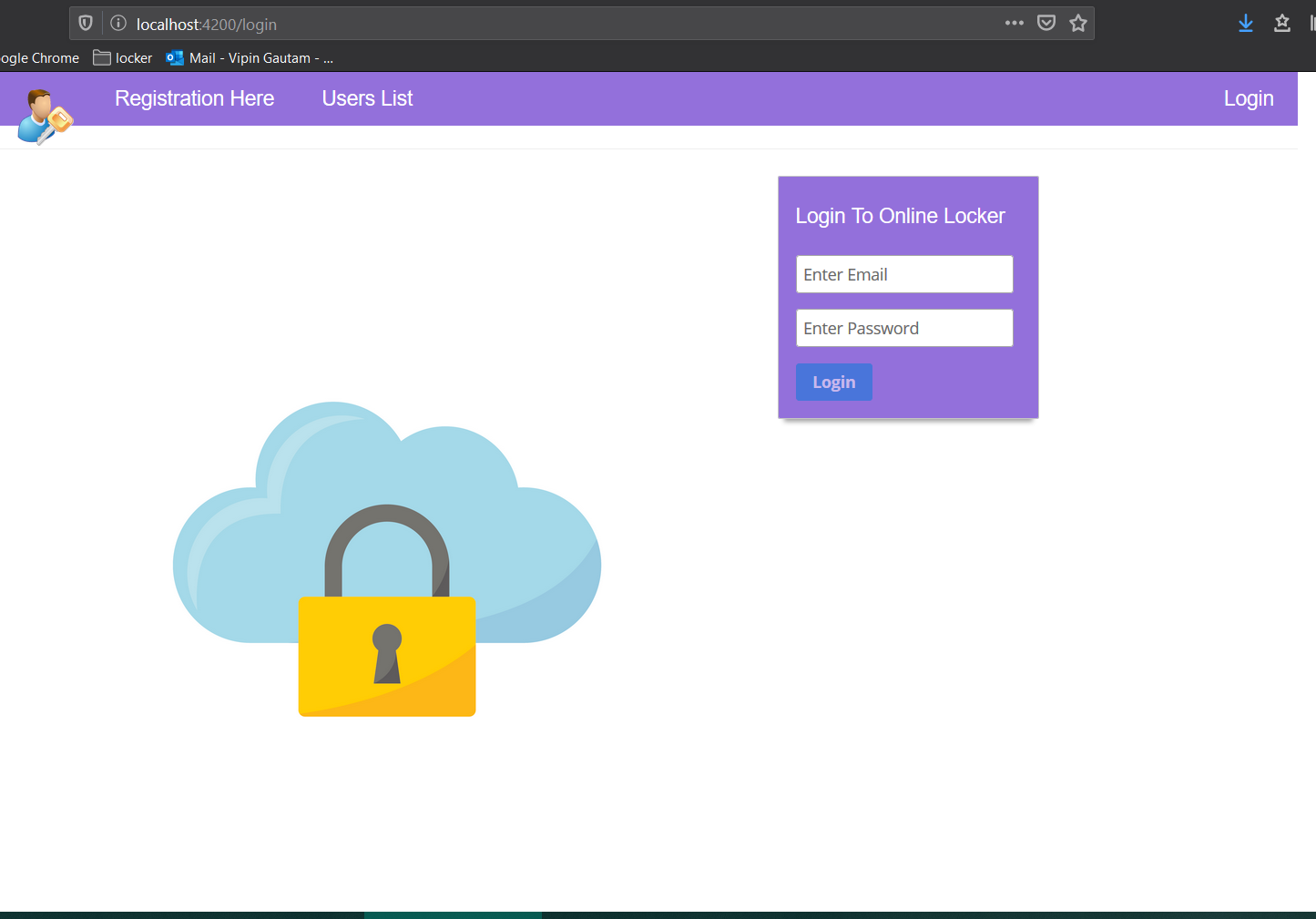
|  |  |
| --- | --- |
| Login: | Every registered will have the option of login to his/her account Dashboard page will show the files/documents which has been upload by user so |
| Register | This function allows the users to create their account on web application |
| Delete | A user can delete the files they’ve uploaded |
| Dashboard: | Apart from Deleting and Uploading the files user will also have the option to download file. |
| Upload | User can upload the File |
| Navigation | Whilst the user is on web app they can navigate with the help of navigation bar |

**Table 6: Features**

In this section we’ve given detailed information about each listed process above, components which are part of application and User Interface of built web Application which will give you more clarity about each function and how the user experience is enhanced.

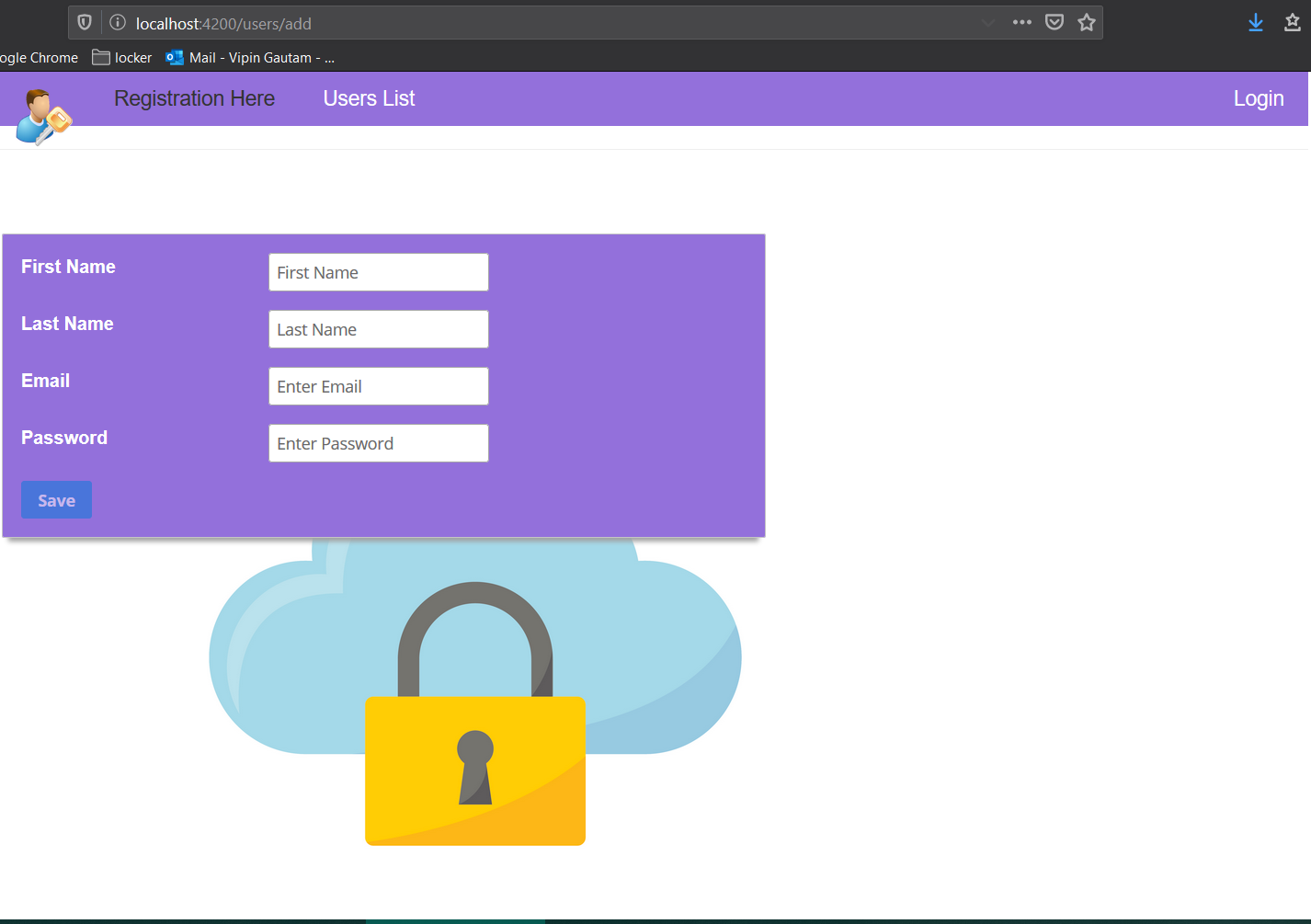
Login: The application is designed in such a way that when a new user opens the website for the first time he/she will be first redirected to login page, which is obviously the prime job. Here user is just required to enter his/her email and password. If fields are left empty then no action can be taken which is due to security so it’s mandatory to have valid email and password. By mandatory we mean valid email which terminates with right domain and valid password now if this is not the issue with user the comes to next part of login where user will be verified by the server whether he/she is really registered on our platform.

Here we have added a snapshot as how the landing page will look like when user first opens our web app.

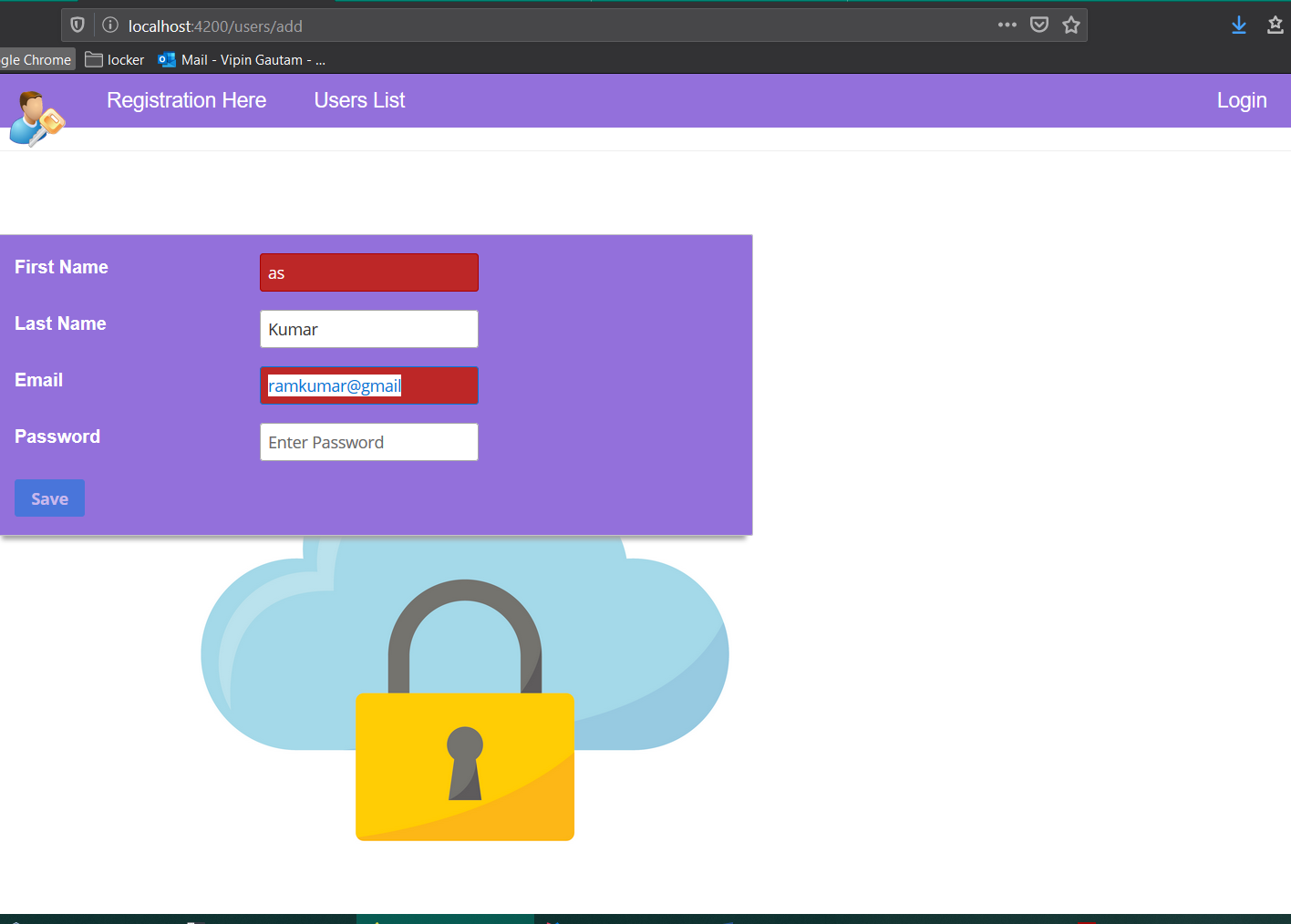


The Login button is currently disabled it’s only because user has not entered the credentials. Now let see how will it look when user has entered valid credentials will there be any change?

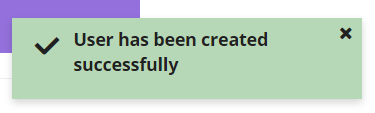
Registration: The is one of the core feature of this application, imagine a instance when user is not a member of website i.e not registered so this case has to be taken care. Hence we have developed a form which focuses on user registration. Here user would need to enter his First Name, Last Name, Valid Email and Password. As soon as user clicks on save he/she will become member of the locker and can therefor he must go to step 1 and perform login and further actions so to say.



As we can see above the Save button is disabled which is important so such request to servicer should be prohibited because we need to maintain the integrity of our database and all the requests which are being password to server. Let see how will the app look when user has entered valid details in above fields.

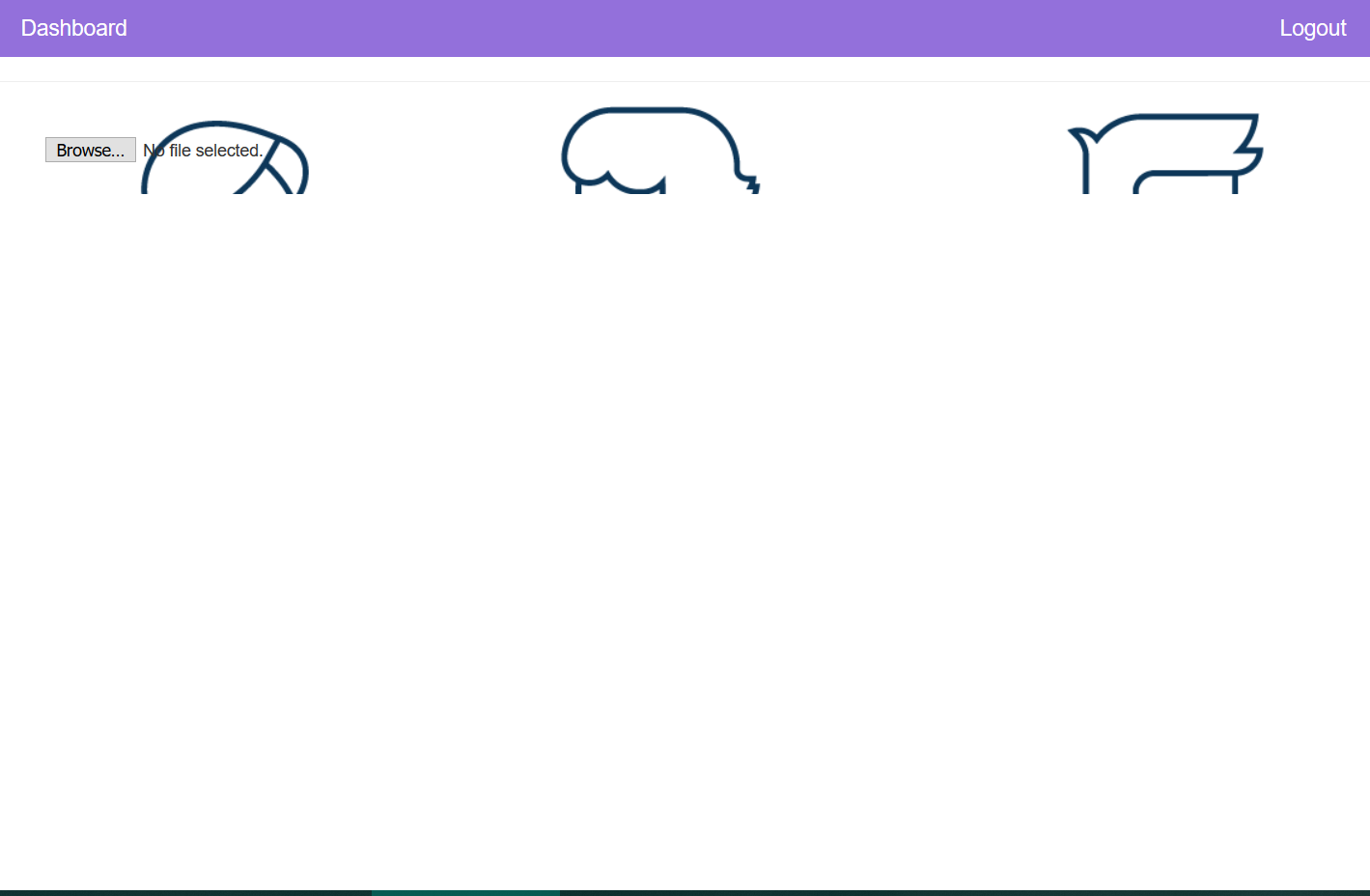


Above as soon as the user hits Save user will see below msg at the right top of app



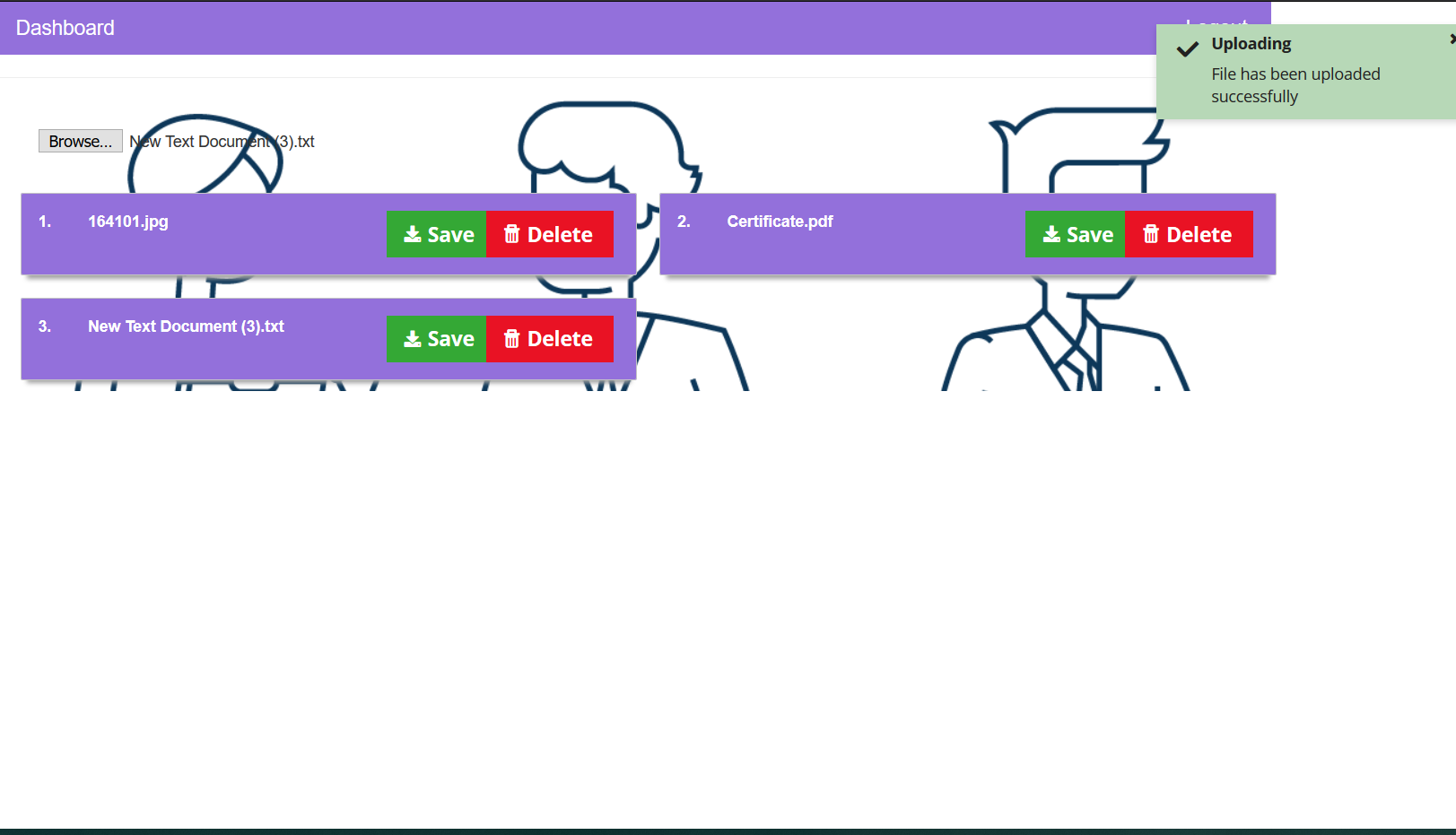
Dashboard: Dashboard is basically responsible for listing all the files user have uploaded so far and giving the option to upload something if he/she wishes to.

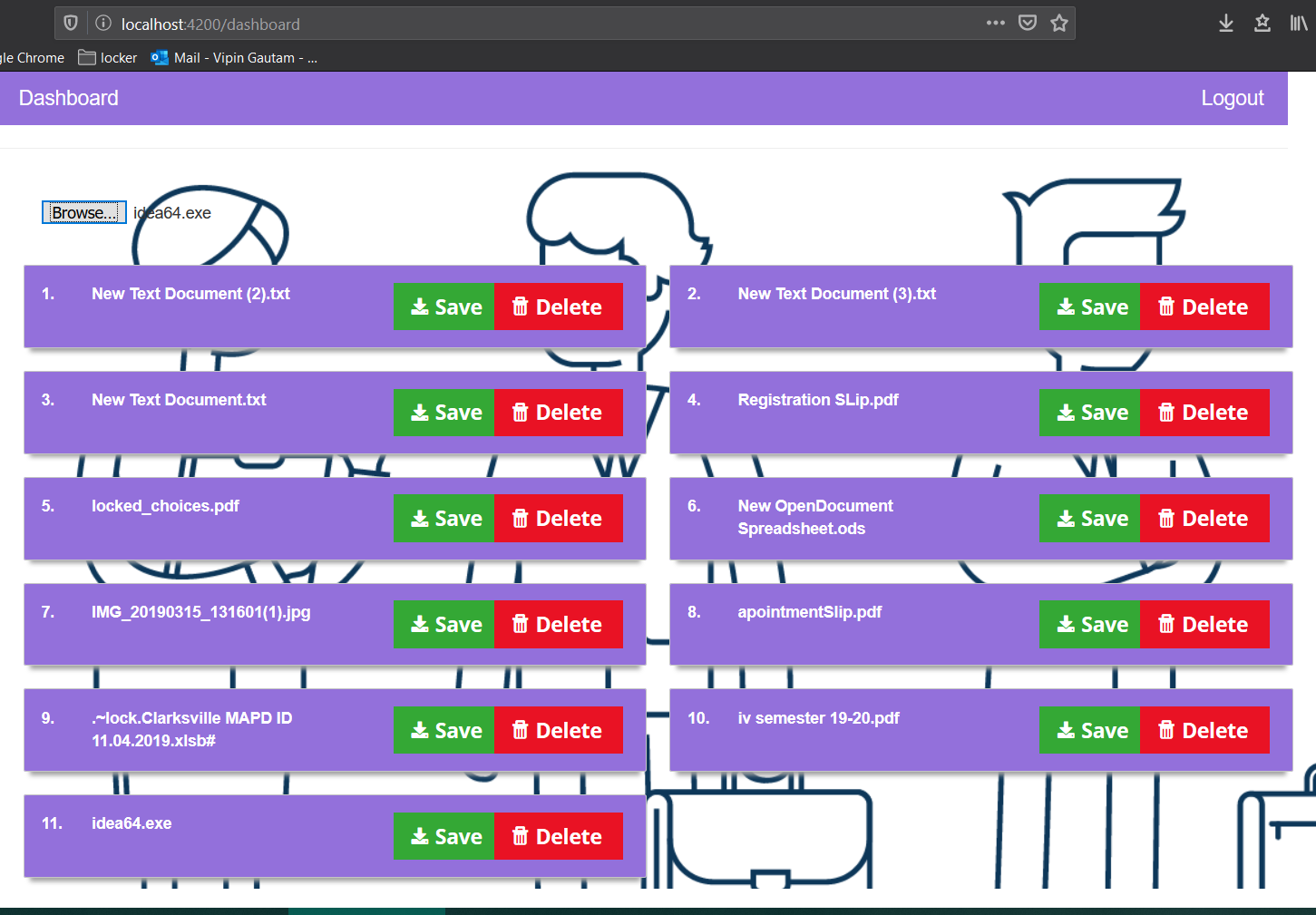
Let us see cases when user has just logged in for the first time.

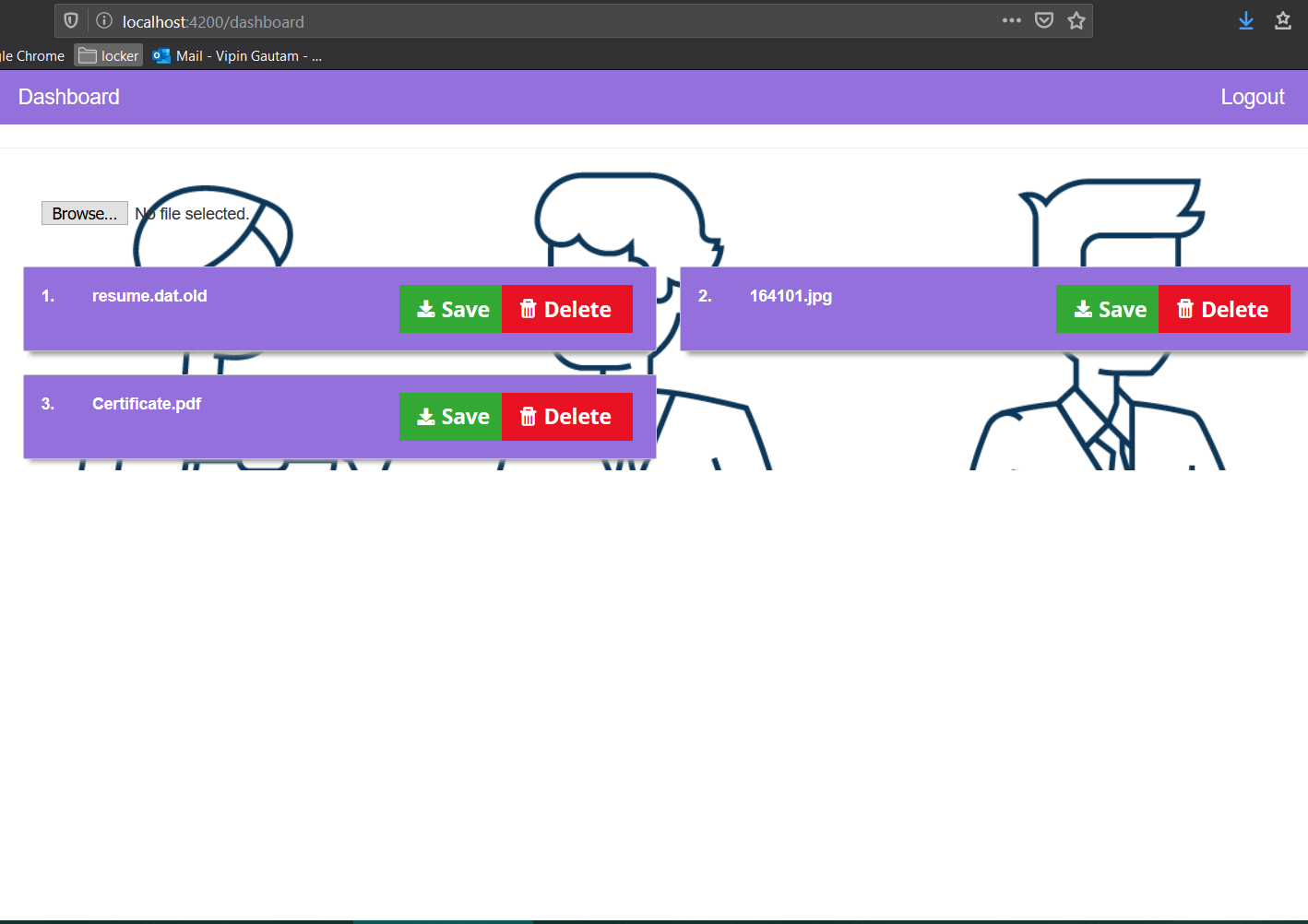


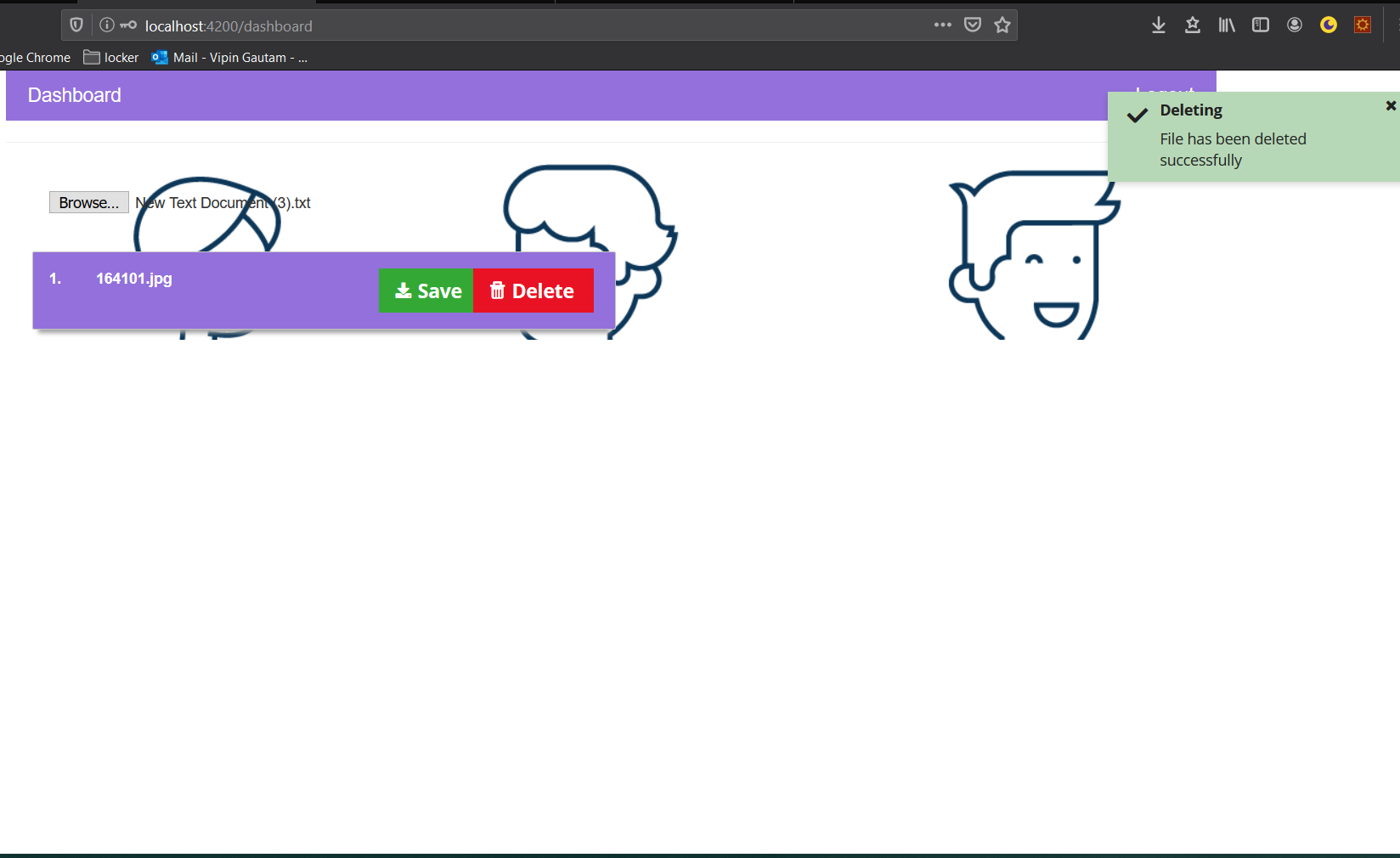
So it’s apparent that the Dashboard function brings out two more functions logout and Browse.

Browser function simple gives the access to user that he/she can upload his documents let us see how will it look after user has selected some file from directory or uploaded something what will the app bring after he has uploaded. We shall see it in each and every process of uploading

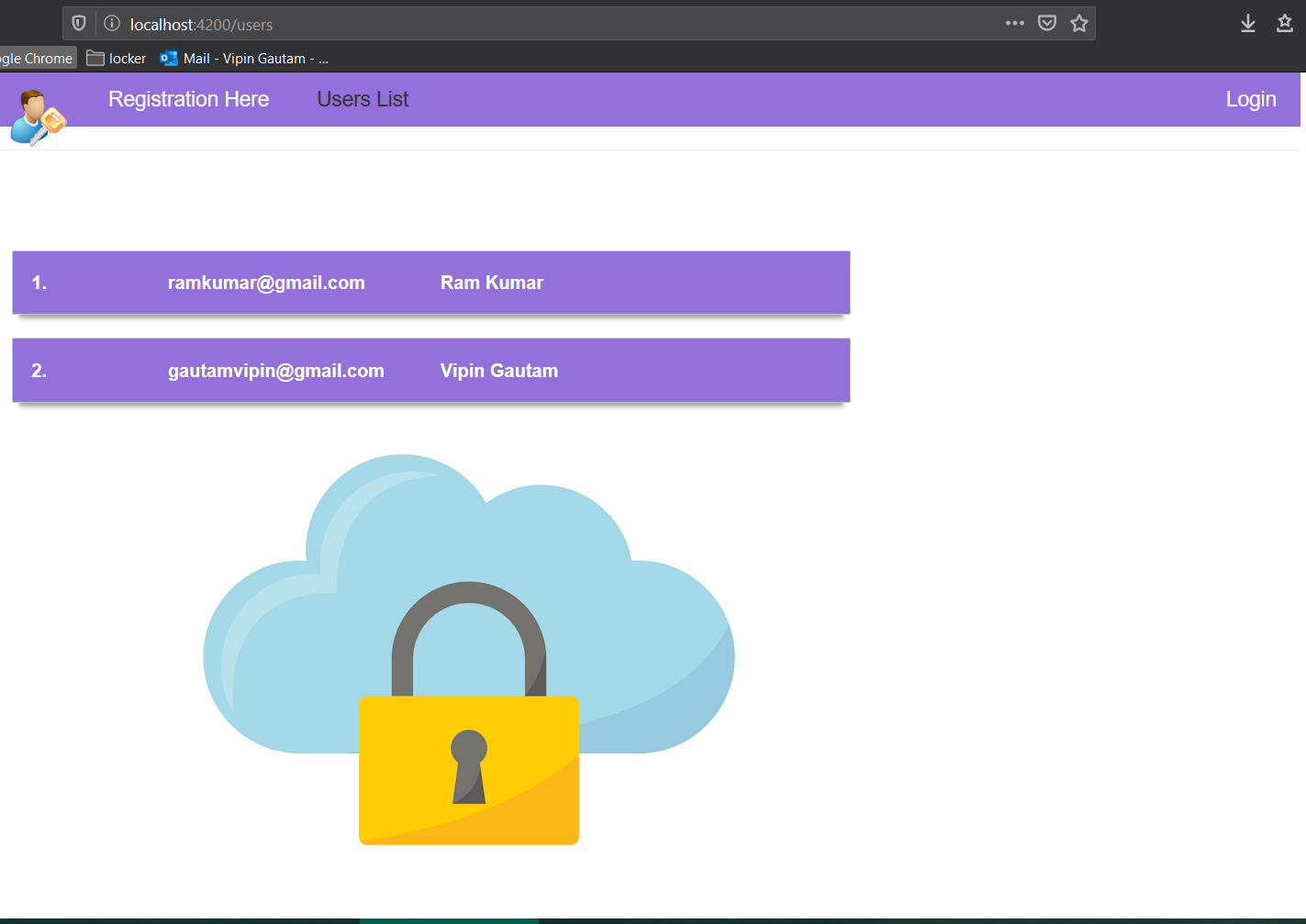








In the above snapshots of the UI we have tried to explain how Upload and Delete function is working in Dashboard for registered user.



This is one of the feature of digital locker which has future scope here we can have admin section on can control the above list of users. Such as only admin is allowed to view list of users and he can even delete the user or may be lock if he sees any incorrect activity.

**7. CONCLUSIONS**

**7.1. Conclusion**

Digital Locker is an web based application that has been developed for users who are registered on our platform to store online documents safely. The registered user can save online all government issued documents on the secure server and retrieve his files on the go whenever required. The locker empowers the users with functionality of registration, login viewing his content on specially designed dashboard and gives option to logout safely. Starting from the introduction of the project till the conclusion part we have been focused on the data security and integrity as it’s one of the core feature of this application. We have explained how Locker’s implementation is done what kind of databases and technologies have been used to design this application. This application has a power to server user of any kind by ensuring that content is secure on online server though the project can certainly be extended as there can be many more features added. There has been immense learning after the completion of the project.

**7.2. Future Scope of the Project**

The Digital Locker can be certainly made live which will serve customer of its kind. It will help those users specially who carry their documents day to day thus it will reduce their effort significantly.

The project can be extended and has very good future scope. There can be many more functionalities added and more options like we can also build a profile for administer who would control the users and maintain the day in day out jobs on website who would handle the user request. The primary goal is to store the users content but what if the database become so large In that case we may have to switch to different data structures and that perspective can certainly be taken care and what if the traffic on website is huge in that case access time can also play big role, what if the user wants to upload huge files on server and it may be possible that we can some users doing malpractice by hiding their identity as users are allowed to register just with email. These are couple of points which definitely can be extended as a future scope of this project.

**REFERENCES**

1. [Angular. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/mDGj) <https://angular.io/docs>

2. [PostgreSQL: About. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/HZQW) <https://www.postgresql.org/about/>

3. [PostgreSQL: About. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/KmkV) <https://www.postgresql.org/about/>

4. [SQL - RDBMS Concepts - Tutorialspoint. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/tyLF) <https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm>

5. [Spring Guides. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/XBQg) <https://spring.io/guides>

6. [Porter B, van Zyl J, Lamy O. Maven – Welcome to Apache Maven. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/51qV) <https://maven.apache.org/>

7. [PERT Estimation Technique - Tutorialspoint. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/8hoU) <https://www.tutorialspoint.com/management_concepts/pert_estimation_technique.htm>

8. [Gantt Chart Tool - Tutorialspoint. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/Qe9F) <https://www.tutorialspoint.com/management_concepts/gantt_chart_tool.htm>

9. [DBMS ER model concept - javatpoint. In: www.javatpoint.com [Internet]. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/dDWq) <https://www.javatpoint.com/dbms-er-model-concept>

10. [OOAD - UML Behavioural Diagrams - Tutorialspoint. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/7sVn) <https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_uml_behavioural_diagrams.htm>

11. [ArrayList in Java - javatpoint. In: www.javatpoint.com [Internet]. [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/aU7G) <https://www.javatpoint.com/java-arraylist>

12. [Contributors to Wikimedia projects. Spiral model - Wikipedia. In: Wikimedia Foundation, Inc. [Internet]. 20 Nov 2002 [cited 21 Nov 2019]. Available:](http://paperpile.com/b/Rff4HW/3b6p) <https://en.wikipedia.org/wiki/Spiral_model>

**GLOSSARY**

* Abstract

A Java keyword used in a class definition to specify that a class is not to be instantiated, but rather inherited by other classes. An abstract class can have abstract methods that are not implemented in the abstract class, but in subclasses.

* Abstract class

A class that contains one or more [abstract methods](https://www.oracle.com/technetwork/java/glossary-135216.html#abstract_method), and therefore can never be instantiated. Abstract classes are defined so that other classes can extend them and make them concrete by implementing the abstract methods.

* Abstract method

A method that has no implementation.

* Abstract Window Toolkit (AWT)

A collection of graphical user interface (GUI) components that were implemented using native-platform versions of the components. These components provide that subset of functionality which is common to all native platforms. Largely supplanted by the Project Swing component set. See also [Swing](https://www.oracle.com/technetwork/java/glossary-135216.html#swing) .

* Access control

The methods by which interactions with resources are limited to collections of users or programs for the purpose of enforcing integrity, confidentiality, or availability constraints.

* ACID

The acronym for the four properties guaranteed by transactions: atomicity, consistency, isolation, and durability.

* Actual parameter list

The arguments specified in a particular method call. See also [formal parameter list](https://www.oracle.com/technetwork/java/glossary-135216.html#formal) .

* API

Application Programming Interface. The specification of how a programmer writing an application accesses the behavior and state of classes and objects.

* Bean

A reusable software component that conforms to certain design and naming conventions. The conventions enable beans to be easily combined to create an application using tools that understand the conventions.

* DOM

Document Object Model. A tree of objects with interfaces for traversing the tree and writing an XML version of it, as defined by the W3C specification.

* HTML

HyperText Markup Language. This is a file format, based on SGML, for hypertext documents on the Internet. It is very simple and allows for the embedding of images, sounds, video streams, form fields and simple text formatting. References to other objects are embedded using URLs.

* Import

A Java keyword used at the beginning of a source file that can specify classes or entire packages to be referred to later without including their package names in the reference.

* JPA

Java Persistence Api

* Dependency Injection

Pattern to hand a component's dependency to the component from outside, freeing the component to lookup the dependant itself. For more information see

* Binding

Generally, the practice of setting a variable or property to a data value. Within Angular, typically refers to [data binding](https://angular.io/guide/glossary#data-binding), which coordinates DOM object properties with data object properties.

* Class decorator

A [decorator](https://angular.io/guide/glossary#decorator) that appears immediately before a class definition, which declares the class to be of the given type, and provides metadata suitable to the type.

* The following decorators can declare Angular class types:
* @[Component](https://angular.io/api/core/Component)()
* @[Directive](https://angular.io/api/core/Directive)()
* @[Pipe](https://angular.io/api/core/Pipe)()
* @[Injectable](https://angular.io/api/core/Injectable)()
* @[NgModule](https://angular.io/api/core/NgModule)()

## Command-line interface (CLI)

## The [Angular CLI](https://angular.io/cli) is a command-line tool for managing the Angular development cycle.

## Data binding

## A process that allows apps to display data values to a user and respond to user actions (such as clicks, touches, and keystrokes).

## DI token

## A lookup token associated with a dependency [provider](https://angular.io/guide/glossary#provider), for use with the [dependency injection](https://angular.io/guide/glossary#di) system.

## Directive

## A class that can modify the structure of the DOM or modify attributes in the DOM and component data model. A directive class definition is immediately preceded by a @[Directive](https://angular.io/api/core/Directive)() [decorator](https://angular.io/guide/glossary#decorator) that supplies metadata.

## Form validation

## A check that runs when form values change and reports whether the given values are correct and complete, according to the defined constraints. Reactive forms apply [validator functions](https://angular.io/guide/form-validation#adding-to-reactive-forms). Template-driven forms use [validator directives](https://angular.io/guide/form-validation#adding-to-template-driven-forms).

## Interpolation

## A form of property [data binding](https://angular.io/guide/glossary#data-binding) in which a [template expression](https://angular.io/guide/glossary#template-expression) between double-curly braces renders as text.

## Constraints

## A [database constraint](https://www.lifewire.com/not-null-constraints-1019824) is a set of rules that define valid data. The primary constraints are:

## Functional Dependency-A [functional dependency constraint](https://www.lifewire.com/functional-dependency-definition-1019257) helps to ensure data validity, and exists when one attribute determines the value of another, described as A -> B which means that the value of A determines the value of B, or that B is functionally dependent on A.

## Key

## A key is a database field whose purpose is to uniquely identify a record. Keys help enforce data integrity and avoid duplication

## Null

## The value [null](https://www.lifewire.com/all-about-null-values-1019266) is frequently confused to mean none or zero.

**APPENDICES**