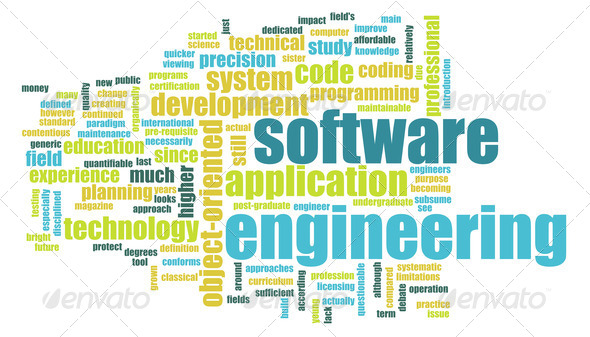
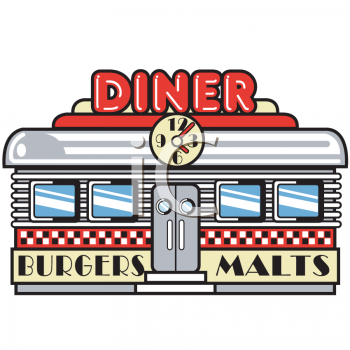
SOFTWARE ENGINEERING THEORY PROJECT  
DESIGN DOCUMENT



**Fall Semester 2016-2017**

**TOPIC:**

Restaurant Service System



**Teacher:**

Prof. Alok Chauhan

**STUDENTS:**

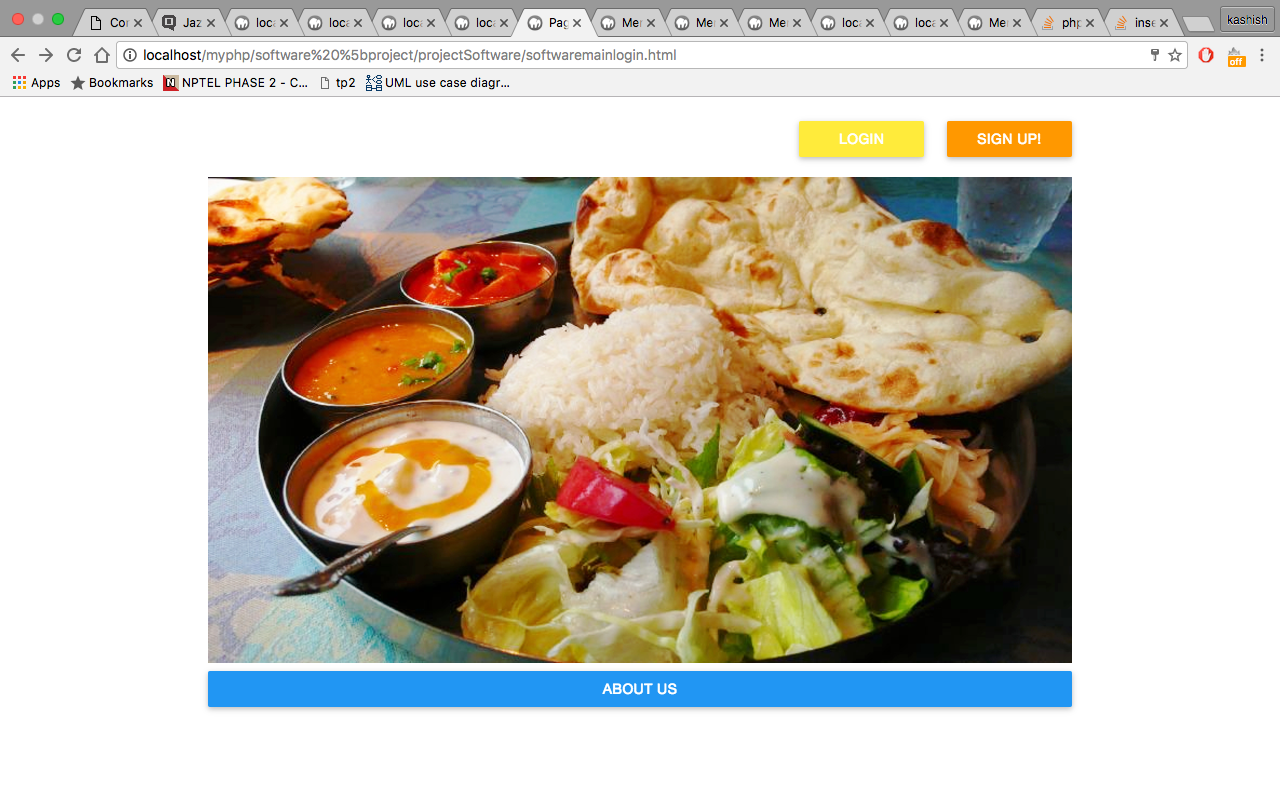
* Osho Agyeya(15BCE1326)
* Kashish Miglani(15BCE1003)
* Sanchay Gupta(15BCE1190)
* Sachin Gopal(15BCE1188)

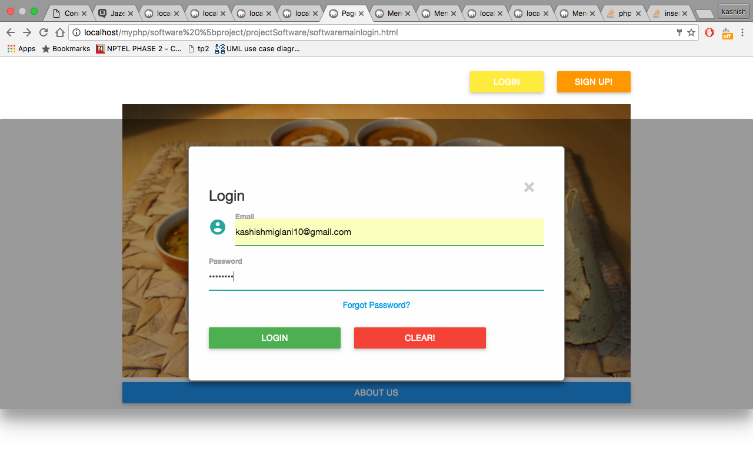
**Design Document**

EXTERNAL DESIGN SPECIFICATIONS:

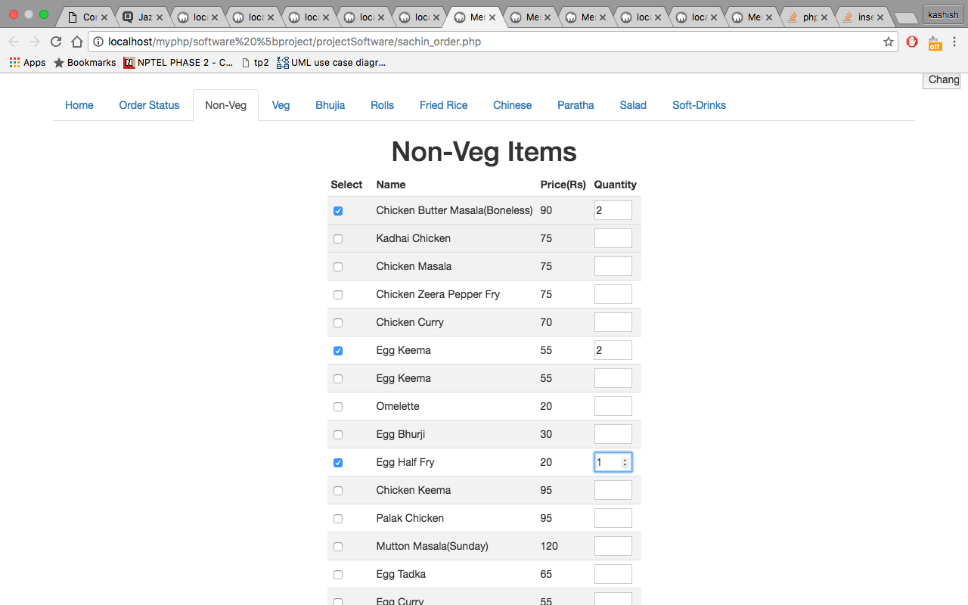
User Displays and report formats:

The user display is lucid and intuitive. The main home screen has been integrated with photos of lip smacking dishes to draw the attention of the user. The buttons are clear and sleek.





The menu has categorized the dishes according to ethnicities.



User command summary:

The various commands that shall be used by the user are:

* Register/create account
* Login
* Choose dishes according to the offerings in the menu
* Calculate the bill
* Delivery options
* Approve/reject order(For admin)
* Change password

Detailed data flow diagrams:

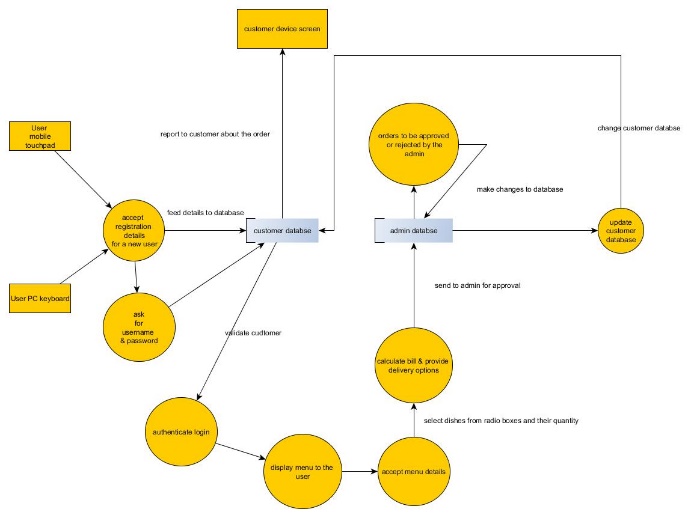
Although the *data flow diagram* (DFD) and related diagrams and information are not a formal part of UML, they can be used to complement UML diagrams and provide additional insight into system requirements and flow. The DFD takes an input-process-output view of a system. That is, data objects flow into the software, are transformed by processing elements, and resultant data

objects flow out of the software. Data objects are represented by labelled arrows, and

transformations are represented by circles (also called bubbles). The DFD is presented

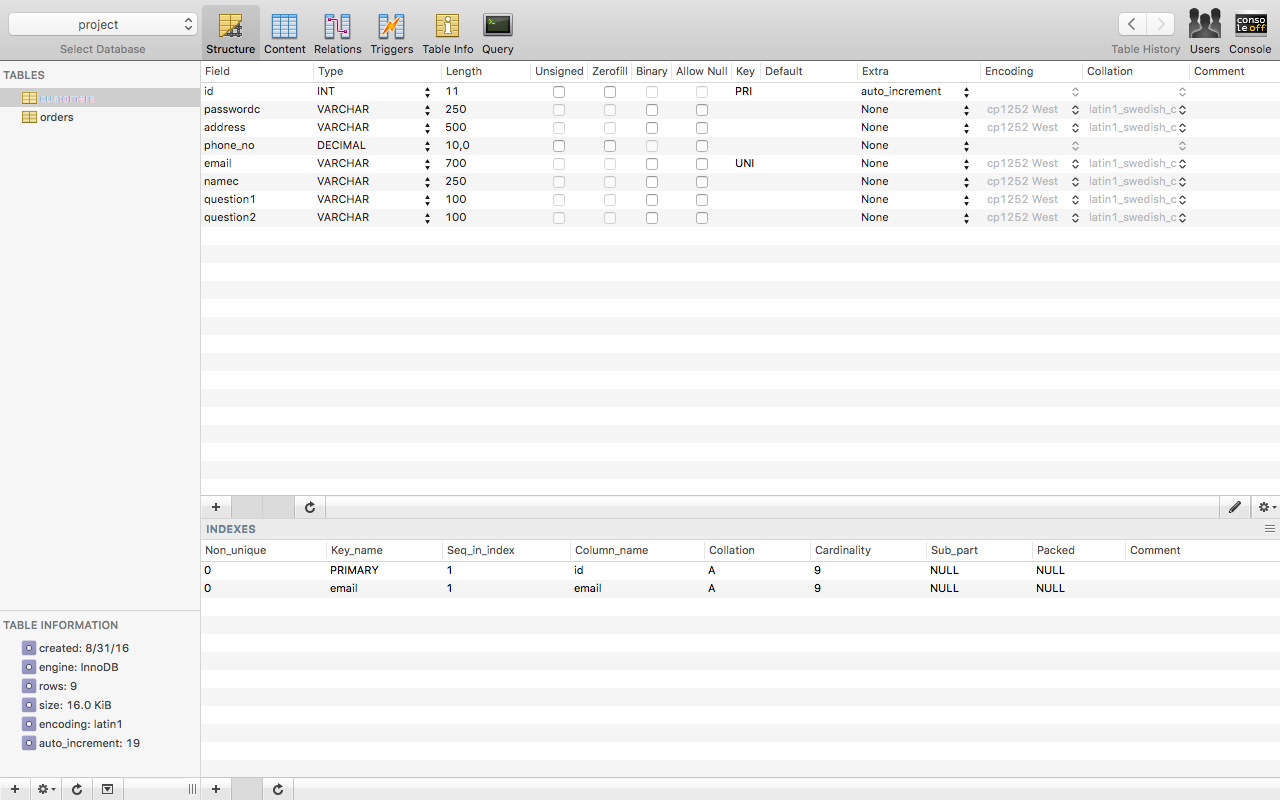
in a hierarchical fashion. That is, the first data flow model (sometimes called

a level 0 DFD or *context diagram*) represents the system as a whole. Subsequent data flow diagrams refine the context diagram, providing increasing detail with each subsequent level.

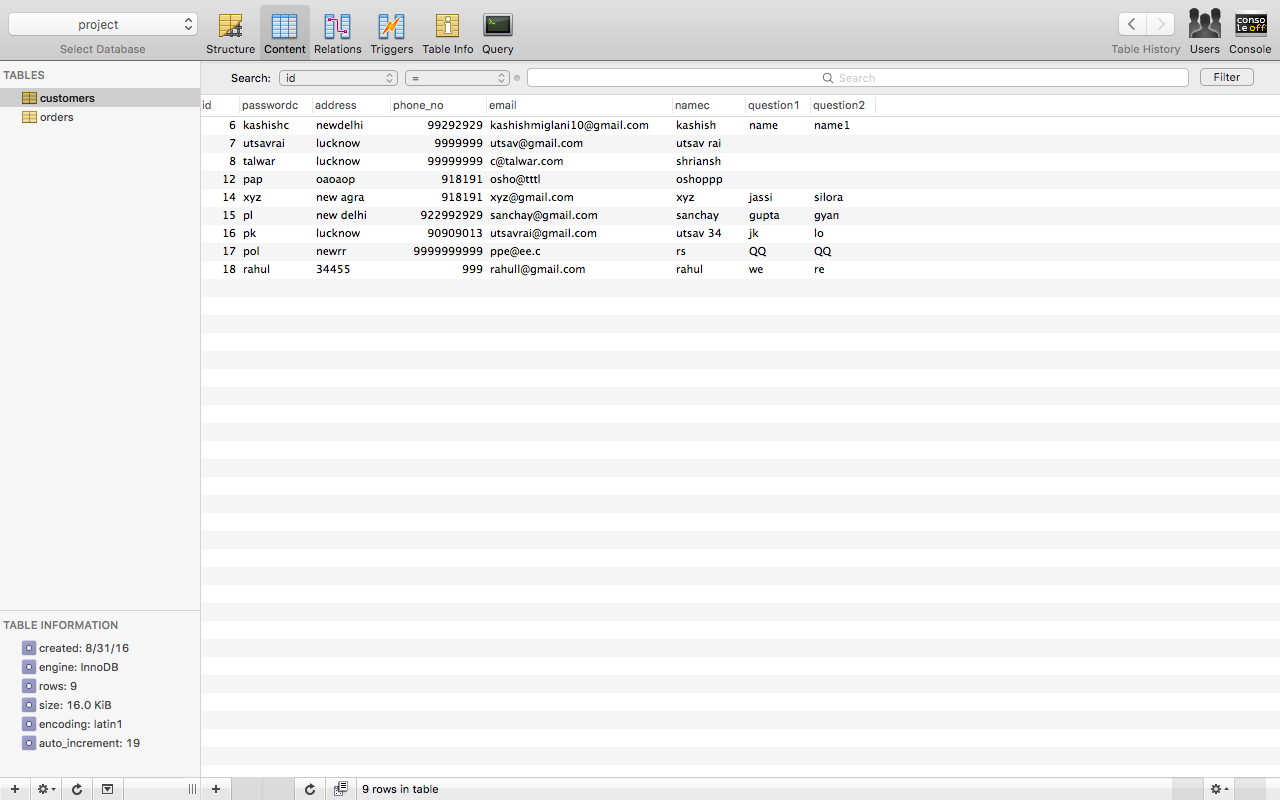


Logical format of database:

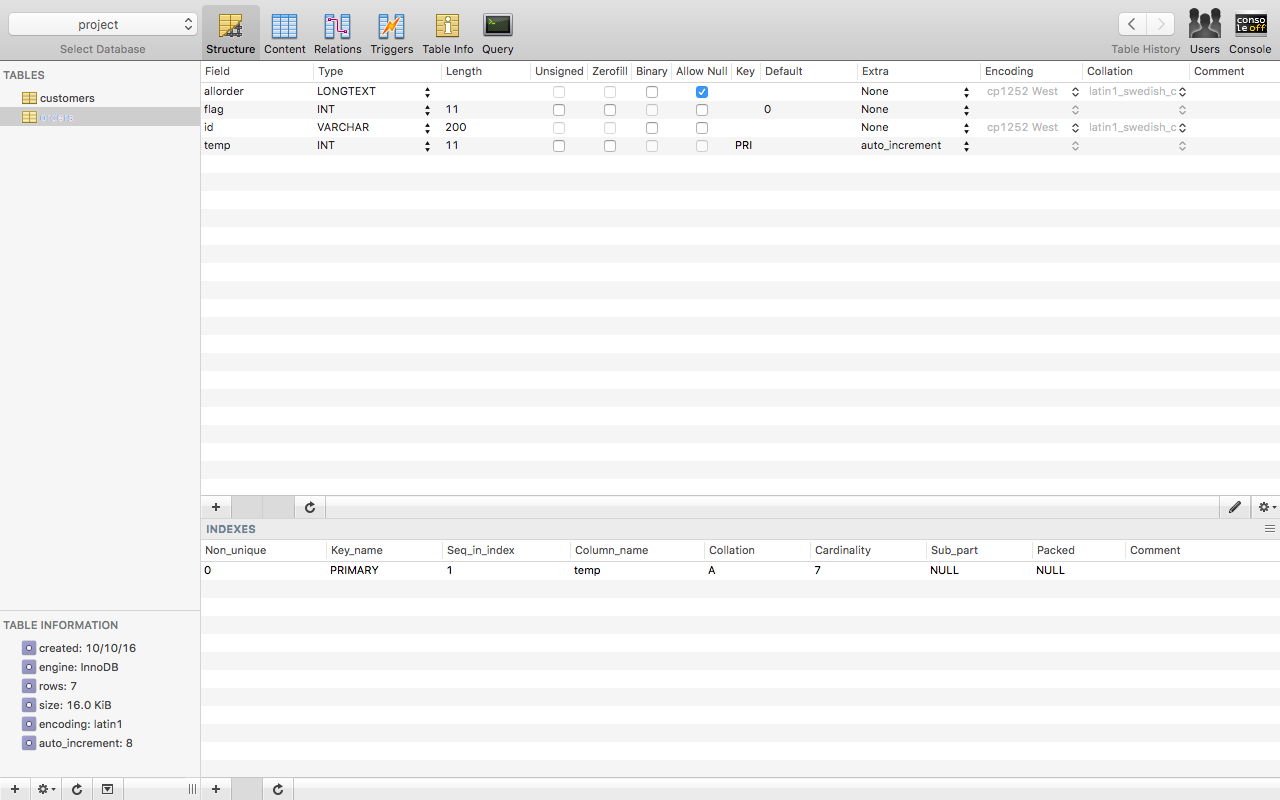
MySQL has been used to create the database “project” which has 2 tables, orders and customers namely.



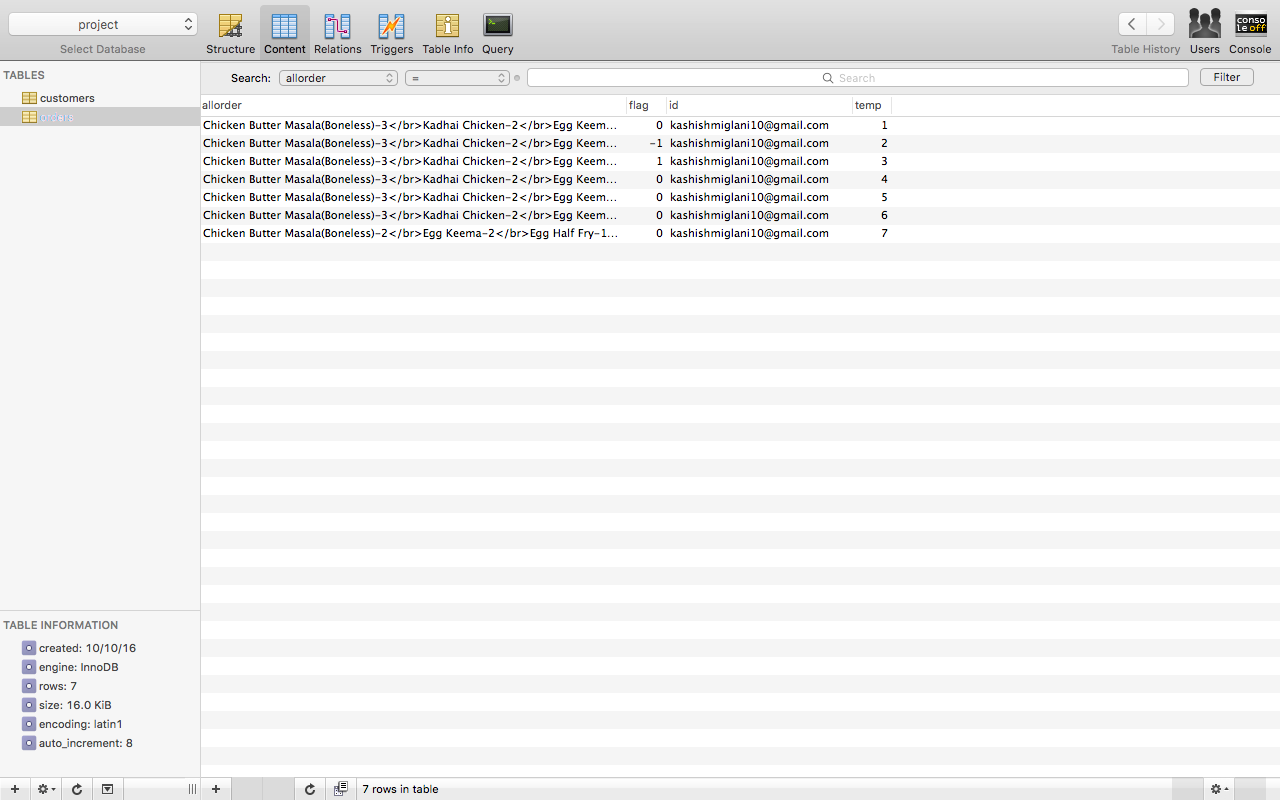
The datatypes for customers table are visible above.



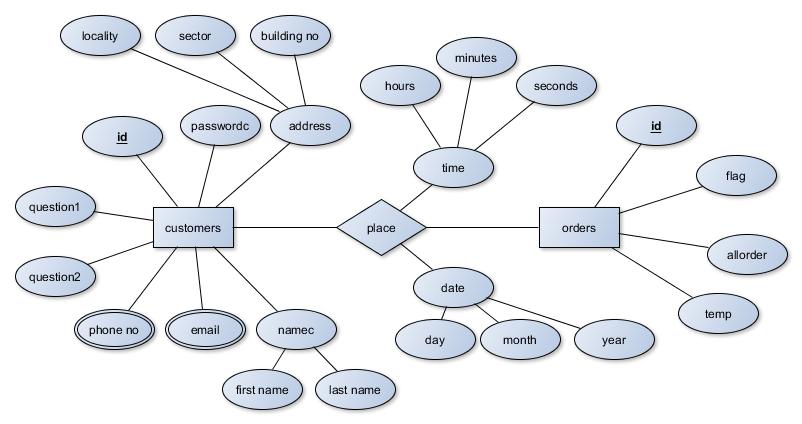
The sample data for customers table are visible above.



The datatypes for customers table are visible above.



The sample data for orders table are visible above.



The ER Diagram is above.

Architectural Design Specifications:

Software architecture:

Architecture Genre:  
  
\*Commercial,

\*Communications

\*Financial

Architecture Style:

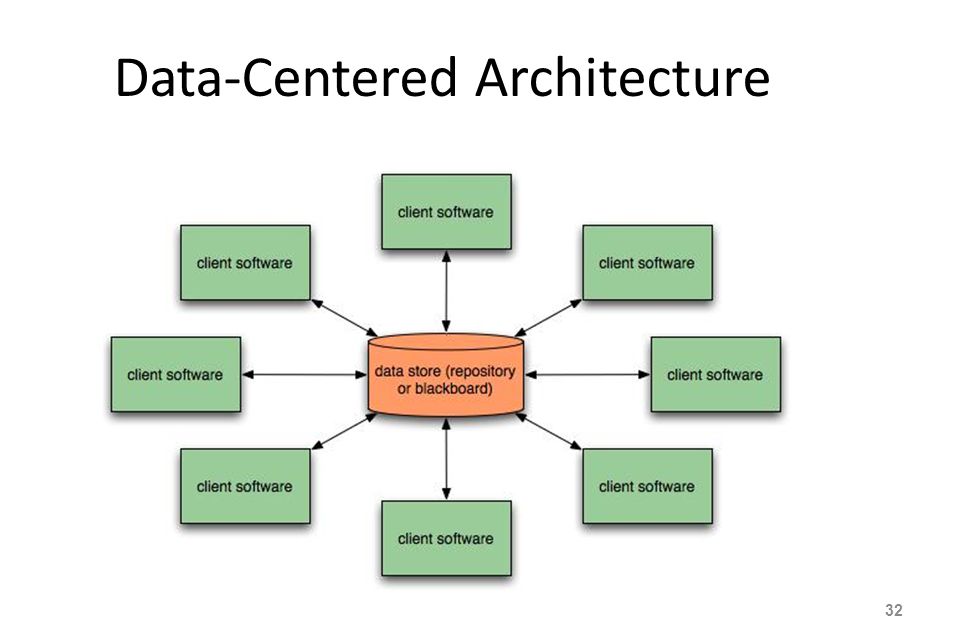
Data-centered architectures. A data store (e.g., a file or database) resides at

the center of this architecture and is accessed frequently by other components that

update, add, delete, or otherwise modify data within the store. Figure 9.1 illustrates a typical data-centered style. Client software accesses a central repository.

In some cases the data repository is passive. That is, client software accesses the

data independent of any changes to the data or the actions of other client software.A variation on this approach transforms the repository into a “blackboard”. that sends notifications to client software when data of interest to the client changes. Data-centered architectures promote integrability . That is, existing components can be changed and new client components added to the architecture without concern about other clients (because the client components operate independently). In addition, data can be passed among clients using the blackboard

mechanism (i.e., the blackboard component serves to coordinate the transfer

of information between clients). Client components independently execute processes.

Since the customer and admin details are going to be stored in the database managed by the adminstrator, therefore a data centered architecture will be the best choice for this project. Plus,the database can be modified as and when required by the administrator.

Detailed Design Specifications:

Component interface specifications:

The interface of the site is aimed to be user friendly and easy to use. Using advance design library the site is easy upon the user eyes and appeal with its simplistic design.

The frontend component is designed using

* HTML
* CSS
* Java Script

The backend is made using

* PHP
* SQL

The different component in our site are:

* Register/create account
* Login
* Choose dishes according to the offerings in the menu
* Calculate the bill
* Delivery options
* Approve/reject order(For admin)
* Change password

Each of the components in the site is linked to one another through the user main page or the menu page using hyperlinking in the front end and using session and cookies in the backend. The entire backend is linked with a SQL database for the storing of data of the system like orders, user details and menu items

Documentation for each routine:



Pseudocode for each routine:

The various routines that are covered under the site are

1. Register/create account:

Get the user following details from the user:

* Username
* Password
* Contact no:
* Email
* Address

When the user presses the submit button the system check whether the email address already exist or not and is it doesn’t it creates a new user

1. Login

A pop up is displayed and ask for the user to enter the username and password

On click of the login button it redirect the user to his/her account

1. Choose dishes according to the offerings in the menu

Create navigable tabs for each section of the menu

List the item with their names and price and with a checkbox to check when the item is to be added to the cart

Get the user to enter the quantity of the food required for a checked item

1. Calculate the bill

When the user place the order by pressing the submit button in the menu then subtotal of each item and grand total are to be displayed in a tabular manner

1. Approve/reject order(For admin)

The admin can reject or accept a particular order on the basis of the real time inventory he has or the current status of the restaurant.

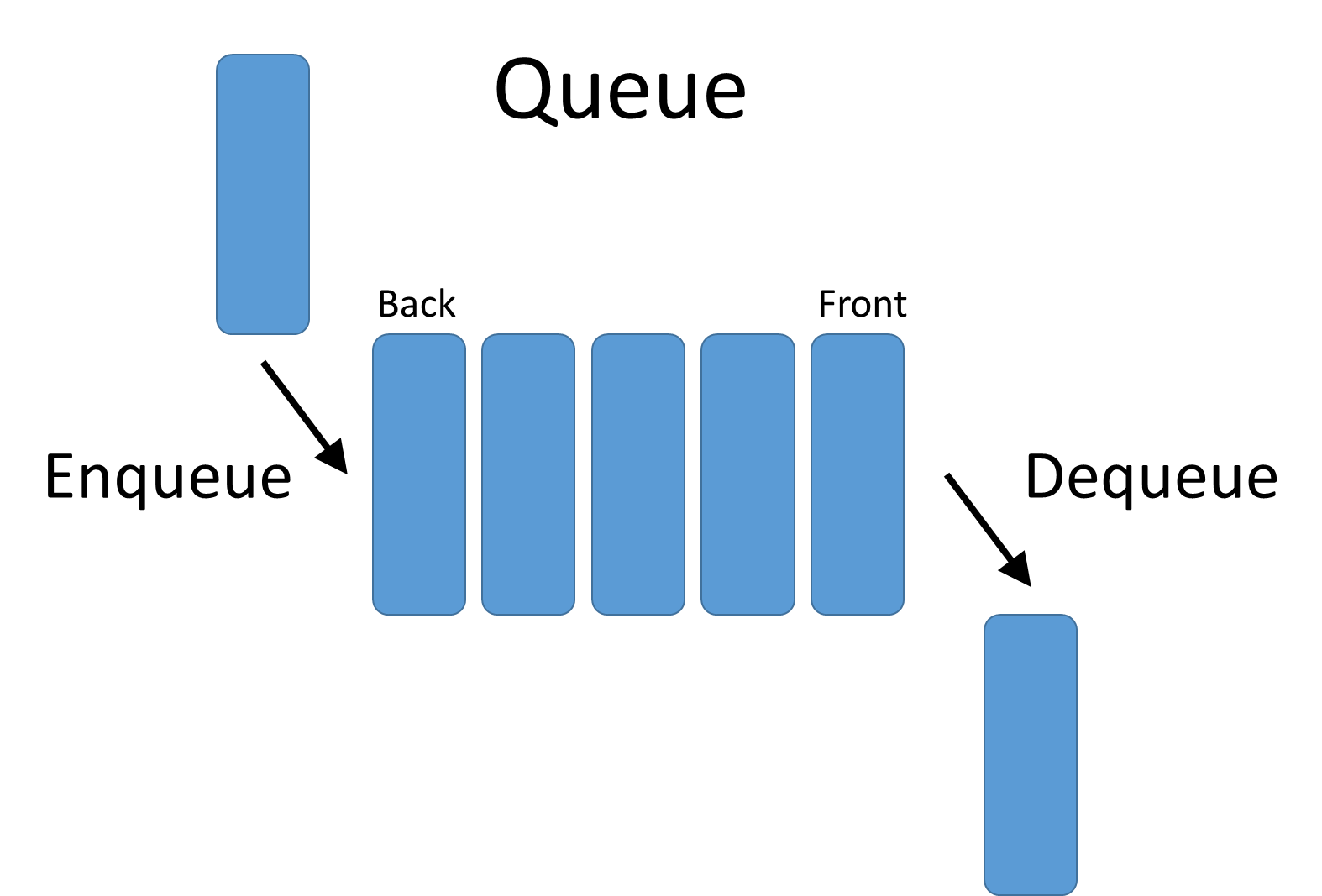
1. Change password

The form ask the user for the following

* His old password
* His new password
* Retyping his new password

Data Structure:

The data structure used for order processing is queue which operates in FIFO manner.



Packaging specifications:

Certain packaging principles have been followed in order to organize the entire system:

**The Release Reuse Equivalency Principle (REP).** *“The granule of reuse is the*

*granule of release”.* When classes or components are designed for reuse, there

is an implicit contract that is established between the developer of the reusable entity

and the people who will use it. The developer commits to establish a release control

system that supports and maintains older versions of the entity while the users slowly

upgrade to the most current version. Rather than addressing each class individually,

it is often advisable to group reusable classes into packages that can be managed and

controlled as newer versions evolve.

**The Common Closure Principle (CCP).** *“Classes that change together belong*

*together.”*. Classes should be packaged cohesively. That is, when classes are

packaged as part of a design, they should address the same functional or behavioural

area. When some characteristic of that area must change, it is likely that only those

classes within the package will require modification. This leads to more effective

change control and release management.

**The Common Reuse Principle (CRP).** *“Classes that aren’t reused together should*

*not be grouped together”*. When one or more classes within a package

changes, the release number of the package changes. All other classes or packages

that rely on the package that has been changed must now update to the most recent

release of the package and be tested to ensure that the new release operates without

incident. If classes are not grouped cohesively, it is possible that a class with no relationship to other classes within a package is changed. This will precipitate unnecessary integration and testing. For this reason, only classes that are reused together should be included within a package.