

BigShop using Reactjs

A PROJECT REPORT

Submitted By

Rajan Singh Baliyan

University Roll No- 1900290149077

**Submitted in partial fulfillment of the
Requirements for the Degree of**

MASTER OF COMPUTER APPLICATIONS

**Under the Supervision of
Oodles Technologies**



Submitted to

**DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206
(JUNE 2021)**

CERTIFICATE

Certified that **Rajan Singh Baliyan (University Roll No. 1900290149077)** has carried out the project work having “**BigShop**” for Master of Computer Applications from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Technical University, Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Date:

**Rajan Singh Baliyan
University Roll No. 1900290149077**

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Signature of Internal Examiner

Signature of External Examiner

**Dr. Ajay Kr. Shrivastava
Head, Department of Computer Applications
KIET Group of Institutions, Ghaziabad**

ABSTRACT

This project BigShop is an Reactjs application where the user is able to sell the crop product to the preferred seller through the online platform and get his fair share of the produce at the optimal price. This BigShop provides them the platform and the means through which they could make their unique account through their user id and password. And upload their contact information and upload the product information with the photograph , and browse through other seller products and compare their price according to it and then place the advertisement on the platform.

A seller could be a buyer also and buy anything of their interest on the BigShop, then find out the information of the seller and contact through it on the different medium and clarify their query and finalize their deal. Earlier the farmer has to go through the middlemen to sell their product and get very low returns after the hard work he has gone through. And the middlemen get the profit from both the seller and the buyer because both of them have to go through the wishes of the middlemen. But through BigShop now there is no place of the middlemen. And both of them could reap the reward. The primary purpose of the app is to provide the farmer the modern platform that is easy to use and also help him to grow with the help of technology alongside the country. BigShop has the easy to use interface and the simple feed of the information through which the buyer and the seller could be up to date according to the price of the product and sell their produce at their own price that seems fair to them. This BigShop is the app for the farmer to make them self-reliant and truly independent.

ACKNOWLEDGEMENTS

Success in life is never attained single handedly. My deepest gratitude goes to my department supervisor, **Dr. Ajay Kumar Shrivastava**, for his guidance, help and encouragement throughout my research work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Ajay Kumar Shrivastava, Professor and Head, Department of Computer Applications**, for his insightful comments and administrative help at various occasions.

Fortunately, I have many understanding friends, who have helped us a lot in many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me moral support and other kinds of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Rajan Singh Baliyan (MCA VIB)

Acknowledgement to Industry Expert

The success and outcome of the project required a lot of guidance and assistance from many people and we are extremely privileged to have got this for completion of our project. We are thankful to the Department of Computer Applications for assigning the Industry expert as a mentor of our project. All that we have the project under supervision and assistance of **Oodles Technologies** and **Ms. Rakshita Malik, Project Manager in Spectra Pvt. Ltd.**

We have taken support time to time from them and **Ms. Rakshita Malik** suggested us shortcoming and future enhancement of the project.

Details of Industry Experts:

Name	Designation	Organization	Mode of Communication (E-mail/ Phone Call)
Ms. Rakshita Malik	Project Manager	Spectra Private Limited	Rakshita.maik.spectra.co

Date:

Rajan Singh Baliyan(MCA VIB)

Table of Contents

Certificate	i
Abstract	ii
Acknowledge	iii
Acknowledge to the Industry Expert	iv
Table of content	v
List of Figure	vi
1 Introduction	
1.1 Project description	1
1.2 Project Scope	3
1.3 Hardware/Software Requirement Specification	4
1.3.1 Introduction	5
1.3.2 Benefit of Reactjs apps	6
1.3.3 How Reactjs apps work	7
2 Feasibility Study	
2.1 Economical feasibility	20
2.2 Technical Feasibility	23
2.3 Behavioral Feasibility	25
2.4 Operational Feasibility	29
2.5 Nalanda e consortium [Reactjs project Development]	30
3 System Design	
3.1 ER Diagram	39
3.2 Database Design	41
3.3 DFD and Flowgraph	46
4 Form Design	
4.1 Input / Output Form (Screenshot)	50
5 Coding	55
6 Testing	94
7 Conclusion & Future Scope	100
8 Bibliography	102

CHAPTER 1

INTRODUCTION

1.1 PROJECT DESCRIPTION

India is mainly an agricultural country. Agriculture is the most important occupation for most of the Indian rural families. The project focuses on farmers of India to sell their products in a smart, and effective way. This also improves productivity in agriculture and provides assistance. With the help of this project one can sell their products worldwide to the Wholesalers. Apart from this it is also suitable for the buyers to buy the crop according to their need with more reliability. This is one of the most convenient ways to take farming to a new level even though each product uploaded by the farmer has a proper description about that product. Farmers will now be able to sell their produce online in any of the markets where they can get the best price and can do their work smartly. Though we can say that this will make India more digitized.

- **Saving time spent on tedious tasks**

Even if you've only got a dozen customers who want to buy the crops at a given price, there are various more efforts done by a farmer to sell his products as there are very few customers who do not bargain, and this process always takes more time as compared to online selling of the crops. Also the online customer is ready to buy the crops at given prices. So the farmers do not need to make so much effort to sell their products at their prices.

- **Increasing Productivity**

This is something that goes hand in hand with the time-consuming tasks. As such, the use of BigShop will help farmers to be more productive. As the time taken by a farmer to sell their products manually

will get reduced and then they can use that part of time over their crops and land to make them more fertilized.

- **Proposed System**

In our mobile project we are giving detailed information about the farmer and about their crop which is provided by the farmer as they can upload the picture of their produce through which the customer get proper satisfaction what he is purchasing and we are also providing the messaging option through which both the seller and the buyer can communicate to each other. As in other projects there is no proper way to make communication possible between the farmer and his customer.

- **Existing System**

Currently farmers are using various mobile projects which are available for farmers in India. But they have a lack of management and are not that much user friendly. To make our App more friendly to use we have provided the Reactjs based UI. And also old apps have not that much reliability as to increase the reliability of both the customer and seller i.e, the farmer can communicate to each other via phone or messaging process provided by our project. And to make payment they can do it in cash in hand mode or via some other project depending upon their mood or conveyance.

1.1 PROJECT SCOPE

To develop a BigShop to help the farmer to sell their product online and help them to find the buyer and his own price, Scope of this project is wide. This project automates the buying and selling process and makes it easy and simple. This project removes the middlemen and helps farmers to contact the buyer through any broker that helps them to save a huge amount of money. BigShop provides Farmer access to the list of buyers and is able to select the buyer who is willing to pay the appropriate price for the product. This project keeps the record of each and every buyer and their information and their product. Farmers can assign the quantity of the product to the buyers, its variety, location, transportation cost. This project provides farmers and buyers to track the product detail before and after the completion of the purchase. Every Farmer wants to sustain in this competitive world. To remain top there should be a right platform to grow and to increase his standard. So there is a need for a project which allows the farmer to sell the product at his own price. Hence there is huge scope for BigShop.

1.1.1 Hardware/Software used in Project

As this is a Reactjs and cloud based project so there is no need for sophisticated hardware only there is a Reactjs phone and Internet connection. Reactjs provides the development infrastructure to create projects. An Reactjs project and a cloud app, is a software program where cloud-based and local components work together. This model relies on remote servers for processing logic that is accessed through a web browser with a continual internet connection.

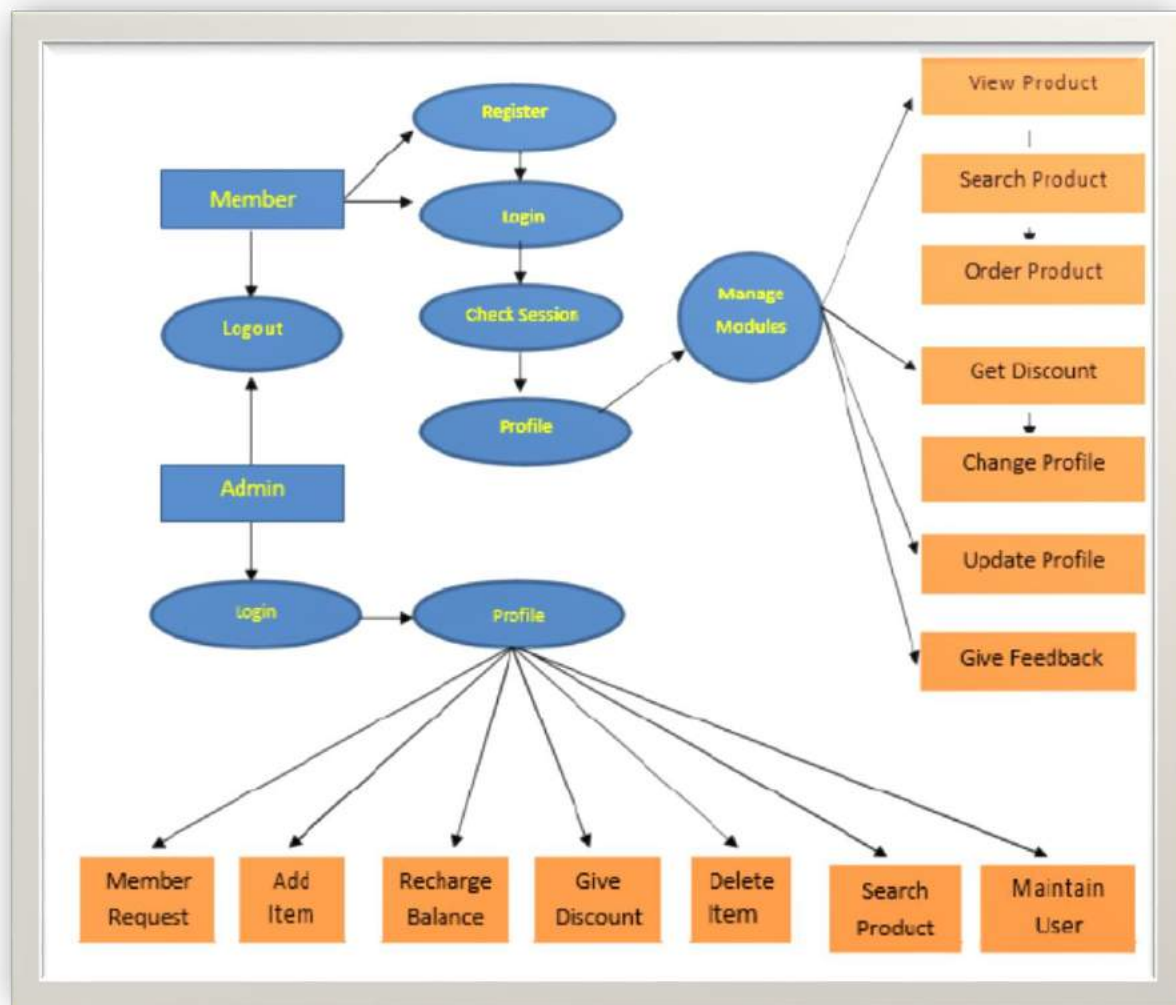
Cloud project servers typically are located in a remote data center operated by a third-party cloud services infrastructure provider. Cloud-based project tasks may encompass email, file storage and sharing, order entry, inventory management, word processing, customer relationship management (CRM), data collection, or financial accounting features.

- **Fast response to business needs.** Cloud projects can be updated, tested and deployed quickly, providing enterprises with fast time to market and agility. This speed can lead to culture shifts in business operations.
- **Simplified operation.** Infrastructure management can be outsourced to third-party cloud providers.
- **Instant scalability.** As demand rises or falls, available capacity can be adjusted.
- **API use.** Third-party data sources and storage services can be accessed with an project programming interface (API). Cloud projects can be kept smaller by using APIs to hand data to projects or API-based back-end services for processing or analytics computations, with the results handed back to the cloud project. Vetted APIs impose passive consistency that can speed development and yield predictable results.
- **Gradual adoption.** Refactoring legacy, on-premises projects to a cloud architecture in steps, allows components to be implemented on a gradual basis.
- **Reduced costs.** The size and scale of data centers run by major cloud infrastructure and service providers, along with competition among providers, has led to lower prices. Cloud-based projects can be less

expensive to operate and maintain than equivalents on-premises installation.

- **Improved data sharing and security.** Data stored on cloud services is instantly available to authorized users. Due to their massive scale, cloud providers can hire world-class security experts and implement infrastructure security measures that typically only large enterprises can obtain. Centralized data managed by IT operations personnel is more easily backed up on a regular schedule and restored should disaster recovery become necessary. cloud project, or cloud app, is a software program where cloud-based and local components work together. This model relies on remote servers for processing logic that is accessed through a web browser with a continual internet connection. Cloud project servers typically are located in a remote data center operated by a third-party cloud services infrastructure provider. Cloud-based project tasks may encompass email, file storage and sharing, order entry, inventory management, word processing, customer relationship management (CRM), data collection, or financial accounting features.
- **Fast response to business needs:** Cloud projects can be updated, tested and deployed quickly, providing enterprises with fast time to market and agility. This speed can lead to culture shifts in business operations.
- **Simplified operation:** Infrastructure management can be outsourced to third-party cloud providers.
- **Instant scalability:** As demand rises or falls, available capacity can be adjusted.

- **API use:** Third-party data sources and storage services can be accessed with an project programming interface (API). Cloud projects can be kept smaller by using APIs to hand data to projects or API-based back-end services for processing or analytics computations, with the results handed back to the cloud project. Vetted APIs impose passive consistency that can speed development and yield predictable results.
- **Gradual adoption:** Refactoring legacy, on-premises projects to a cloud architecture in steps, allows components to be implemented on a gradual basis.
- **Reduced costs:** The size and scale of data centers run by major cloud infrastructure and service providers, along with competition among providers, has led to lower prices. Cloud-based projects can be less expensive to operate and maintain than equivalents on-premises installation.
- **Improved data sharing and security:** Data stored on cloud services is instantly available to authorized users. Due to their massive scale, cloud providers can hire world-class security experts and implement infrastructure security measures that typically only large enterprises can obtain. Centralized data managed by IT operations personnel is more easily backed up on a regular schedule and restored should disaster recovery become necessary.



3.1.1 How Reactjs apps work

Reactjs apps can be written using Kotlin, Java, and C++ languages. The Reactjs SDK tools compile your code along with any data and resource files into an APK, an *Reactjs package*, which is an archive file with an .apk suffix. One APK file contains all the contents of an Reactjs app and is the file that Reactjs-powered devices use to install the app.

Each Reactjs app lives in its own security sandbox, protected by the following Reactjs security features:

- The Reactjs operating system is a multi-user Linux system in which each app is a different user.

- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. The Reactjs system starts the process when any of the app's components need to be executed, and then shuts down the process when it's no longer needed or when the system must recover memory for other apps.

The Reactjs system implements the *principle of least privilege*. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission. However, there are ways for an app to share data with other apps and for an app to access system services:

- It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM. The apps must also be signed with the same certificate.
- An app can request permission to access device data such as the device's location, camera, and Bluetooth connection. The user has to explicitly grant these permissions. For more information, see [Working with System Permissions](#).

The rest of this document introduces the following concepts:

- The core framework components that define your app.

- The manifest file in which you declare the components and the required device features for your app.
- Resources that are separate from the app code and that allow your app to gracefully optimize its behavior for a variety of device configurations.

3.1 App components

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four different types of app components:

- Activities
- Services
- Broadcast receivers
- Content providers

Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. The following sections describe the four types of app components.

- **Activities**

An *activity* is the entry point for interacting with the user. It represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities if the email app allows it. For example, a camera app can start the activity in the email app

that composes new mail to allow the user to share a picture. An activity facilitates the following key interactions between system and app:

- Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
- Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
- Helping the app handle having its process killed so the user can return to activities with their previous state restored.
- Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

You implement an activity as a subclass of the Activity class. For more information about the Activity class, see the Activities developer guide.

- **Services**

A *service* is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. There are actually two very distinct semantics services tell the system about how to manage an app: Started services tell the system to keep them running until their work is completed. This could

be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music also represent two different types of started services that modify how the system handles them:

- Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
- A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B (and its service) running for A. Further, if process A is something the user cares about, then it also knows to treat process B as something the user also cares about. Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screensavers, input methods, accessibility services, and many other core system features are all built as services that projects implement and the system binds to when they should be running.

A service is implemented as a subclass of Service. For more information about the Service class, see the Services developer guide.

CHAPTER 2

Feasibility Study

As the name implies, a feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and technically feasible as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

A well-designed study should offer a historical background of the business or project, such as a description of the product or service, accounting statements, details of operations and management, marketing research and policies, financial data, legal requirements, and tax obligations. Generally, such studies precede technical development and project implementation.

- **Types of Feasibility Study**

A feasibility analysis evaluates the project's potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lending institutions. There are four types of feasibility study—separate areas that a feasibility study examines, described below.

2.1 Economical Feasibility

System is economical feasible and can be easily implement with minimum hardware and software resources as this is a cloud based project platform is provided by cloud provider only there is a need of Internet connection and a Browser project. It is very important for designer to first analyze the system economically and determines that project is economical feasible or not. Costs and benefits of the proposed computer system must always be considered together, because they are interrelated and often interdependent. Although the systems analyst is trying to propose a system that fulfills various information requirements, decisions to continue with the proposed system will be based on a cost-benefit analysis, not on information requirements. In many ways, benefits are measured by costs, as becomes apparent in the next section. Systems analysts are required to predict certain key variables before the proposal is submitted to the client. To some degree, a systems analyst will rely on a what-if analysis, such as, "What if labor costs rise only 5 percent per year for the next three years, rather than 10 percent?" The systems analyst should realize, however, that he or she cannot rely on what-if analysis for everything if the proposal is to be credible, meaningful, and valuable.

The systems analyst has many forecasting models available. The main condition for choosing a model is the availability of historical data. If they are unavailable, the analyst must turn to one of the judgment methods: estimates from the sales force, surveys to estimate customer demand, Delphi studies (a consensus forecast developed independently by a group of experts through a series of iterations), creating scenarios, or drawing historical analogies.

If historical data are available, the next differentiation between classes of techniques involves whether the forecast is conditional or unconditional. Conditional implies that there is an association among variables in the model or that such a causal relationship exists. Common methods in this group include

correlation, regression, leading indicators, econometrics, and input/output models.

Unconditional forecasting means the analyst isn't required to find or identify any causal relationships. Consequently, systems analysts find that these methods are low-cost, easy-to-implement alternatives. Included in this group are graphical judgment, moving averages, and analysis of time-series data. Because these methods are simple, reliable, and cost effective, the remainder of the section focuses on them.

- **Estimation of Trends**

Trends can be estimated in a number of different ways. One way to estimate trends is to use a moving average. This method is useful because some seasonal, cyclical, or random patterns may be smoothed, leaving the trend pattern. The principle behind moving averages is to calculate the arithmetic mean of data from a fixed number of periods; a three-month moving average is simply the average of the last three months. For example, the average sales for January, February, and March is used to predict the sales for April. Then the average sales for February, March, and April are used to predict the sales for May, and so on.

When the results are graphed, it is easily noticeable that the widely fluctuating data are smoothed. The moving average method is useful for its smoothing ability, but at the same time it has many disadvantages. Moving averages are more strongly affected by extreme values than by using graphical judgment or estimating using other methods such as least squares. The analyst should learn forecasting well, as it often provides information valuable in justifying the entire project.

- **Tangible Costs**

The concepts of tangible and intangible costs present a conceptual parallel to the tangible and intangible benefits discussed already. Tangible costs are those that can be accurately projected by the systems analyst and the business's accounting personnel.

Included in tangible costs are the cost of equipment such as computers and terminals, the cost of resources, the cost of systems analysts' time, the cost of programmers' time, and other employees' salaries. These costs are usually well established or can be discovered quite easily, and are the costs that will require a cash outlay of the business.

- **Intangible Costs**

Intangible costs are difficult to estimate and may not be known. They include losing a competitive edge, losing the reputation for being first with an innovation or the leader in a field, declining company image due to increased customer dissatisfaction, and ineffective decision making due to untimely or inaccessible information. As you can imagine, it is next to impossible to project a dollar amount for intangible costs accurately. To aid decision makers who want to weigh the proposed system and all its implications, you must include intangible costs even though they are not quantifiable.

2.2 Technical Feasibility

It is the study of the function performance and constraints that may affect the ability to achieve an acceptable system. The project development requires designer to have technical knowledge of Reactjs.com for both project development and database system. Technical feasibility is one of the most important criteria for selecting material for digitization. The physical characteristics of source material and the project goals for capturing, presenting and storing the digital surrogates dictate the technical requirements. Libraries must evaluate those requirements for each project and determine whether they can be met with the resources available. If the existing staff, hardware and software resources cannot meet the requirements, then the project will need funding to upgrade equipment or hire an outside conversion agency. If these resources are not available, or if the technology does not exist to meet the requirements, then it is not technically feasible to digitize that material.

Considerations for technical feasibility include:

- **Image capture**

Image capture requires equipment, such as a scanner or a digital camera. Different types of material require different equipment, and different equipment produces images of differing quality. When selecting materials for digitising, technical questions that need to be addressed include: does the original source material require high resolution to capture? Are there any oversized items in the collection? Are there any bound volumes in the collection? What

critical features of the source material must be captured in the digital product? In what condition are the source materials? Will they be damaged by the digitization process?

- **Presentation**

Presentation refers to how the digitized materials will be displayed online. Consider the following questions to determine the technical feasibility of presenting the digitized material:

Will the materials display well digitally?

How will users use the digital versions?

How will users navigate within and among digital collections?

Do the institutionally supported platforms and networked environment have the capability for accessing the images and delivering them with reasonable speed to the target audience?

Do the images need to be restricted to a specified community?

Do the images need special display features such as zooming, panning and page turning?

- **Description**

Some archival and special collections have been catalogued for public use and contain detailed finding aids with descriptions about each item and the collection as a whole. Other collections may not have been reviewed and documented in detail and do not have much information on individual items. Those collections will require more time, human resources and significant additional expense to research the materials, check the accuracy of the information obtained, and

write appropriate descriptions to aid in discovery and use of the digital items. Typewritten documents, like the Drew Pearson columns described above, can have reasonably accurate OCR applied to them to replace, for some uses, the detailed descriptions required for discovery of hand-written or picture materials. The selection criteria should clearly state whether the items and collections that do not contain descriptions should be considered for digitisation.

- **Human resources**

When selecting materials for digitisation, the library should consider whether it has the staff and skill sets to support the digitisation, metadata entry, user interface design, programming and search engine configuration that is required for the project to implement the desired functionality. For large collaborative projects, dedicated staff are usually required from each partner. Digital collections also require long-term maintenance, which needs to be considered and planned for. If a project does not have the necessary staff and skills in-house, but funding is available, outsourcing may be a good choice.

2.3 Behavioral Feasibility

In the project domain our system works as an project. There are simple form to fill and service requires no ambiguous entries, all the behavioural entries are simple and GUI based. The system design is very user friendly, interactive. The

project should be used by Administrator. People are inherently resistant to change, and computers have been known to facilitate change. An estimate should be made of how strong a reaction the user staff is likely to have toward the development of a computerized system. [t is common knowledge that computer installations have something to do with turnover, transfers, retraining, and changes in employee product status. Therefore, it is understandable that the introduction of a Product system requires special effort to educate, sell, and train the staff on new ways of conducting business.

In our safe deposit example, three employees are more than 50 years old and have been with the bank over 14 years, four years of which have been in safe deposit. The remaining two employees are in their early thirties. They joined safe deposit about two years before the study. Based on data gathered from extensive products, the younger employees want the programmable aspects of safe deposit (essentially billing) put on a computer. Two of the three older employees have voiced resistance to the idea. Their view is that billing is no problem. The main emphasis is customer service-personal contacts with customers. The decision in this case was to go ahead and pursue the project.

2.4 Operational Feasibility

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.^[10]

The operational feasibility assessment focuses on the degree to which the proposed development project fits in with the existing business environment and

objectives with regard to development schedule, delivery date, corporate culture and existing business processes.

To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely project engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases

An example of an operational feasibility study, or the fourth type, analyzes the inside operations on how a deemed process will work, be implemented, and how to deal with change resistance and acceptance.

Operational feasibility studies are generally utilized to answer the following questions:

- Process – How do the end-users feel about a new process that may be implemented?
- In-House Strategies – How will the work environment be affected? How much will it change?
- Adapt & Review – Once change resistance is overcome, explain how the new process will be implemented along with a review process to monitor the process change.

If an operational feasibility study must answer the six items above, how is it used in the real world? A good example might be if a company has determined that it needs to totally redesign the workspace environment.

After analyzing the technical, economic, and scheduling feasibility studies, next would come the operational analysis. In order to determine if the redesign of the workspace environment would work, an example of an operational feasibility study would follow this path based on six elements:

- Process – Input and analysis from everyone the new redesign will affect along with a data matrix on ideas and suggestions from the original plans.
- Evaluation – Determinations from the process suggestions; will the redesign benefit everyone? Who is left behind? Who feels threatened?
- Implementation – Identify resources both inside and out that will work on the redesign. How will the redesign construction interfere with current work?
- Resistance – What areas and individuals will be most resistant? Develop a change resistance plan.
- Strategies – How will the organization deal with the changed workspace environment? Do new processes or structures need to be reviewed or implemented in order for the redesign to be effective?
- Adapt & Review – How much time does the organization need to adapt to the new redesign? How will it be reviewed and monitored? What will happen if through a monitoring process, additional changes must be made?

The most important part of operational feasibility study is input—from everyone, especially when it affects how or what an organization does as far as

processes. If the process were to build a new sports arena for a client, then a study determining how the arena will operate in a way that is conducive to its inhabitants, parking, human flow, accessibility and other elements is a good example of an operational feasibility study.

Create a sample operational feasibility study if you plan to change something inside the company that will affect how the organization runs or when a client asks you to explore a new product or process that will affect elements within their own organization.

2.5 Nalanda e consortium [Reactjs project Development]

Reference-

- Author Name-Michael Burton
- Publisher Name-Wiley (John Wiley & Sons)

<https://ebooks.wileyindia.com/product/Reactjs-app-development-for-dummies>

- **Introducing the Reactjs App Development for dummie**

The updated edition of the bestselling guide to Reactjs app development. If you have ambitions to build an Reactjs app, this hands-on guide gives you everything you need to dig into the development process and turn your great idea into a reality! In this new edition of Reactjs App Development For Dummies, you'll find easy-to-follow access to the latest programming techniques that take advantage of the new features of the Reactjs operating system. Plus, two programs are provided: a simple program to get you started and an intermediate program that uses more advanced aspects of the Reactjs platform.

Reactjs mobile devices currently account for nearly 80% of mobile phone market share worldwide, making it the best platform to reach the widest possible audience. With the help of this friendly guide, developers of all stripes will quickly find out how to install the tools they need, design a good user interface, grasp the design differences between phone and tablet projects, handle user input, avoid common pitfalls, and turn a "meh" app into one that garners applause.

Create seriously cool apps for the latest Reactjs smartphones and tablets

Adapt your existing apps for use on an Reactjs device

Start working with programs and tools to create Reactjs apps

Publish your apps to the Google Play Store

Whether you're a new or veteran programmer, Reactjs App Development For Dummies will have you up and running with the ins and outs of the Reactjs platform in no time.

- **The Dawn of the Customer Journey**

We're at a tipping point in digital marketing, where data, tools, and predictive analytics are coming together to drive a concept known as the customer journey. Before we can dive into the depths of modern-day customer journeys, however, we need to take you on a journey of our own. We're going to go back to where it began - email marketing – to understand email marketing as a channel and how we got from there to where we are

Today

- **Early email marketing**

The technology to send email messages emerged in the early 1970s, but only government and educational institutions really had access to it. In the mid-1980s, commercial networks began opening up the potential of this messaging channel to private citizens mostly early adopters who loved technology for its own sake. Email as a common messaging medium, with practical projects for average citizens, didn't really take off until the 1990s.

At that time, major commercial networks, such as CompuServe and AOL, started connecting to the Internet and allowing messages to pass among competing systems. These messages were mostly text based and basic.

- **Navigating ParseUser Cloud**

ParseUser Cloud starts with the dashboard. The dashboard contains its own overarching tools — such as a calendar of your planned marketing activities and a real-time snapshot of your campaign performance — and is also how to access your apps.

Apps are the meat of Cloud's functionality. Cloud contains a variety of powerful apps you can use for your online marketing campaigns.

You can license all or just a few of the Cloud apps, depending on your online marketing tool needs. Regardless of whether you have licensed a particular app, though, you can see all Cloud apps in the dashboard. (If you try to open an app that you haven't licensed, a message appears to explain that the app is not available.) Reactjs wants you to know that the tool you need could be just a click away!

In this chapter, you take a whirlwind tour of all the dashboard tools and the apps. This journey sets the stage for later chapters, where you dig into the specifics of how to use the tools and apps.

- **Exploring Cloud**

The ParseUser Cloud dashboard, shown in Figure 2-1, is the first thing you see when you log in to your Cloud account. From the dashboard you can access the following:

The dashboard tools, which are available to every Cloud account. Links to the dashboard tools appear in the toolbar.

The apps, which are available in your Cloud account if you licensed them. Links to the apps appear in the app switcher. The app switcher is visible when you first log in to your Cloud account. It disappears when you choose a tool or an app. You can get back to it at any time by hovering your mouse pointer over the ParseUser Cloud icon on the left side of the toolbar.

- **Administering Cloud**

If you have the responsibility of acting as an administrator for your ParseUser Cloud account the decisions you make ensure that your account is secure your users can do their work, and your marketing campaigns get delivered without issue. No pressure!

Thankfully, many of the administrative tasks don't require a lot of ongoing time or effort to maintain. This chapter is not intended to be a comprehensive view of every administrative function, just the necessities to set up your account for success.

This chapter assumes that your Cloud user account has administrator permissions. If your user doesn't have administrator permissions, you won't be able to see many of the features described here.

- **Managing Cloud Users**

Every person who uses your Cloud account should have his or her own user account. That means each person will have a username and password that he or she doesn't share with anyone.

Having a separate user account for each person makes your Cloud account more secure. You can track when and from where each user accesses Cloud. And if a person leaves the company, you can disable his user account without disrupting the other users' workflow.

Cloud uses permissions and roles to determine which features in Cloud a particular user can access. So having separate user accounts for each person has the added benefit of giving you granular control over who accesses what in Cloud.

You maintain Cloud user accounts on the Users page. You can get to this page from the Cloud toolbar or the Email app.

- **Identifying and Preparing Your Data**

without question, the biggest hurdle we've encountered in our marketing careers is data. Early on, the challenge was getting access to meaningful information. Everything seemed to stand in our way from systems that weren't designed for sharing to corporate fiefdoms that were threatened by the incorrect assumption that we were trying to compete with their sales team.

These days, however, the situation is reversed. Rather than struggling to find any useful data, we are now overwhelmed with data from all over the organization. It seems like everyone wants a finger in the pie because he thinks his own data is the most important to incorporate into the online marketing efforts. In this type of situation, it's easy to lose sight of what is essential.

- **Defining Your Data Set**

Your data set is the list of the pieces of information you maintain about each subscriber. Sadly, we can't define your data set for you. Although some kinds of data are useful for almost everyone, the combination of your marketing plan, target demographics, and business objectives make your data needs unique. What we can do, though, is help guide your thinking about what data components you need as we walk you through the process of designing your data set for use in ParseUser Cloud. we discuss how to implement the plan you define here.

- **What data do you have?**

A good first task is to take inventory of the customer data you already have. Even the smallest business has data, but it might be residing in a surprisingly wide variety of business systems. Customer contact information is essential, of course, but you might be able to make use of other kinds of data, such as purchase history.

Cast a wide net when listing your possible data sources. Don't limit yourself at this point you compare the scope of the data later. Don't forget to consider the following locations:

Customer relationship management (CRM) system: For example, an automotive service shop probably has contact information, information about the vehicle, and a history of services

Point-of-sale or billing system: If you collect information from your customers as part of the sale, your point-of-sale or billing system can be a rich resource of customer data.

Existing communications methods: For example, if you've ever set up a form for customers to sign up for a newsletter or request more information, that form has been collecting valuable data you can use.

Loyalty program: If you offer rewards to your customers for their continued

Business, don't forget to mine the system you use to administer it.

Customer preference center: Your existing content publishing processes might have already inspired you to set up a website where customers can indicate preferences, such as what topics they're interested in and how often they want to receive messages from you.

- **Establishing Your Data Model**

You searched far and wide for data you could consider using in your online

marketing campaigns before mercilessly culling the data that didn't help you meet your marketing objectives. Now that you have a healthy, well-thought-out pool of data, you need to get it into a place where ParseUser Cloud can use it.

For a long time, Cloud had only one simple and effective approach to storing data. But new functionality demanded a more sophisticated — and more complex model to store different kinds of data that you use in multimedia online marketing campaigns.

In this chapter, we discuss your data storage options and help you decide which one to use. Then we talk about how to set up your data model and how to import the data you already have into it.

- **Understanding Cloud Data Models**

The two data models in Cloud are as follows:

Subscriber/list model: All data is stored in fields in a subscriber record and

subscribers are grouped into lists for sending.

Relational data model: Data is stored in relational database tables called data

Extensions Subscriber-and-list model

The traditional way to store subscriber data in Cloud has been to use the subscriber-and-list model. This approach is simple, straightforward, and has some nice built-in conveniences.

In this model, Cloud considers each subscriber to be a complete entity.

The email address identifies the subscriber entity, and profile and preference attributes contain the following kinds of data about the subscriber:

Profile attributes: Contain demographic data about the subscriber.

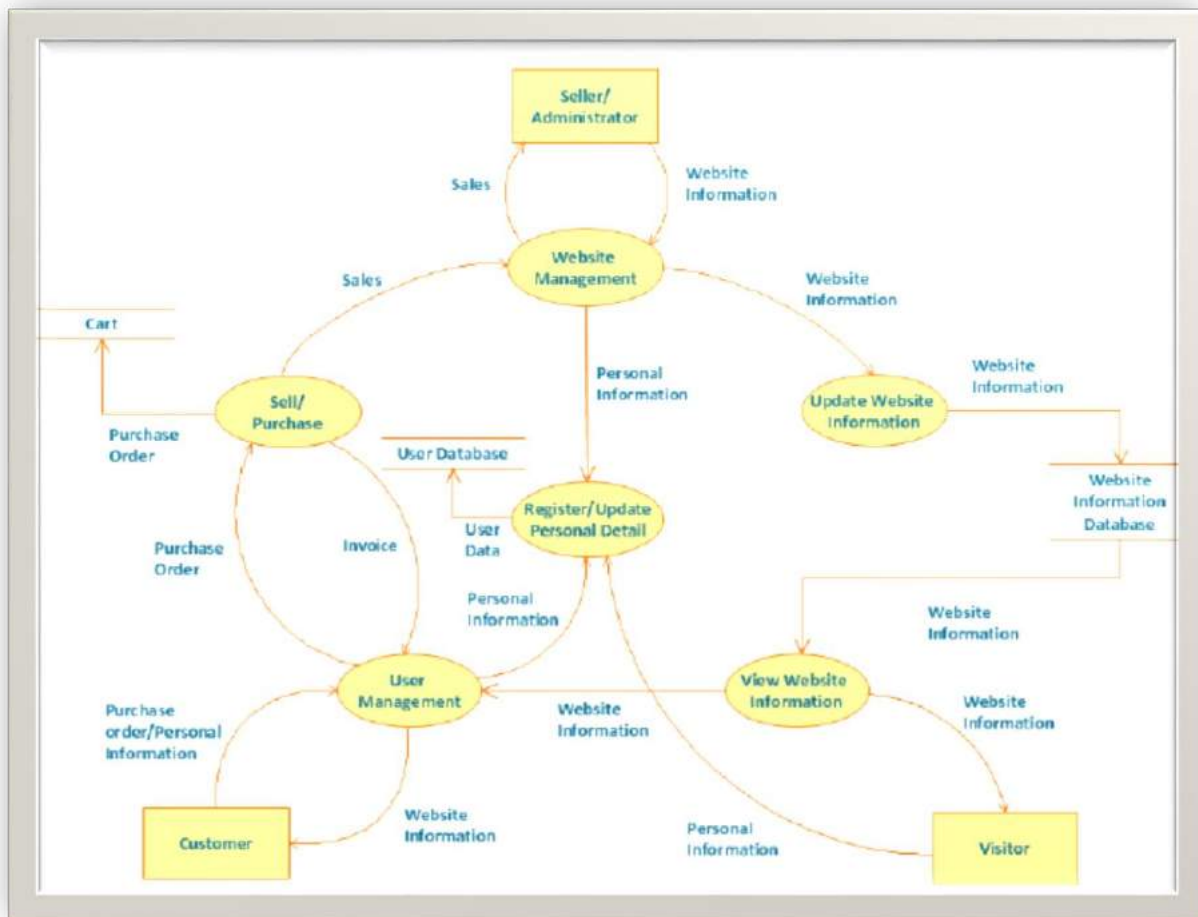
Figure 6-1

shows the profile attributes that exist in your account by default.

Preference attributes: Contain yes no choices that subscribers communicate to you about how they want to hear from you. The only preference attribute delivered by default is called HTML Email. When subscribers select the check box next to this preference, they receive your emails in beautiful, full-color HTML. If they deselect this check box, they receive a text-only version of your email.

CHAPTER 3

System Design



3.2 ER-Diagram

E-R (Entity-Relationship) Diagram is used to represent the relationship between entities in a table. ER diagrams represent the logical structure of databases. ER Diagrams represent relationships between two database tables.

E-R diagram means Entity Relationship diagram. Entity is an object of the system, generally we refer to the entity as a database table, the e-r diagram represents the relationship between each table of database. E-R diagrams represent entities with attributes, attributes are the properties of the entity. If we

assume an entity is a database table then all the columns of the table are treated as attributes.

Here are the geometric shapes and their meaning in an E-R Diagram. We will discuss these terms in detail in the next section(Components of a ER Diagram) of this guide so don't worry too much about these terms now, just go through them once.

- **Rectangle:** Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set

The User applies to products and the company posts products and the user has applied to certain products. The tables are represented in the form of rectangular boxes as shown in the diagram. The primary keys are underlined. The relationships are shown through diamond boxes. The cardinalities are mentioned through the numbers.

3.3 Database Design

- **A Parseuser DATABASE**

Database technology is the persistence layer at the heart of all data centric projects, the tier that's in charge of organizing, protecting, and managing shared database access reliably, securely, efficiently. The persistence layer underlying Parseuser (and Database.com) is proven database technology that powers all of Reactjs.com's products today, serving more than 100,000 organizations, 135,000 projects, 3 million users, 1 billion transactions per day with an average request response time of less than 300ms, all with an average uptime of 99.9+ percent.

- **Features of Parse Server Database**

Easy to use with Parseuser, there's nothing to manage — Reactjs.com takes care of everything for you. There's no software to install, update, and patch. No waiting on someone else when you want to provision databases. No worries about database backup and disaster recovery. No complex documentation set with thousands of pages and parameters to tune for performance or elasticity. There's even automatic indexing. Whether you have 1 database or 1,000 databases, all you need to focus on is building great apps.

TRUSTWORTHY Parseuser is built with the security and privacy of customer information in mind. Reactjs.com's infrastructure and corporate workplace meet all of the highest industry standards, including SAS 70 Type II, SysTrust, and ISO 27001 certifications.

MODERN Parseuser is more than just another database system — it's jam packed with next generation features that make building and maintaining highly functional, secure, social, and mobile apps a snap.

- Parseuser users, profiles, roles, groups, and row level sharing rules help you build secure apps without the need to code, test, and maintain your own complicated security logic.

- With Parseuser, it's easy to implement common project logic without writing complicated and error prone code. Such features include declarative, 36 point and click configuration for workflows, encrypted/masked fields, validation rules, formula fields, rollup summary fields, and cross object validation rules.
- Parseuser is "social" because it includes the Reactjs Chatter API, a builtin data model apps can leverage to become instantly social and collaborative.
- Parseuser's REST APIs, OAuth implementation for user authentication/authorization, data feeds, custom Web services, embedded security model, and other features make it a perfect fit for easily building secure, scalable mobile apps, either native or HTML5.

OPEN Parseuser's full complement of open APIs lets you build and integrate projects using the approach of your choice. REST and SOAP based APIs are standards based APIs that make Parseuser open to whatever programming language you want to use. Using various APIs, your projects can do many things such as create read update delete (CRUD) business data, load a large number of records asynchronously, and take advantage of the Chatter API to provide collaboration and social networking capabilities to any project.

POWERFUL Most modern apps use server side logic to centralize complex business logic and enforce complex data integrity rules. Apex, with syntax much like Java, is Parseuser's procedural language that you can use to create server side logic for an project. For example, Apex lets you create stored procedures that modify the database within the context of ACID transactions, and expose them as a custom Web services API (RESTful or SOAP) for your apps. You can also use Apex to build database triggers, routines that automatically fire (execute) when apps modify records in your database.

- Database design in BigShop Reactjs provides Parseuser database to store, manage and retrieve data. In Farmer project we used Parseuser database to store Product details, positions, product level, salary. The tables we have designed are as follows.

● Product Table

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> admins	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	32 KiB	-
<input type="checkbox"/> brands	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> carts	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_unicode_ci	64 KiB	-
<input type="checkbox"/> categories	★ Browse Structure Search Insert Empty Drop	4	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> districts	★ Browse Structure Search Insert Empty Drop	5	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> divisions	★ Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> migrations	★ Browse Structure Search Insert Empty Drop	14	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> orders	★ Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_unicode_ci	48 KiB	-
<input type="checkbox"/> password_resets	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	32 KiB	-
<input type="checkbox"/> payments	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_unicode_ci	32 KiB	-
<input type="checkbox"/> products	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> product_images	★ Browse Structure Search Insert Empty Drop	7	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> settings	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> sliders	★ Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_unicode_ci	16 KiB	-
<input type="checkbox"/> users	★ Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	64 KiB	-
15 tables	Sum	67	InnoDB	latin1_swedish_ci	416 KiB	0 B

● Admin Table

Table structure

Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(10)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
<input type="checkbox"/> 2	name	varchar(191)	utf8mb4_unicode_ci		No	None			Change Drop More
<input type="checkbox"/> 3	email	varchar(191)	utf8mb4_unicode_ci		No	None			Change Drop More
<input type="checkbox"/> 4	password	varchar(191)	utf8mb4_unicode_ci		No	None			Change Drop More
<input type="checkbox"/> 5	phone_no	varchar(191)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 6	avatar	varchar(191)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 7	type	varchar(191)	utf8mb4_unicode_ci		No	Super Admin	Admin Super Admin		Change Drop More
<input type="checkbox"/> 8	remember_token	varchar(100)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
<input type="checkbox"/> 9	created_at	timestamp			Yes	NULL			Change Drop More
<input type="checkbox"/> 10	updated_at	timestamp			Yes	NULL			Change Drop More

● Category Table

Table structure									
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(10)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(191)	utf8mb4_unicode_ci		No	None			Change Drop More
3	description	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
4	image	varchar(191)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
5	parent_id	int(11)			Yes	NULL			Change Drop More
6	created_at	timestamp			Yes	NULL			Change Drop More
7	updated_at	timestamp			Yes	NULL			Change Drop More

● Brand Table

Table structure									
#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(10)		UNSIGNED	No	None		AUTO_INCREMENT	Change Drop More
2	name	varchar(191)	utf8mb4_unicode_ci		No	None			Change Drop More
3	description	text	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
4	image	varchar(191)	utf8mb4_unicode_ci		Yes	NULL			Change Drop More
5	created_at	timestamp			Yes	NULL			Change Drop More
6	updated_at	timestamp			Yes	NULL			Change Drop More

Product Table consists details of Product who have applied-

- User id – Unique id taken by the individual farmer
 - User Phone - Farmer contact number
 - Name- Name of the farmer
 - City - City to which farmer belongs
 - State - State to which farmer belongs
 - Email- Email id of the user

3.4 DFD and Flow Graph

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation. Data flow diagrams can be divided into logical and physical. The logical data flow diagram describes flow of data through a system to perform certain functionality of a business. The physical data flow diagram describes the implementation of the logical data flow.

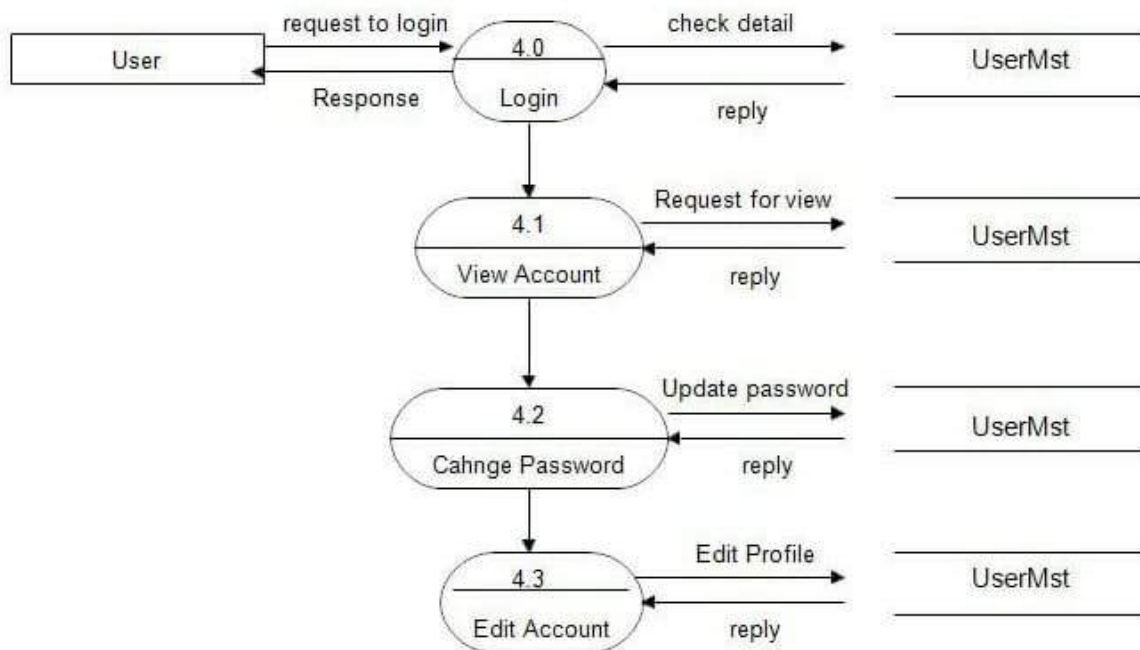
DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer. Structure of DFD allows starting from a broad overview and expand it to a hierarchy of detailed diagrams. DFD has often been used due to the following reasons:

- Logical information flow of the system
- Determination of physical system construction requirements
- Simplicity of notation
- Establishment of manual and automated systems requirements

3.4.1 Process

A process receives input data and produces output with a different content or form. Processes can be as simple as collecting input data and saving in the database, or it can be complex as producing a report containing monthly sales of all retail stores in the northwest region.

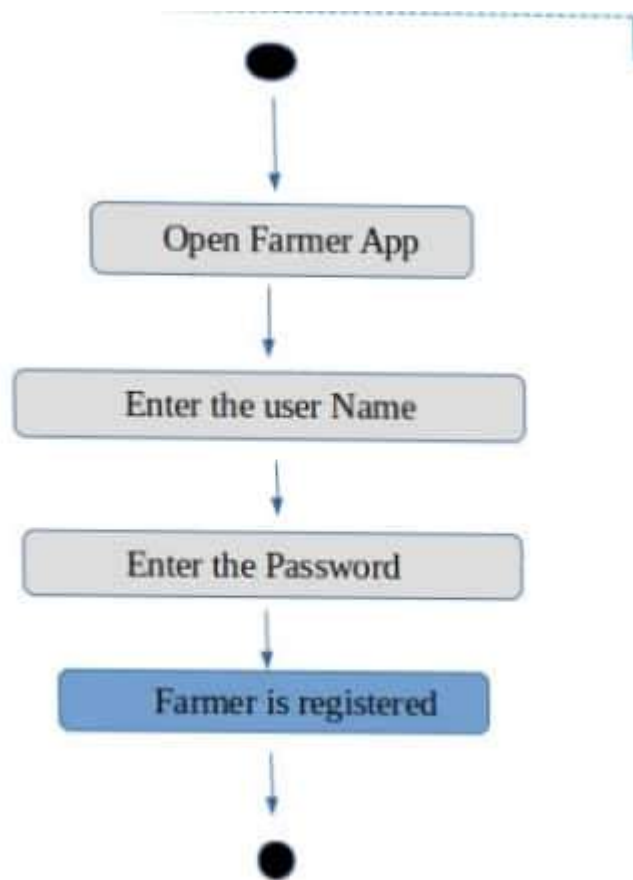
- **Farmer Login Data Flow**



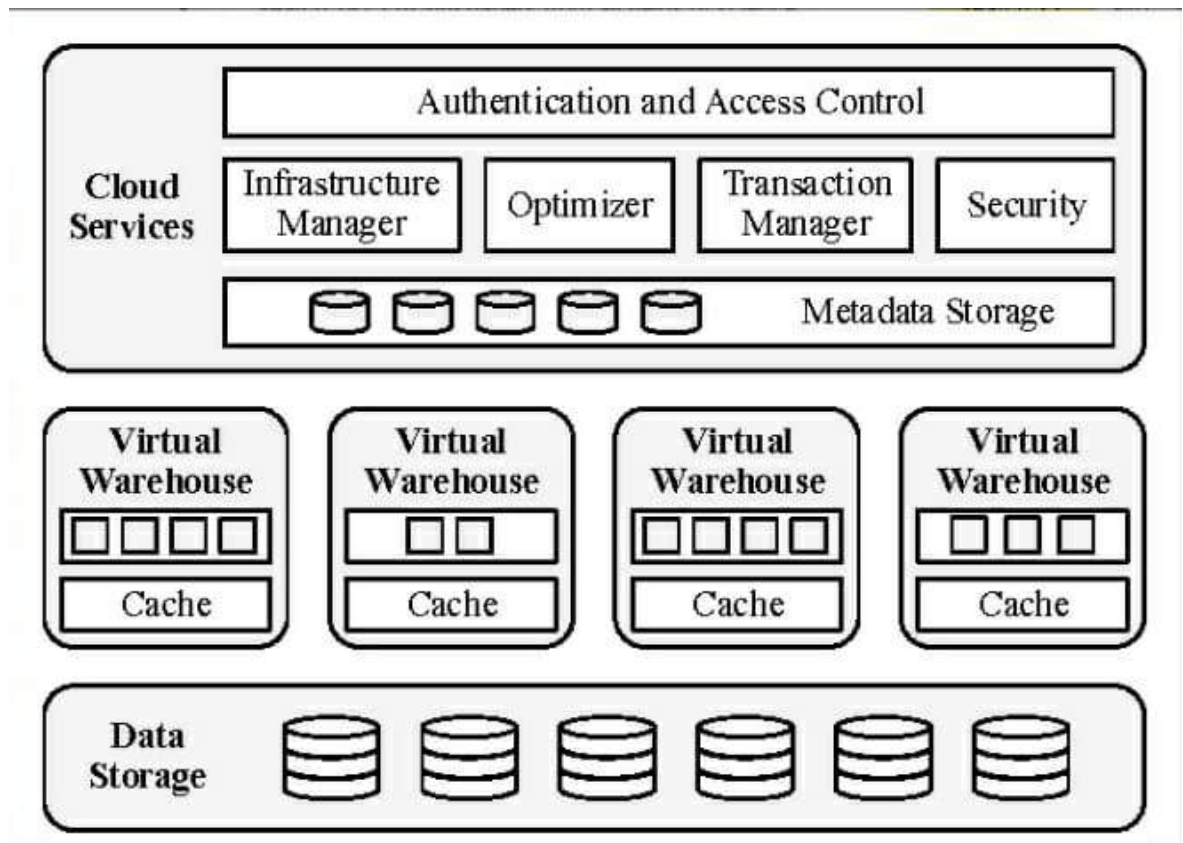
- **BigShop Data Flow**

- This project starts with Registration of Farmer in which Farmer enters username and password and other details.
- If Farmer registered then he/she can login to the system by providing username and password.
- Farmer store the details of Product, emailId, city, state etc.
- Farmer send Email alerts to list of selected Products in each round.
- Farmer can see the applicant details and status and send confirmation or rejection letter.

- **Reactjs Farmer project Authentication flow**



- **Database Table Information**



Farmer project is developed for farmers to conduct the Farmer process and assign the positions to selected Products. This project provides Farmer to store and manage data of objects and conduct the product process very easily and smoothly. There is no need to make extra sheets, to store the details of the Products. This project provides you to store all the data of Products on cloud storage. This project works on 5 layer model of cloud which are defined as follows:

- **Cloud project Layer** Cloud project layer provides the most visible layer to the end users of the cloud. Farmer project provides an user interface to interact with users.
- **Cloud Software Environment Layer** The cloud software environment layer(also dubbed the software platform layer). The users of this layer are

cloud projects' developers, implementing their projects for and deploying them on the cloud. In the Farmer project Parseuser is a project development environment.

- **Cloud Software Infrastructure Layer** The cloud software infrastructure layer provides fundamental resources to other higher level layers. Cloud services offered in this layer can be categorized into: computational resources, data storage, and communications. In the Farmer project Reactjs manages the infrastructure of auto scaling and load balancing on the basis of the number of working users.
- **Software Kernel** This cloud layer provides the basic software management for the physical servers that compose the cloud. Software kernels at this level can be implemented as an OS kernel, hypervisor, and virtual machine monitor and/or clustering middleware.
- **Hardware and Firmware** The bottom layer of the cloud stack in our proposed system is the actual physical hardware and switches that form the backbone of the cloud. In this regard, users of this layer of the cloud are normally big enterprises with huge IT requirements in need of subleasing Hardware as a Service (HaaS). Farmer project is developed in the Sales force cloud computing environment known as Parseuser development environment. In the Parseuser environment you can develop and deploy projects. Reactjs provides an app store known as Appexchange just like the play store of Google Reactjs Marketplace. Reactjs AppExchange is Reactjs.com's cloud computing marketplace through which end users can access, download and install software apps. You can download the BigShop from Appexchange. To download the project first user need to register on Reactjs. After Registration users are able to download and install the projects.

CHAPTER 4

Form Design (Screenshots)

4.1 Home Page

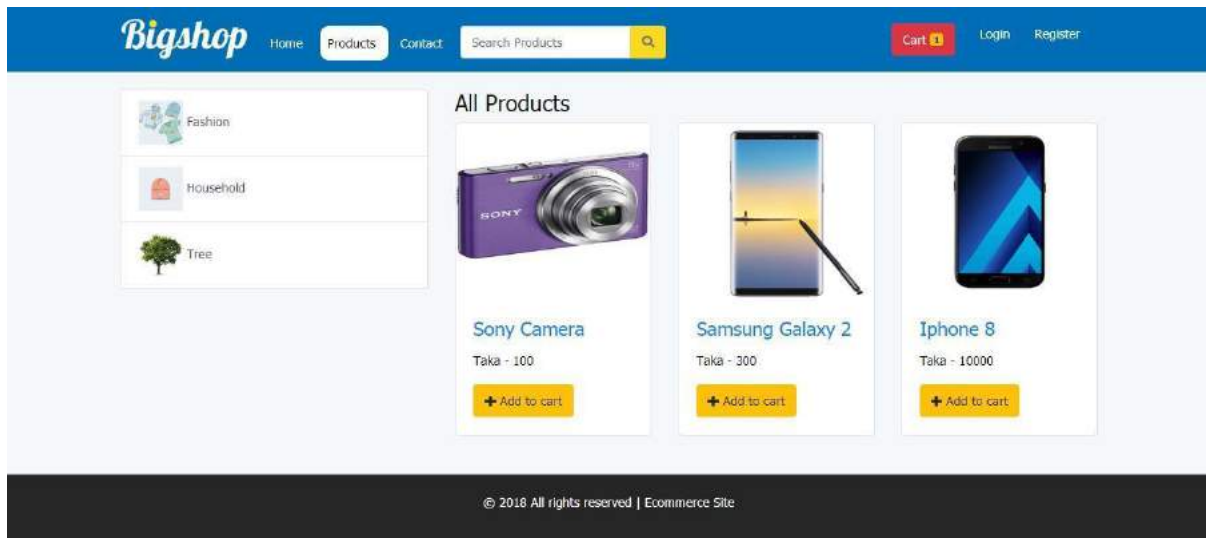


The First Page rendered when the project opened. Farmers have to register or login to sell and buy products.

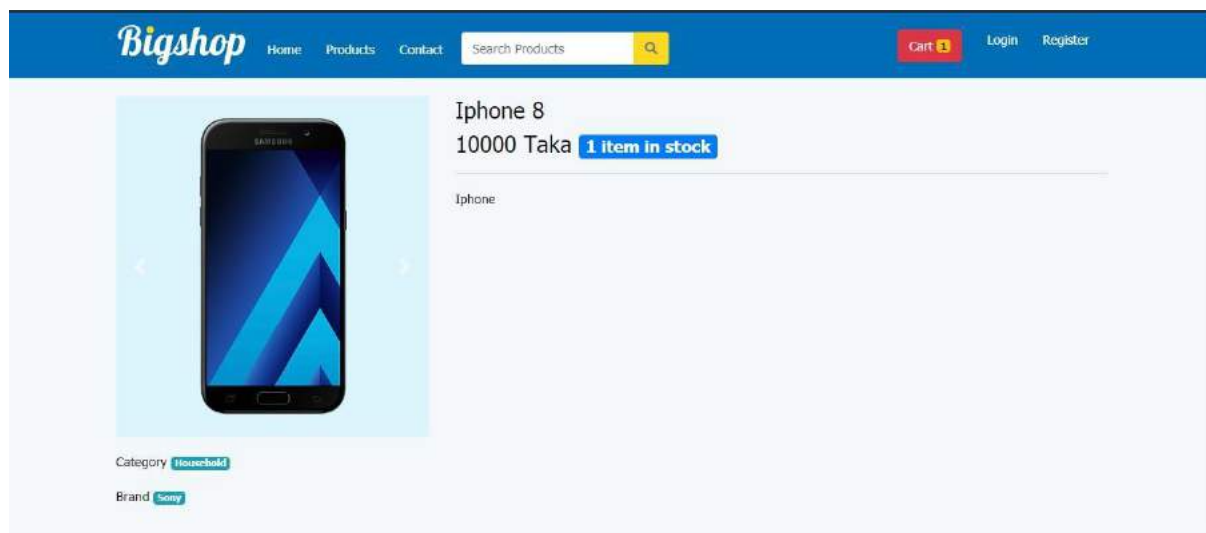
On clicking Top Right button(Login), Login Page rendered which Consists

- Username.
- Password
- Login Button
- Sign Up

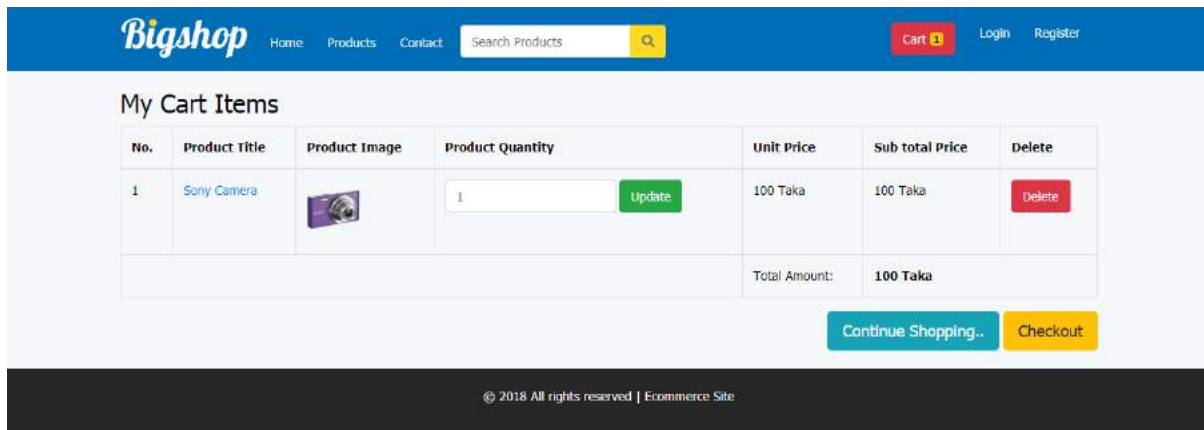
- Shop Page



- Single Product Page



- Cart Page



● Checkout Page

Confirm Items

Sony Camera - 100 taka - 1 item

[Change Cart items](#)

Total Price : 100 Taka

Total Price with shipping cost: 200 Taka

Shipping Address

Receiver Name

E-Mail Address

Phone No

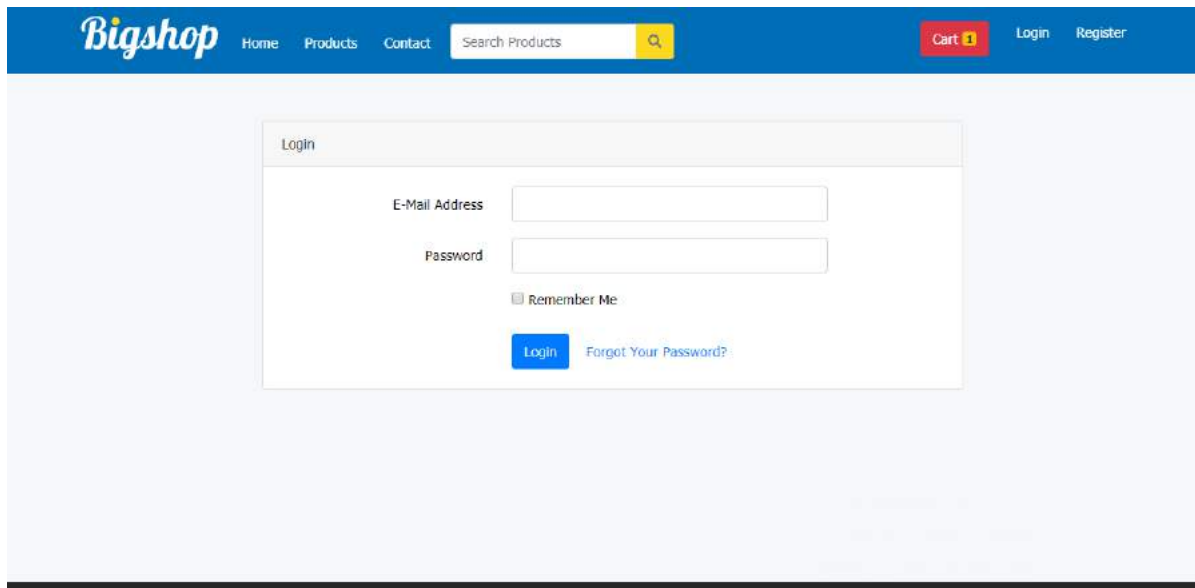
Additional Message (optional)

Shipping Address (*)

Select a payment method

[Order Now](#)

● User Login Page



The screenshot shows the Bigshop website's login interface. The header is a blue bar with the Bigshop logo, navigation links (Home, Products, Contact), a search bar, and a shopping cart icon. The main content area features a light blue background with a central white login box. The box is titled 'Login' and contains fields for 'E-Mail Address' and 'Password'. Below these fields is a 'Remember Me' checkbox and a blue 'Login' button. A link for 'Forgot Your Password?' is also present. At the bottom of the page, there is a row of small, faint links: 'Home', 'About Us', 'Contact Us', 'Privacy Policy', 'Terms & Conditions', 'FAQ', 'Sitemap', and 'RSS Feeds'.

Bigshop Home Products Contact Search Products Cart 1 Login Register

Login

E-Mail Address

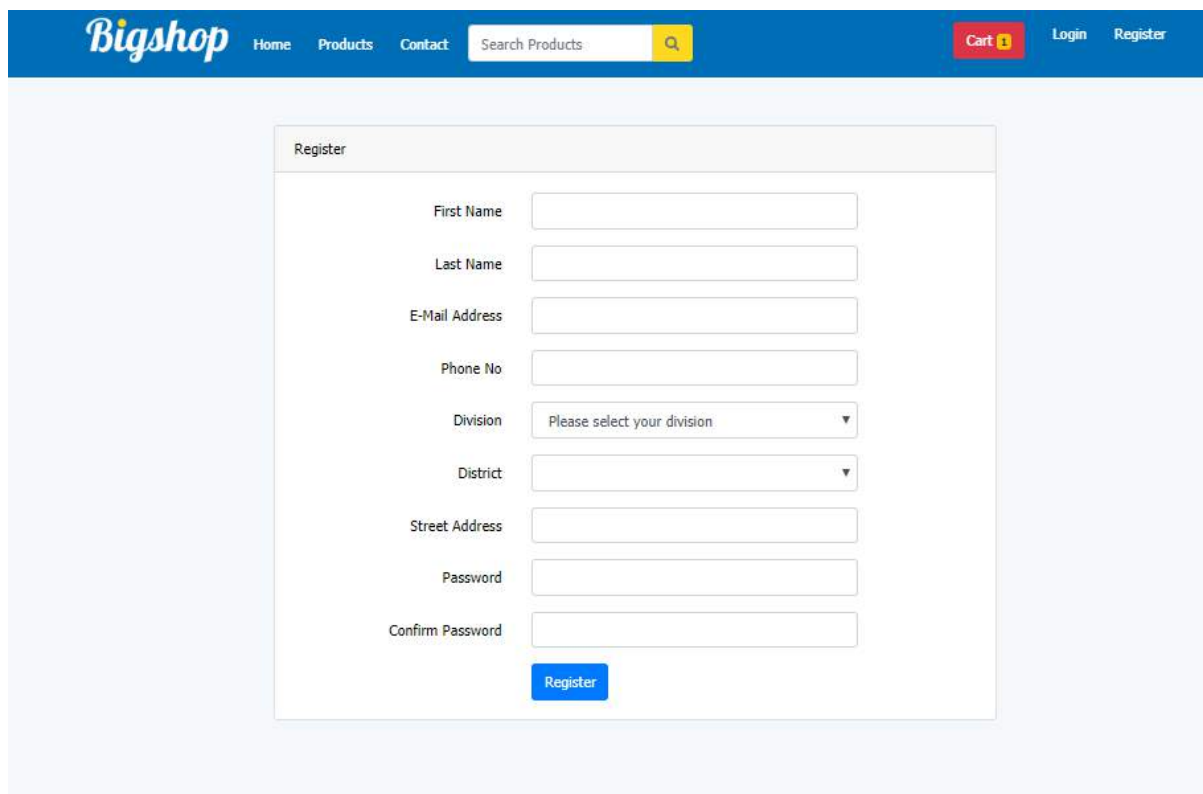
Password

☐ Remember Me

Login Forgot Your Password?

Home About Us Contact Us Privacy Policy Terms & Conditions FAQ Sitemap RSS Feeds

- User Registration Page



The screenshot displays the Bigshop website's registration form. The header is identical to the login page. The main content area has a light blue background with a central white registration box titled 'Register'. The form includes fields for 'First Name', 'Last Name', 'E-Mail Address', 'Phone No', 'Division' (a dropdown menu with the text 'Please select your division'), 'District' (a dropdown menu), 'Street Address', 'Password', and 'Confirm Password'. A blue 'Register' button is located at the bottom of the form.

Bigshop Home Products Contact Search Products Cart 1 Login Register

Register

First Name

Last Name

E-Mail Address

Phone No

Division Please select your division ▼

District ▼

Street Address

Password

Confirm Password

Register

- **SIGNUP Responsive**

- After the farmer provides the Username and password then the contact information is to be provided by the farmer.
- Then the farmer should provide the country in which he resides.
 - Then the city to which he belongs.
 - Then the personal phone number.
 - And then the pincode for the ease of transport.
 - And if the farmer wishes to personalise the home page he/she could add the profile picture.

- **Seller Activity**

- The Seller page shows the name of the buyer and the seller and by tabing upon their name you could see their feed.
- And if the desired result is found then they could contact the buyer and the seller, clarify their query .

CHAPTER 5

Coding

5.1 Login Page

- **Starting Activity**

```
/*  
 * Copyright (c) 2015-present, Parse, LLC.  
 * All rights reserved.  
 *  
 * This source code is licensed under the BSD-style license found in the  
 * LICENSE file in the root directory of this source tree. An additional  
grant  
 * of patent rights can be found in the PATENTS file in the same  
directory.  
 */  
  
package com.parse.starter;  
  
  
import Reactjs.app.Application;  
import Reactjs.util.Log;  
  
  
import com.parse.Parse;  
import com.parse.ParseACL;  
import com.parse.ParseException;  
import com.parse.ParseObject;  
import com.parse.ParseUser;  
import com.parse.SaveCallback;
```

```

public class StarterApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();

        // Enable Local Datastore.
        Parse.enableLocalDatastore(this);

        // Add your initialization code here
        Parse.initialize(new
        Parse.Configuration.Builder(getApplicationContext())
            .applicationId("myappID")
            .clientId("j5wUUU1hGFqN")
            .server("http://18.222.3.228/parse/")
            .build()
        );

        //ParseUser.enableAutomaticUser();
        ParseACL defaultACL = new ParseACL();
        defaultACL.setPublicReadAccess(true);
        defaultACL.setPublicWriteAccess(true);
        ParseACL.setDefaultACL(defaultACL, true);

    }
}

```

}

- **Main Activity**

```
/*
 * Copyright (c) 2015-present, Parse, LLC.
 * All rights reserved.
 *
 * This source code is licensed under the BSD-style license found in the
 * LICENSE file in the root directory of this source tree. An additional grant
 * of patent rights can be found in the PATENTS file in the same directory.
 */

package com.parse.starter;

import Reactjs.Manifest;
import Reactjs.content.Intent;
import Reactjs.content.pm.PackageManager;
import Reactjs.os.Bundle;
import Reactjs.support.v7.app.AppCompatActivity;
import Reactjs.util.Log;
import Reactjs.view.View;
import Reactjs.widget.EditText;
import Reactjs.widget.Switch;
import Reactjs.widget.Toast;

import com.parse.FindCallback;
import com.parse.GetCallback;
import com.parse.LogInCallback;
import com.parse.LogOutCallback;
```

```
import com.parse.Parse;
import com.parse.ParseAnalytics;
import com.parse.ParseAnonymousUtils;
import com.parse.ParseException;
import com.parse.ParseObject;
import com.parse.ParseQuery;
import com.parse.ParseUser;
import com.parse.SaveCallback;
import com.parse.SignUpCallback;
```

```
import java.util.List;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    EditText userName;
```

```
    EditText password;
```

```
    public void showUserList () {
```

```
        Intent intent = new Intent(getApplicationContext(),UserListActivity.class);
```

```
        startActivity(intent);
```

```
    }
```

```
    public void getUserDetails() {
```

```
        Intent intent = new Intent(getApplicationContext(),GetUserDetails.class);
```

```
        startActivity(intent);
```

```
    }
```



```
@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);
```

```
ParseAnalytics.trackAppOpenedInBackground(getIntent());  
  
}
```

```
public void signup(View view) {  
    userName = findViewById(R.id.userName);  
    password = findViewById(R.id.password);  
  
    final String tempUserName = userName.getText().toString();  
    final String tempPassword = password.getText().toString();  
  
    if ( tempUserName.matches("") || tempPassword.matches("")) {  
        Toast.makeText(this, "You must enter both username and password",  
Toast.LENGTH_SHORT).show();  
    } else {  
  
        ParseUser user = new ParseUser();  
        user.setUsername(tempUserName);
```

```

        user.setPassword(tempPassword);
        ParseUser.logOutInBackground();

        user.signUpInBackground(new SignUpCallback() {
            @Override
            public void done(ParseException e) {

                if (e == null) {
                    Toast.makeText(MainActivity.this, "The sign up is successful",
Toast.LENGTH_SHORT).show();
                    Log.i("SignUpRequest", "The sign up is successful");
                    getUserDetails();
                } else {
                    Log.i("SignUpRequest", "Error occurred in the sign up
process"+e.getMessage());
                    Toast.makeText(MainActivity.this, e.getMessage(),
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}

```

```

public void login(View view) {
    userName = findViewById(R.id.userName);
    password = findViewById(R.id.password);
    final String tempUserName = userName.getText().toString();

```

```

final String tempPassword = password.getText().toString();

if (tempUserName.matches("") || tempPassword.matches("")) {
    Toast.makeText(this, "You must enter both username and password",
Toast.LENGTH_SHORT).show();
} else {
    ParseUser.logInInBackground(tempUserName, tempPassword, new
LoginCallback() {
        @Override
        public void done(ParseUser user, ParseException e) {
            if (e == null) {
                Toast.makeText(MainActivity.this, "Login is successful",
Toast.LENGTH_SHORT).show();
                Log.i("LoginRequest", "Login is success. " + user.getUsername()
+ " is logged in.");
                showUserList();
            } else {
                Toast.makeText(MainActivity.this, e.getMessage(),
Toast.LENGTH_SHORT).show();
                Log.i("LoginRequest", e.getMessage());
            }
        }
    });
}
}
}

```

- **Main Activity Layout**

<!--

~ Copyright (c) 2015-present, Parse, LLC.

~ All rights reserved.

~

~ This source code is licensed under the BSD-style license found in the

~ LICENSE file in the root directory of this source tree. An additional
grant

~ of patent rights can be found in the PATENTS file in the same
directory.

-->

<Reactjs.support.constraint.ConstraintLayout

xmlns:Reactjs="http://schemas.Reactjs.com/apk/res/Reactjs"

xmlns:app="http://schemas.Reactjs.com/apk/res-auto"

xmlns:tools="http://schemas.Reactjs.com/tools"

Reactjs:layout_width="match_parent"

Reactjs:layout_height="match_parent"

Reactjs:background="#FFFFFF"

Reactjs:paddingLeft="@dimen/activity_horizontal_margin"

Reactjs:paddingTop="@dimen/activity_vertical_margin"

Reactjs:paddingRight="@dimen/activity_horizontal_margin"

Reactjs:paddingBottom="@dimen/activity_vertical_margin"

tools:context=".MainActivity">

<TextView

Reactjs:id="@+id/appname"

```
Reactjs:layout_width="304dp"
Reactjs:layout_height="54dp"
Reactjs:text="FarmerApp"
Reactjs:textColor="#8BC34A"
Reactjs:textSize="42sp"
Reactjs:textStyle="bold|italic"
app:fontFamily="@font/berkshire_swash"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="1.0"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.023" />
```

<Button

```
Reactjs:id="@+id/signup"
Reactjs:layout_width="87dp"
Reactjs:layout_height="49dp"
Reactjs:layout_marginBottom="72dp"
Reactjs:background="#C7F0CE"
Reactjs:onClick="signup"
Reactjs:text="Signup"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.468"
app:layout_constraintStart_toEndOf="@+id/login" />
```

<EditText

```
Reactjs:id="@+id/userName"
```

```
Reactjs:layout_width="159dp"
Reactjs:layout_height="wrap_content"
Reactjs:ems="10"
Reactjs:hint="Username"
Reactjs:inputType="textPersonName"
Reactjs:textColor="#1F5A20"
app:layout_constraintBottom_toTopOf="@+id/password"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.448"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.94" />
```

<EditText

```
Reactjs:id="@+id/password"
Reactjs:layout_width="154dp"
Reactjs:layout_height="47dp"
Reactjs:layout_marginBottom="176dp"
Reactjs:ems="10"
Reactjs:hint="Password"
Reactjs:inputType="textPassword"
Reactjs:textColor="#1F5A20"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintHorizontal_bias="0.447"
app:layout_constraintStart_toStartOf="parent" />
```

<Button

```
Reactjs:id="@+id/login"
```

```

Reactjs:layout_width="wrap_content"
Reactjs:layout_height="wrap_content"
Reactjs:layout_marginStart="76dp"
Reactjs:layout_marginLeft="76dp"
Reactjs:background="#C7F0CE"
Reactjs:onClick="login"
Reactjs:shadowColor="#8FD665"
Reactjs:text="Login"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/password"
app:layout_constraintVertical_bias="0.44" />

```

```

<ImageView
    Reactjs:id="@+id/imageViewGetPhoto"
    Reactjs:layout_width="286dp"
    Reactjs:layout_height="262dp"
    app:layout_constraintBottom_toTopOf="@+id/userName"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.496"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="1.0"
    app:srcCompat="@drawable/logo" />
</Reactjs.support.constraint.ConstraintLayout>

```

5.2 Signup Page

- **User Detail Activity**

```
package com.parse.starter;

import Reactjs.Manifest;
import Reactjs.content.Intent;
import Reactjs.content.pm.PackageManager;
import Reactjs.graphics.Bitmap;
import Reactjs.icu.text.TimeZoneFormat;
import Reactjs.net.Uri;
import Reactjs.provider.MediaStore;
import Reactjs.support.annotation.NonNull;
import Reactjs.support.annotation.Nullable;
import Reactjs.support.v7.app.AppCompatActivity;
import Reactjs.os.Bundle;
import Reactjs.util.Log;
import Reactjs.view.View;
import Reactjs.widget.EditText;
import Reactjs.widget.ImageView;
import Reactjs.widget.TextView;
import Reactjs.widget.Toast;

import com.parse.ParseException;
import com.parse.ParseFile;
import com.parse.ParseObject;
import com.parse.ParseUser;
import com.parse.SaveCallback;
```



```

import java.io.ByteArrayOutputStream;
import java.io.IOException;

public class GetUserDetails extends AppCompatActivity {

    public void getPhoto () {

        Intent intent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        startActivityForResult(intent,1);

    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);

        if (requestCode == 1){

            if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED){

                getPhoto();

            }

        }
    }
}

```

```
EditText userName, country, state , city , pincode, phone;  
Bitmap bitmap;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_get_user_details);  
    userName = findViewById(R.id.usernamegetdetailseditText);  
    userName.setText(ParseUser.getCurrentUser().getUsername());  
}
```

```
public void showUserList () {
```

```
    Intent intent = new  
Intent(getApplicationContext(),UserListActivity.class);  
    startActivity(intent);  
}
```

```
public void Submit(View view) {
```

```
    country = findViewById(R.id.countryedittext);  
    state = findViewById(R.id.statedittext);  
    city = findViewById(R.id.cityedittext);  
    pincode = findViewById(R.id.pincodeedittext);  
    phone = findViewById(R.id.phonenumberedittext);
```

```
ParseObject userDetails = new ParseObject("UserDetails");
```

```
userDetails.put("username",ParseUser.getCurrentUser().getUsername());  
userDetails.put("country",country.getText().toString());  
userDetails.put("state",state.getText().toString());  
userDetails.put("city",city.getText().toString());  
userDetails.put("pincode",pincode.getText().toString());  
userDetails.put("phone",phone.getText().toString());
```

```
ByteArrayOutputStream stream = new ByteArrayOutputStream();
```

```
bitmap.compress(Bitmap.CompressFormat.PNG,100,stream);
```

```
byte[] byteArray = stream.toByteArray();
```

```
ParseFile file = new ParseFile("profileimage.png", byteArray);  
userDetails.put("profileImage",file);
```

```
userDetails.saveInBackground(new SaveCallback() {  
    @Override  
    public void done(ParseException e) {  
        if (e == null){  
            Log.d("GetUserDetails","User detil is susscesfull");
```

```

        Toast.makeText(GetUserDetails.this, "Sign up is Successful",
Toast.LENGTH_SHORT).show();
        showUserList();
    } else{
        Log.d("GetUserDetails","Not saved successfully");
        Toast.makeText(GetUserDetails.this, e.getMessage(),
Toast.LENGTH_SHORT).show();
    }
}
});

```

```

}

public void Image(View view) {

    if
(checkSelfPermission(Manifest.permission.READ_EXTERNAL_STOR
AGE) != PackageManager.PERMISSION_GRANTED){

        requestPermissions(new
String[]{Manifest.permission.READ_EXTERNAL_STORAGE},1 );

    } else {

        getPhoto();
    }
}

```

```

    }

}

@Override
protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1 && resultCode == RESULT_OK && data
!=null) {

        Uri selectedImage = data.getData();

        try {

            bitmap =
MediaStore.Images.Media.getBitmap(this.getContentResolver(),selectedI
mage);

            Log.i("Photo", "Received");

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

}

- **User Detail Layout**

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:Reactjs="http://schemas.Reactjs.com/apk/res/Reactjs"
    xmlns:app="http://schemas.Reactjs.com/apk/res-auto"
    xmlns:tools="http://schemas.Reactjs.com/tools"
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="match_parent"
    tools:context=".GetUserDetails">
```

```
<LinearLayout
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="match_parent"
    Reactjs:orientation="vertical"
    Reactjs:paddingTop="55dp">
```

```
<LinearLayout
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="wrap_content"
    Reactjs:orientation="horizontal">
```

```

<TextView
    Reactjs:id="@+id/appname"
    Reactjs:layout_width="276dp"
    Reactjs:layout_height="91dp"
    Reactjs:gravity="center"
    Reactjs:text="FarmerApp"
    Reactjs:textAlignment="center"
    Reactjs:textColor="#8BC34A"
    Reactjs:textSize="36sp"
    Reactjs:textStyle="bold|italic"
    app:fontFamily="@font/berkshire_swash" />
<ImageView
    Reactjs:id="@+id/imageView2"
    Reactjs:layout_width="98dp"
    Reactjs:layout_height="91dp"
    app:srcCompat="@drawable/logo" />
</LinearLayout>

```

```

<LinearLayout
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="wrap_content"
    Reactjs:orientation="horizontal"
    Reactjs:paddingTop="25dp">

    <TextView
        Reactjs:id="@+id/usernamegetdetails"
        Reactjs:layout_width="wrap_content"
        Reactjs:layout_height="wrap_content"
        Reactjs:layout_weight="1"

```

```
Reactjs:text="User Name"
Reactjs:textColor="#1F5A20"
Reactjs:textSize="24sp"
Reactjs:textStyle="italic"
app:fontFamily="@font/aladin"
Reactjs:shadowColor="#8FD665"
Reactjs:gravity="center"/>
```

```
<EditText
```

```
Reactjs:id="@+id/usernamegetdetailseditText"
Reactjs:layout_width="154dp"
Reactjs:layout_height="47dp"
Reactjs:layout_weight="1"
Reactjs:ems="10"
Reactjs:textColor="#1F5A20"
Reactjs:inputType="textPersonName"
Reactjs:hint="Name"
Reactjs:gravity="center"/>
```

```
</LinearLayout>
```

```
<LinearLayout
```

```
Reactjs:layout_width="match_parent"
Reactjs:layout_height="wrap_content"
Reactjs:orientation="horizontal"
Reactjs:paddingTop="5dp">
```

```
<TextView
```

```
Reactjs:id="@+id/country"
Reactjs:layout_width="wrap_content"
```



```
Reactjs:layout_height="wrap_content"
Reactjs:layout_weight="1"
Reactjs:text="Country"
Reactjs:textColor="#1F5A20"
Reactjs:textSize="24sp"
Reactjs:textStyle="italic"
app:fontFamily="@font/aladin"
Reactjs:shadowColor="#8FD665"
Reactjs:gravity="center"/>
```

```
<EditText
```

```
Reactjs:id="@+id/countryedittext"
Reactjs:layout_width="154dp"
Reactjs:layout_height="47dp"
Reactjs:layout_weight="1"
Reactjs:ems="10"
Reactjs:textColor="#1F5A20"
Reactjs:inputType="textPersonName"
Reactjs:hint="Country"
Reactjs:gravity="center"/>
```

```
</LinearLayout>
```

```
<LinearLayout
```

```
Reactjs:layout_width="match_parent"
Reactjs:layout_height="wrap_content"
Reactjs:orientation="horizontal"
Reactjs:paddingTop="5dp">
```

```
<TextView
```

```
Reactjs:id="@+id/state"
Reactjs:layout_width="wrap_content"
Reactjs:layout_height="wrap_content"
Reactjs:layout_weight="1"
Reactjs:text="State"
Reactjs:textColor="#1F5A20"
Reactjs:textSize="24sp"
Reactjs:textStyle="italic"
app:fontFamily="@font/aladin"
Reactjs:shadowColor="#8FD665"
Reactjs:gravity="center"/>
```

```
<EditText
```

```
Reactjs:id="@+id/statedittext"
Reactjs:layout_width="154dp"
Reactjs:layout_height="47dp"
Reactjs:layout_weight="1"
Reactjs:ems="10"
Reactjs:textColor="#1F5A20"
Reactjs:inputType="textPersonName"
Reactjs:hint="State"
Reactjs:gravity="center"/>
```

```
</LinearLayout>
```

```
<LinearLayout
```

```
Reactjs:layout_width="match_parent"
Reactjs:layout_height="wrap_content"
Reactjs:orientation="horizontal"
Reactjs:paddingTop="5dp">
```

```
<TextView
    Reactjs:id="@+id/city"
    Reactjs:layout_width="wrap_content"
    Reactjs:layout_height="wrap_content"
    Reactjs:layout_weight="1"
    Reactjs:text="City"
    Reactjs:textColor="#1F5A20"
    Reactjs:textSize="24sp"
    Reactjs:textStyle="italic"
    app:fontFamily="@font/aladin"
    Reactjs:shadowColor="#8FD665"
    Reactjs:gravity="center"/>
```

```
<EditText
    Reactjs:id="@+id/cityedittext"
    Reactjs:layout_width="154dp"
    Reactjs:layout_height="47dp"
    Reactjs:layout_weight="1"
    Reactjs:ems="10"
    Reactjs:textColor="#1F5A20"
    Reactjs:inputType="textPersonName"
    Reactjs:hint="City"
    Reactjs:gravity="center"/>
```

```
</LinearLayout>
```

```
<LinearLayout
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="wrap_content"
```

Reactjs:orientation="horizontal"

Reactjs:paddingTop="5dp">

<TextView

Reactjs:id="@+id/phonenummer"

Reactjs:layout_width="wrap_content"

Reactjs:layout_height="wrap_content"

Reactjs:layout_weight="1"

Reactjs:text="Phone"

Reactjs:textColor="#1F5A20"

Reactjs:textSize="24sp"

Reactjs:textStyle="italic"

app:fontFamily="@font/aladin"

Reactjs:shadowColor="#8FD665"

Reactjs:gravity="center"/>

<EditText

Reactjs:id="@+id/phonenumberedittext"

Reactjs:layout_width="154dp"

Reactjs:layout_height="47dp"

Reactjs:layout_weight="1"

Reactjs:ems="10"

Reactjs:textColor="#1F5A20"

Reactjs:inputType="textPersonName"

Reactjs:hint="Phone"

Reactjs:gravity="center"/>

</LinearLayout>

<LinearLayout

```
Reactjs:layout_width="match_parent"  
Reactjs:layout_height="wrap_content"  
Reactjs:orientation="horizontal"  
Reactjs:paddingTop="5dp">
```

```
<TextView  
    Reactjs:id="@+id/pincode"  
    Reactjs:layout_width="wrap_content"  
    Reactjs:layout_height="wrap_content"  
    Reactjs:layout_weight="1"  
    Reactjs:text="Pincode"  
    Reactjs:textColor="#1F5A20"  
    Reactjs:textSize="24sp"  
    Reactjs:textStyle="italic"  
    app:fontFamily="@font/aladin"  
    Reactjs:shadowColor="#8FD665"  
    Reactjs:gravity="center"/>
```

```
<EditText  
    Reactjs:id="@+id/pincodeedittext"  
    Reactjs:layout_width="154dp"  
    Reactjs:layout_height="47dp"  
    Reactjs:layout_weight="1"  
    Reactjs:ems="10"  
    Reactjs:textColor="#1F5A20"  
    Reactjs:inputType="textPersonName"  
    Reactjs:hint="Pincode"  
    Reactjs:gravity="center"/>
```

```
</LinearLayout>
```

```
<LinearLayout  
    Reactjs:layout_width="match_parent"  
    Reactjs:layout_height="wrap_content"  
    Reactjs:orientation="horizontal"  
    Reactjs:paddingTop="5dp">
```

```
</LinearLayout>
```

```
<LinearLayout  
    Reactjs:layout_width="match_parent"  
    Reactjs:layout_height="wrap_content"  
    Reactjs:orientation="horizontal"  
    Reactjs:paddingTop="5dp"  
    Reactjs:gravity="center">>
```

```
<Button  
    Reactjs:id="@+id/image"  
    Reactjs:layout_width="wrap_content"  
    Reactjs:layout_height="wrap_content"  
    Reactjs:background="#C7F0CE"  
    Reactjs:onClick="Image"  
    Reactjs:shadowColor="#8FD665"  
    Reactjs:text="Image"  
    Reactjs:gravity="center"
```

/>

</LinearLayout>

<LinearLayout

Reactjs:layout_width="match_parent"

Reactjs:layout_height="wrap_content"

Reactjs:orientation="horizontal"

Reactjs:paddingTop="45dp"

Reactjs:paddingBottom="45dp"

Reactjs:gravity="center">

<Button

Reactjs:id="@+id/submit"

Reactjs:layout_width="wrap_content"

Reactjs:layout_height="wrap_content"

Reactjs:background="#C7F0CE"

Reactjs:onClick="Submit"

Reactjs:shadowColor="#8FD665"

Reactjs:text="Submit"

Reactjs:gravity="center"

/>

</LinearLayout>

</LinearLayout>

</RelativeLayout>

- **RecyclerView Holder Activity**

:

```
package com.parse.starter;
```

```
import Reactjs.content.Context;
```

```
import Reactjs.support.annotation.NonNull;
```

```
import Reactjs.support.v7.widget.RecyclerView;
```

```
import Reactjs.util.Log;
```

```
import Reactjs.view.LayoutInflater;
```

```
import Reactjs.view.View;
```

```
import Reactjs.view.ViewGroup;
```

```
import Reactjs.widget.ImageView;
```

```
import Reactjs.widget.RelativeLayout;
```

```
import Reactjs.widget.TextView;
```

```
import Reactjs.widget.Toast;
```

```
import com.bumptechnology.glide.Glide;
```

```
import java.util.ArrayList;
```

```
/*public class RecyclerViewAdapter extends
```

```
RecyclerView.Adapter<RecyclerViewAdapter.ViewHolder> {
```



```

private static final String TAG = "RecyclerViewAdapter";

private ArrayList<String> parseImages = new ArrayList<>();
private ArrayList<String> parseUserName = new ArrayList<>();
private ArrayList<String> parseUserDiscription = new ArrayList<>();
private Context context;

public RecyclerViewAdapter(Context context,ArrayList<String>
parseImages, ArrayList<String> parseUserName, ArrayList<String>
parseUserDiscription) {
    this.parseImages = parseImages;
    this.parseUserName = parseUserName;
    this.parseUserDiscription = parseUserDiscription;
    this.context = context;
}

@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup,
int i) {

    View view =
LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.user_list,viewGr
oup,false);
    ViewHolder holder = new ViewHolder(view);

    return holder;
}

```

```

@Override
public void onBindViewHolder(@NonNull ViewHolder viewHolder, final int
i) {

    Log.d(TAG,"onBindViewHolder: called");
    Glide.with(context)
        .asBitmap()
        .load(parseImages.get(i))
        .into(viewHolder.profilepic);

    viewHolder.username.setText(parseUserName.get(i));
    viewHolder.userdiscription.setText(parseUserDiscription.get(i));

    viewHolder.parentLayout.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v) {
            Log.d(TAG,"onClick: clicked on: " + parseUserName.get(i));

            Toast.makeText(context, parseUserName.get(i),
Toast.LENGTH_SHORT).show();
        }
    });

}

@Override
public int getItemCount() {

```

```

        return parseUserName.size();
    }

    public class ViewHolder extends RecyclerView.ViewHolder{

        ImageView profilepic;
        TextView username;
        TextView userdiscription;
        RelativeLayout parentLayout;

        public ViewHolder (View itemView){
            super(itemView);
            profilepic = itemView.findViewById(R.id.profilepic);
            username = itemView.findViewById(R.id.usernameList);
            userdiscription = itemView.findViewById(R.id.userdiscription);
            parentLayout = itemView.findViewById(R.id.parent_layout);

        }

    }
}

```

- **RecyclerView Holder Layout**

```

<?xml version="1.0" encoding="utf-8"?>
<Reactjs.support.constraint.ConstraintLayout
xmlns:Reactjs="http://schemas.Reactjs.com/apk/res/Reactjs"
    xmlns:app="http://schemas.Reactjs.com/apk/res-auto"

```

```
xmlns:tools="http://schemas.Reactjs.com/tools"
Reactjs:layout_width="match_parent"
Reactjs:layout_height="match_parent"
tools:context=".UserListActivity">
```

```
<TextView
    Reactjs:id="@+id/appname"
    Reactjs:layout_width="195dp"
    Reactjs:layout_height="57dp"
    Reactjs:text="FarmerApp"
    Reactjs:textColor="#8BC34A"
    Reactjs:textSize="36sp"
    Reactjs:textStyle="bold|italic"
    app:fontFamily="@font/berkshire_swash"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.159"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.04" />
```

```
<ImageView
    Reactjs:id="@+id/imageView2"
    Reactjs:layout_width="98dp"
    Reactjs:layout_height="91dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.636"
    app:layout_constraintStart_toEndOf="@+id/appname"
```

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintVertical_bias="0.025"
app:srcCompat="@drawable/logo" />
```

```
<TextView
```

```
    Reactjs:id="@+id/heading"
    Reactjs:layout_width="161dp"
    Reactjs:layout_height="36dp"
    Reactjs:layout_marginTop="20dp"
    Reactjs:text="User List"
    Reactjs:textColor="#1F5A20"
    Reactjs:textSize="24sp"
    Reactjs:textStyle="italic"
    app:fontFamily="@font/aladin"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.136"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/appname" />
```

```
<ListView
```

```
    Reactjs:id="@+id/userListView"
    Reactjs:layout_width="410dp"
    Reactjs:layout_height="175dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/heading"
    app:layout_constraintVertical_bias="0.0" />
```

```

<Reactjs.support.v7.widget.RecyclerView
    Reactjs:id="@+id/userRecyclerView"
    Reactjs:layout_width="410dp"
    Reactjs:layout_height="372dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/userListView"
    tools:layout_editor_absoluteX="1dp" />

</Reactjs.support.constraint.ConstraintLayout>

```

5.3 User Feed Page

- **User List Activity**

```

package com.parse.starter;

import Reactjs.support.v7.app.AppCompatActivity;
import Reactjs.os.Bundle;
import Reactjs.support.v7.widget.LinearLayoutManager;
import Reactjs.support.v7.widget.RecyclerView;
import Reactjs.widget.AdapterView;
import Reactjs.widget.AdapterView.OnItemClickListener;
import Reactjs.widget.AdapterView.OnItemSelectedListener;
import Reactjs.widget.AdapterView.OnItemClickListener;
import Reactjs.widget.AdapterView.OnItemSelectedListener;
import Reactjs.widget.Toast;

import com.parse.FindCallback;
import com.parse.ParseException;

```

```

import com.parse.ParseQuery;
import com.parse.ParseUser;

import java.util.ArrayList;
import java.util.List;

public class UserListActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_list);

        final ListView userList = findViewById(R.id.userListView);
        final ArrayList<String> username = new ArrayList<>();

        //final RecyclerView recyclerView =
        findViewById(R.id.userRecyclerView);
        //final RecyclerViewAdapter adapter=new
        RecyclerViewAdapter(this,null,username,null);

        final ArrayAdapter arrayAdapter = new ArrayAdapter(this,
        Reactjs.R.layout.simple_list_item_1,username);

        ParseQuery<ParseUser> query = ParseQuery.getUserQuery();

```

```

query.whereNotEqualTo("username",ParseUser.getCurrentUser().getUsername(
));
query.addAscendingOrder("username");
query.findInBackground(new FindCallback<ParseUser>() {
    @Override
    public void done(List<ParseUser> objects, ParseException e) {

        if (e == null){

            if (objects.size()>0) {
                for (ParseUser object : objects) {
                    username.add(object.getUsername());
                }

                userList.setAdapter(arrayAdapter);
                //recyclerView.setAdapter(adapter);
                //recyclerView.setLayoutManager(new
LinearLayoutManager(this));

            }else{
                Toast.makeText(UserListActivity.this, "No Username is Present",
Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(UserListActivity.this, e.getMessage(),
Toast.LENGTH_SHORT).show();
        }
    }
}

```


});

}

}

- **User List Layout**

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<RelativeLayout
```

```
    xmlns:Reactjs="http://schemas.Reactjs.com/apk/res/Reactjs"
```

```
    xmlns:app="http://schemas.Reactjs.com/apk/res-auto"
```

```
    Reactjs:layout_width="match_parent"
```

```
    Reactjs:layout_height="match_parent"
```

```
    Reactjs:padding="15dp"
```

```
    Reactjs:id="@+id/parent_layout">
```

```
<Reactjs.support.v7.widget.CardView
```

```
    Reactjs:layout_width="match_parent"
```

```
    Reactjs:layout_height="100dp"
```

```
    app:cardBackgroundColor="#F0DAB1"
```

```
    app:cardCornerRadius="30dp"
```

```
    app:cardElevation="10dp"
```

```
    app:contentPadding="5dp" >
```

```
<LinearLayout
```

```
    Reactjs:layout_width="match_parent"
```

```
    Reactjs:layout_height="match_parent"
```

```
Reactjs:orientation="horizontal">
```

```
<ImageView
```

```
Reactjs:id="@+id/profilepic"
```

```
Reactjs:layout_width="213dp"
```

```
Reactjs:layout_height="match_parent"
```

```
Reactjs:layout_weight="1"
```

```
app:srcCompat="@drawable/cast_expanded_controller_seekbar_thumb"
```

```
/>
```

```
<LinearLayout
```

```
Reactjs:layout_width="match_parent"
```

```
Reactjs:layout_height="match_parent"
```

```
Reactjs:layout_weight="1"
```

```
Reactjs:orientation="vertical">
```

```
<Reactjs.support.v7.widget.CardView
```

```
Reactjs:layout_width="match_parent"
```

```
Reactjs:layout_height="35dp"
```

```
app:cardBackgroundColor="#D8814F"
```

```
app:cardCornerRadius="10dp" >
```

```
<TextView
```

```
Reactjs:id="@+id/usernameList"
```

```
Reactjs:layout_width="match_parent"
```

```
Reactjs:layout_height="match_parent"
```

```
Reactjs:text="TextView" />
```

```
</Reactjs.support.v7.widget.CardView>
```

```
<Reactjs.support.v7.widget.CardView
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="match_parent"
    app:cardBackgroundColor="#D5AA6C"
    app:cardCornerRadius="10dp" >
```

```
<TextView
    Reactjs:id="@+id/userdiscription"
    Reactjs:layout_width="match_parent"
    Reactjs:layout_height="match_parent"
    Reactjs:text="TextView" />
```

```
</Reactjs.support.v7.widget.CardView>
</LinearLayout>
```

```
</LinearLayout>
```

```
</Reactjs.support.v7.widget.CardView>
</RelativeLayout>
```

CHAPTER 6

Testing

- **Introduction**

Information Processing has undergone major improvements in the past two decades in both hardware and software. Hardware has decreased in size and price, while providing more and faster processing power. Software has become easier to use, while providing increased capabilities. There is an abundance of products available to assist both end-users and software developers in their work. Software testing, however, has not progressed significantly. It is still largely a manual process conducted as an art rather than a methodology. It is almost an accepted practice to release software that contains defects. Software that is not thoroughly tested is released for production. This is true

for both off-the-shelf software products and custom applications. Software vendor and in-house systems developers release an initial system and then deliver fixes to the code. They continue delivering fixes until they create a new system and stop supporting the old one. The user is then forced to convert to the new system, which again will require fixes. In-house systems developers generally do not provide any better level of support. They require the users to submit Incident Reports specifying the system defects. The Incident Reports are then assigned a priority and the defects are fixed as time and budgets permit.

- **Importance Of Testing**

Testing is difficult. It requires knowledge of the application and the system architecture. The majority of the preparation work is tedious. The test

conditions, test data, and expected results are generally created manually. System testing is also one of the final activities before the system is released for production. There is always pressure to complete systems testing promptly to meet the deadline. Nevertheless, systems testing are important.

In mainframe when the system is distributed to multiple sites, any errors or omissions in the system will affect several groups of users. Any savings realized in downsizing the application will be negated by costs to correct software errors and reprocess information. Software developers must deliver reliable and secure systems that satisfy the user's requirements. A key item in successful systems testing is developing a testing methodology rather than relying on individual style of the test practitioner. The systems testing effort must follow a defined strategy. It must have an objective, a scope, and an approach. Testing is not an art; it is a skill that can be taught. Testing is generally associated with the execution of programs. The emphasis is on the outcome of the testing, rather than what is tested and how it's tested. Testing is not a one-step activity; execute the test. It requires planning and design. The tests should be reviewed prior to execution to verify their accuracy and completeness. They must be documented and saved for reuse. System testing is the most extensive testing of the system. It requires more manpower and machine processing time than any other testing level. It is therefore the most expensive testing level. It is a critical process in system development. It verifies that the system performs the business requirements accurately, completely, and within the required performance limits. It must be thorough, controlled and managed.

- **Testing Definition**

Software development has several levels of testing.

- Unit Testing
- Systems Testing
- Acceptance Testing

- **Unit Testing**

The first level of testing is called unit testing which is done during the development of the system. Unit testing is essential for verification of the code produced during the coding phase. Errors were noted down and corrected immediately. It is performed by the programmer. It uses the program specifications and the program itself as its source. Thus, our modules are individually tested here. There is no formal documentation required for the unit-testing program.

- **Integration Testing**

The second level of testing includes integration testing. Here different dependent modules are assembled and tested for any bugs that may surface due to the integration of modules. Thus, the administrator module and various visa immigration modules are tested here.

- **System Testing**

The third level of testing includes systems testing. Systems testing verify that the system performs the business functions while meeting the specified performance requirements. It is performed by a team consisting of software technicians and users. It uses the Systems Requirements document, the System Architectural Design and Detailed Design Documents, and the Information Systems Department standards as its sources. Documentation is recorded and saved for systems testing.

- **Acceptance Testing**

The final level of testing is the acceptance testing. Acceptance testing provides the users with assurance that the system is ready for production use; it is performed by the users. It uses the System Requirements document as its source. There is no formal documentation required for acceptance testing.

Systems testing is the major testing effort of the project. It is the functional testing of the application and is concerned with following,

- Quality/standards compliance
- Business requirements
- Performance capabilities
- Operational capabilities

Below are defined a few test cases which have been implemented for the various screens. The outputs have been registered and the required changes have been incorporated.

CHAPTER 7

Conclusion and Future Scope

- **Conclusion**

This project BigShop is an Reactjs application where the user is able to sell the crop product to the preferred seller through the online platform and get his fair share of the produce at the optimal price. This BigShop provides them the platform and the means through which they could make their unique account through their user id and password. And upload their contact information and upload the product information with the photograph , and browse through other seller products and compare their price according to it and then place the advertisement on the platform.

A seller could be a buyer also and buy anything of their interest on the BigShop, then find out the information of the seller and contact through it on the different medium and clarify their query and finalise their deal. Earlier the farmer has to go through the middlemen to sell their product and get very low returns after the hard work he has gone through. And the middlemen gets the profit from both the seller and the buyer because both of them have to go through the wishes of the middlemen. But through the BigShop now there is no place of the middlemen. And both of them could reap the reward.

- **Limitaion**

Farmer projects have some limitations as this is the first project. In this project the list searching is done through the farmer manually. Farmers select the list and contact the buyer through another app or medium. The other limitation is there must be an Internet connection and Reactjs phone. Only attract buyers and not actual difference makers.

Is to verify the authenticity of the buyer and the seller. And the assurance of the confirmed payment. Farmers sometimes spend time in contacting the false contact information.

The barrier to entry is too low and anybody can apply to your product.

- **Future Scope**

In the future update of the app we could provide the means through which we could verify the authenticity of the buyer and the seller, and also provide the medium to chat with each other through the app itself and keep the history as the proof if any fraud or malicious activity occurs.

The other facility could be provided to the user of paying and receiving the payment through the different means on the application itself and all the transaction could be saved on the cloud itself for safe keeping.

CHAPTER 8

Bibliography

- ✓ Fresherworld.com
- ✓ Shine.com 44
- ✓ developer.Reactjs.com
- ✓ Reactjs.com
- ✓ CommunityParseuser