

A
PROJECT REPORT ON

AN ANDROID
APPLICATION
FOR FARMER'S
SOCIAL NETWORK

By

CHARVIT ARDESHNA(CE009)(19CEUOS130)
ISHITA CHAUHAN (CE021) (19CEUOS140)

B.Tech CE Semester-VI
Subject: System Design Practice

Guided by:
Prof.Pandav K. Patel
Assistant Professor
Dept. of Comp. Engg.



Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University



**Faculty of Technology
Department of Computer Engineering
Dharmsinh Desai University**

CERTIFICATE

This is to certify that the practical / term work carried out in the
subject of

**CHARVIT ARDESHNA(CE009)(19CEUOS130)
ISHITA CHAUHAN (CE021) (19CEUOS140)**

System Design Practice and recorded in this journal is the
Bonafede work of

of B.Tech semester **VI** in the branch of **Computer Engineering**
during the academic year **2021-2022.**

Prof. Pandav K. Patel
Assistant Professor,
Dept. of Computer Engg.,
Engg.,
Faculty of Technology

Dharmsinh Desai University, Nadiad
University, Nadiad

Dr. C. K. Bhensdadia,
Head,
Dept. of Computer

Faculty of Technology

Dharmsinh Desai

Contents:

1.Abstract	3
2.Introduction	3
3. Software Requirement Specifications	4
4.Design	
1. ER Diagram	6
2 Data Flow Diagram.....	7
3.class Diagram.....	9
4.Data dictionary.....	11
5. Implementation Detail	
1. Modules	14
2.Major Functionality	14
6. Testing	20
7. Screen-Shots	21
8. Conclusion	28
9. Limitation and Future extension.....	28
10.Bibliography	29

Abstract:

Farmer's Social Network is specially created for farmers to gain wealth of knowledge and ideas, opportunity to establish key partnership, opportunity to reach wider consumers, experts in agricultural field.

Introduction:

Farmers Discussion is specially created for farmers. Here they can post their queries. Other users of the application can answer to their queries. Farmers can follow other farmers to view their post. They can search their queries. Farmers can view weather for upcoming days.

Technology/Tools Used:

Technology/Tools:

React Native Expo, NodeJS, MongoDB

Visual studio code, MongoDB atlas

Platform used: Local Development Server

Software Requirement Specifications:

Farmers Social Network

End User:

User

Functional Requirements:

1. Registration:

1.1 Signup User:

A user who has no existing account on our application can register to our app and will be added to database.

Input:

- Username, Email Address, password

Output:

- User redirected to home page

1.2 Login User:

Existing users can login using their credentials.

Input:

- Email, Password

Output:

- If credentials are correct then redirected to home page else error msg will be displayed.

2. Manage Post:

2.1 Add Post:

User needs to write title, Description and choose an image To add post.

Input:

- Upload image, write title and description.

Output:

- The post will be added to database.

2.2 View Post:

Users can see other user's post in feed by following the user and also in post section.

Input:

- Follow user or click on the post.

Output:

- All the post of the friends are shown in feed section and

Post can be viewed by clicking on the post.

2.3 Delete Post:

User can delete the post.

Input:

- Click on delete icon.

Output:

- Post and comments on that post will be deleted.

2.4 Search Post:

User can search post by keyword.

Input:

- User can type keyword.

Output:

- If database contains the post having the same keyword It will be shown.

3. View Profile:

Users can view their profile.

Input:

- Click on view profile icon.

Output:

- User's profile will be displayed.

4. Manage friendship:

4.1 Follow user:

Users can follow other users to show their post in feed section.

Input:

- Click on follow button.

Output:

- Post of that user will be added to feed section.

4.2 unfollow user:

users can unfollow users.

Input:

- Click on unfollow button.

Output:

- Post of that user will be deleted from feed section.

5. Comment Manage:

5.1 Add comment:

Users can add comment on any post.

Input:

- Comment, submit.

Output:

- Comment will be added to database and will be displayed on post.

5.2 Delete comment:

Users can delete comment on any post.

Input:

- Click on delete comment icon.

Output:

- Comment will be deleted from database and will be removed from post.

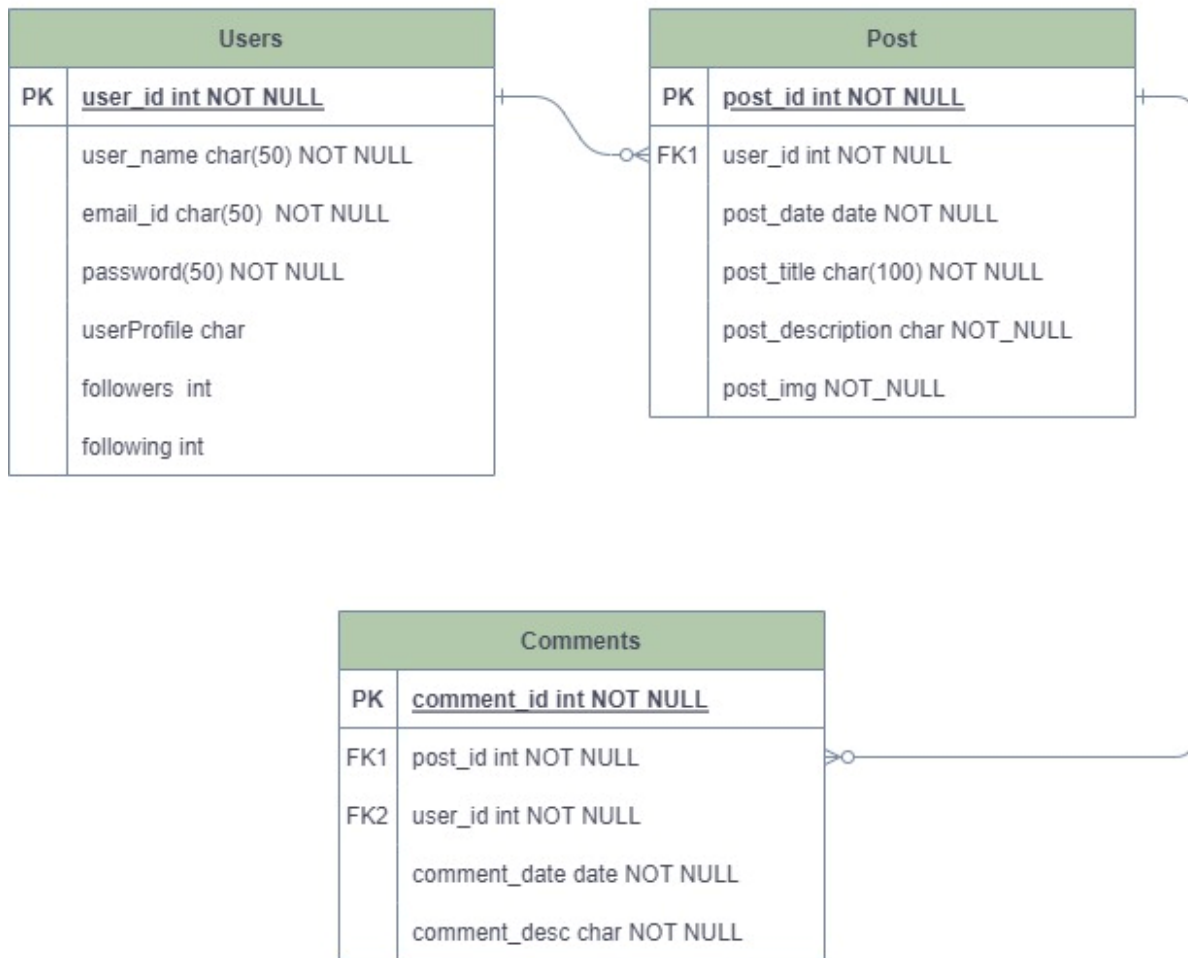
6. View Weather:

Users can view daily, hourly and Weekly weather by clicking on weather option.

Design:

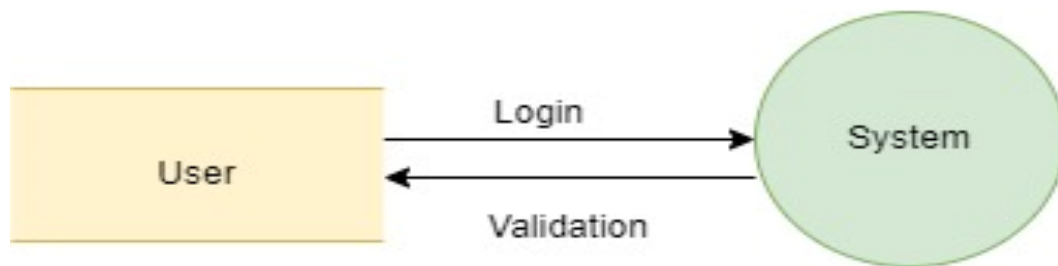
ER diagram:

ER Diagram

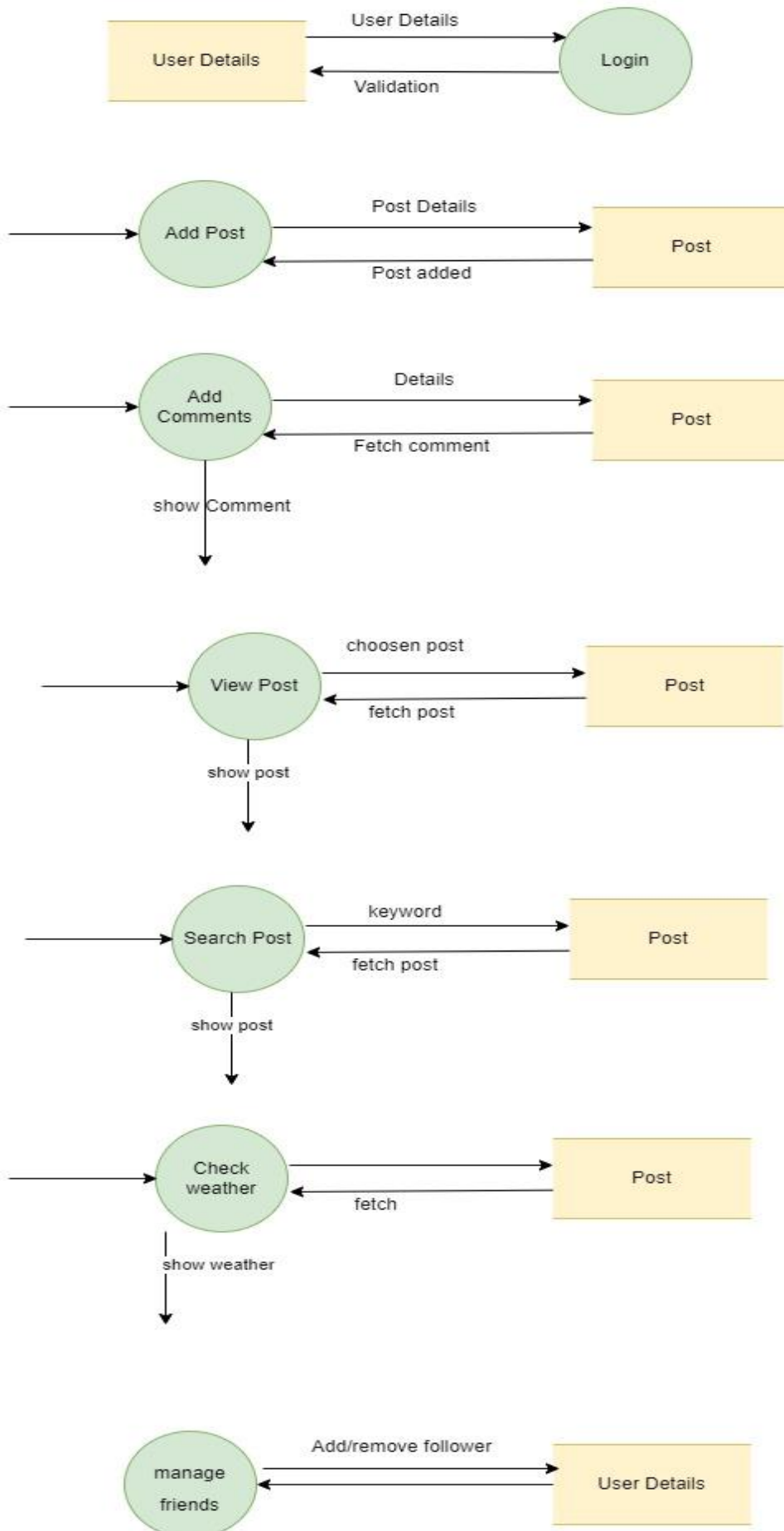


Dataflow diagram:

LEVEL -0

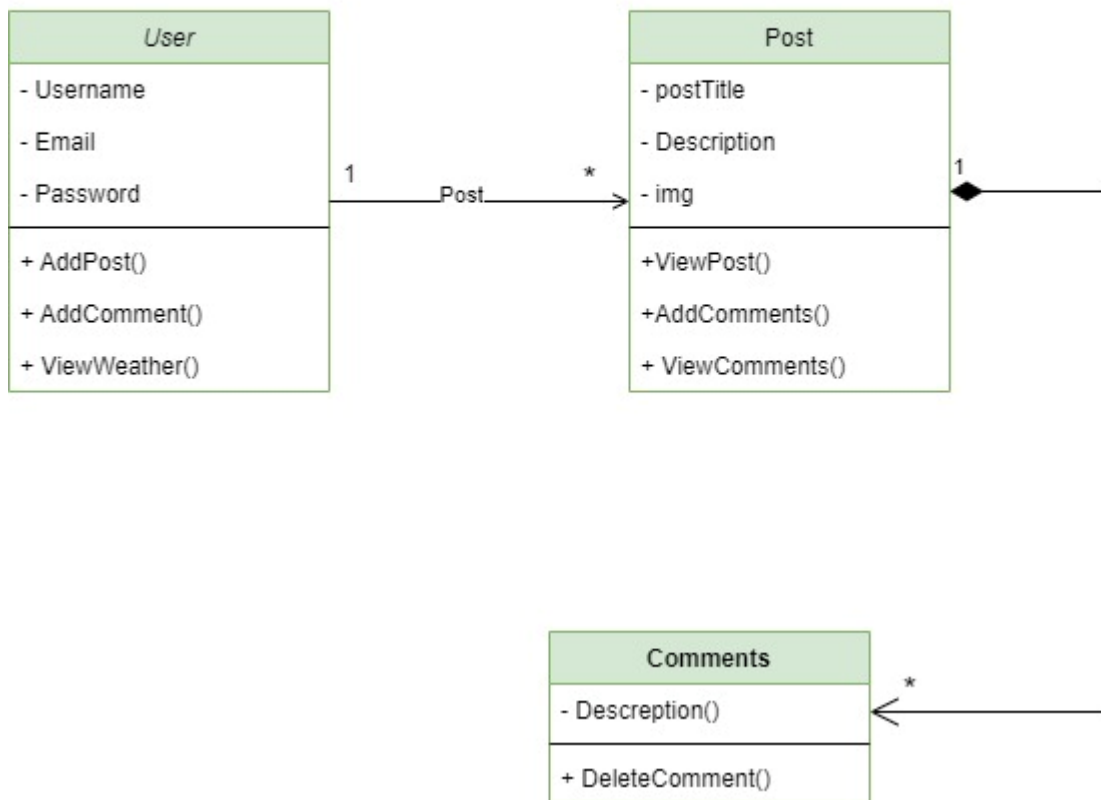


LEVEL -1



3. Class Diagram:

Class Diagram



Data Dictionary:

We have used MongoDB database in our project so we cannot create data dictionary but we are providing screenshot of our database.

UserSchema:

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  username: {
    type: String,
    required: true,
    trim: true,
```

```

    maxlength: 30,
    unique: true
  },
  email: {
    type: String,
    required: true,
    trim: true,
    unique: true,
  },
  password: {
    type: String,
    required: true,
  },
  avatar: {
    type: String,
    default:
'https://i.pinimg.com/474x/65/25/a0/6525a08f1df98a2e3a545fe2ace4b
e47.jpg',
  },
  role: {
    type: String,
    default: 'user',
  },
  following:{
    type:Array,
  },
  follower:{
    type:Array,
  }
}, {
  timestamps: true
})

module.exports = mongoose.model('user', userSchema);

```

DiscussionSchema:

```

const mongoose = require('mongoose');

const discussSchema = new mongoose.Schema({
  title: {
    type: String,

```

```

    required: true,
    trim: true,
  },
  description: {
    type: String,
    required: true,
    trim: true,
  },
  images:{
    type:Array,
    require:true
  },
  user:{
    type: mongoose.Types.ObjectId,
    ref: 'user'
  }
}, {
  timestamps: true
})

module.exports = mongoose.model('discuss', discussSchema);

```

CommentSchema:

```

const mongoose = require('mongoose');

const commentSchema = new mongoose.Schema({
  cmnt: {
    type: String,
    required: true
  },
  user: {
    type: mongoose.Types.ObjectId, ref: 'user'
  },
  postId: {
    type: mongoose.Types.ObjectId, ref: 'discuss'
  },
  postUserId: {
    type: mongoose.Types.ObjectId, ref: 'user'
  }
}, {
  timestamps: true
});

```

```
module.exports = mongoose.model('comment', commentSchema);
```

Implementation Details:

1. Modules:

User module:

User module saves all the information during registration of user if user details are valid. During login , credentials of the user would be verify and if valid would be successfully logged in.

Post Module:

User would be able to add, delete view post of their own. At time of post user needs to provide title, description and image.

Comment Module:

User can add comment to any post and also can delete his/her own comment.

Weather module:

User can view weather of upcoming days.

2. Major Functionality:

1. Register User:

```
register: async (req, res) => {  
  try {
```

```
const { username, email, password } = req.body;
let newUserUsername = username.toLowerCase().replace(/ /g, "");

if(!username || !email || !password)
  return res.status(500).json({ msg: "Please fill all fields!!" });

const user_name = await Users.findOne({ username:
newUserName });
if (user_name)
  return res.status(500).json({ msg: "This Username is already in
Use." });

const user_email = await Users.findOne({ email });
if (user_email)
  return res.status(500).json({ msg: "This Email is already in
Use." });

if(!validateEmail(email))
  return res.status(500).json({ msg: "Invalid emails!!" });

if (password.length < 6)
  return res.status(500).json({ msg: "Password must be at least 6
characters." });

const passwordHash = await bcrypt.hash(password, 12);

const newUser = new Users({
  username: newUserUsername, email, password: passwordHash
});

const access_token = createAccessToken({ id: newUser._id });
const refresh_token = createRefreshToken({ id: newUser._id });

res.cookie('refreshtoken', refresh_token, {
  httpOnly: true,
  path: '/api/refresh_token',
  maxAge: 30 * 7 * 24 * 60 * 60 * 1000 // 30days
});

await newUser.save();

res.json({
  msg: 'Registered Successfully :)',
```

```

        access_token,
        user: {
            ...newUser._doc,
            password: "
        }
    });
} catch (err) {
    return res.status(500).json({msg: err.message});
}
},

```

2. Login:

```

login: async (req, res) => {
    try {
        const { email, password } = req.body;

        const user = await Users.findOne({ email });

        if (!user)
            return res.status(500).json({ msg: "This email does not exist."
});

        const isMatch = await bcrypt.compare(password, user.password);

        if (!isMatch)
            return res.status(500).json({ msg: "Password is Wrong." });

        const access_token = createAccessToken({ id: user._id });
        const refresh_token = createRefreshToken({ id: user._id });

        res.cookie('refreshtoken', refresh_token, {
            httpOnly: true,
            path: '/api/refresh_token',
            maxAge: 30 * 7 * 24 * 60 * 60 * 1000 // 30days
        });

        res.json({
            msg: 'Login Successfully :)',
            access_token,
            user: {
                ...user._doc,
                password: "
            }
        });
    } catch (err) {
        return res.status(500).json({msg: err.message});
    }
}

```



```

    }
  });
} catch (err) {
  return res.status(500).json({msg: err.message});
}

```

3. Add Post:

```

try {
  const { title, description, picture, id } = req.body;
  const user = await Users.findById(id);
  const Detail = new discussSchema({
    title: title,
    description: description,
    images: picture,
    user,
  });

  await Detail.save();

  res.json({
    msg: "Details added Successfully :",
    Detail,
  });
} catch (err) {
  return res.status(500).json({ msg: err.message });
}
},

```

4. Delete Post:

```

deletePost: async (req, res) => {
  try {
    const post = await discussSchema.findByIdAndDelete({ _id:
req.body.id });
    const cmnt = await Comments.find({ postId: req.body.id
}).deleteMany({
    postId: req.body.id,
  });
  }
}

```

```

    res.json({
      msg: "Post Removed",
    });
  } catch (err) {
    return res.status(500).json({ msg: err.message });
  }
},

```

5. Add Comment:

```

addComment: async (req, res) => {
  try {
    const { cmnt, id, postId, postUser } = req.body;

    const com = await discussSchema.findOne({ _id: postId });
    if (!com)
      return res.status(400).json({ msg: "This Post does not exist." });

    const newCmnt = new Comments({
      cmnt,
      postId,
      postUser,
      user: id,
    });

    await newCmnt.save();

    res.json({
      msg: "News Added XD",
      newCmnt,
    });
  } catch (err) {
    return res.status(500).json({ msg: "error" });
  }
},

```

6. Delete Comment:

```

deleteCmnt: async (req, res) => {
  try {
    const cmnt = await Comments.findOneAndDelete({ _id: req.body.id
  });

```

```

res.json({
  msg: "Cmnt Removed",
});
} catch (err) {
return res.status(500).json({ msg: err.message });
}

```

7. Follow-Unfollow User:

```

follow: async (req, res) => {
  const { id,currentUser } = req.body;
  await Users.updateOne(
    { _id: currentUser },
    { $push: { following: id } });
  await Users.updateOne(
    { _id: id },
    { $push: { follower: currentUser } });
  res.json({
    msg:"user started following",
  });
},
following: async (req, res) => {
  const { currentUser } = req.body;
  const user = await Users.findOne({ _id:currentUser })
  const retu=user.following
  res.json({
    retu,
  });
},
unfollow: async (req, res) => {
  const { id,currentUser } = req.body;
  // const user = await Users.findOne(currentUser);
  await Users.updateOne(
    { _id: currentUser },
    { $pull: { following: id } });
  await Users.updateOne(
    { _id: id },
    { $pull: { follower: currentUser } });
  res.json({
    msg:"user has unfollowed",
  });
}

```

```
});  
}
```

8. Search Post

```
search: async (req, res) => {  
  try {  
    const posts = await discussSchema.find({  
      title: { $regex: req.query.posts },  
    });  
    res.json({ posts });  
  } catch (err) {  
    return res.status(500).json({ msg: err.message });  
  }  
},
```

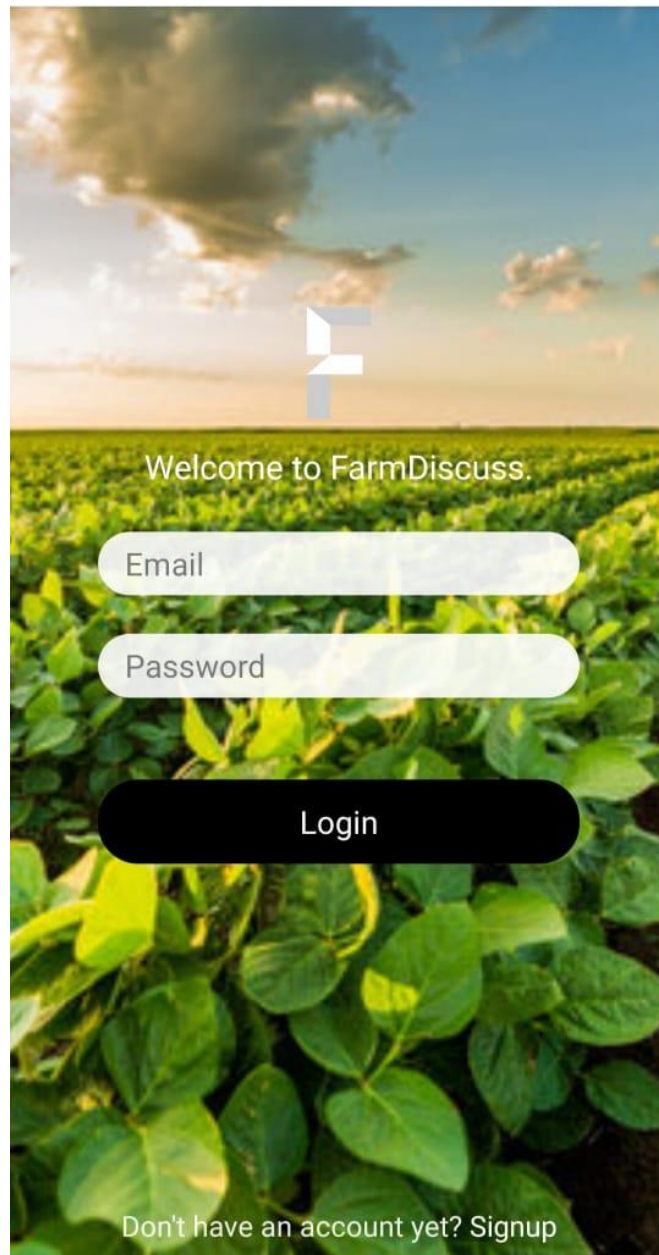
Testcase:

Test Case Id	SRS Id	Test Case Objective	Input Data	Expected Output	Actual Output
TC_01	R 1.2	User Login	User Id, Password	Success Message	Success Message
TC_02	R 1.2	User Login	Wrong User Email	User Not Found	User Not Found
TC_03	R 1.2	User Login	Wrong Password	Password is Wrong	Password is Wrong
TC_04	R 1.1	User Registration	Already user details	User already exist	User already exist
TC_05	R 1.1	User Registration	Valid user details	User Added	User Added

TC_o6	R 2.1	Add post	Add post data	Added Successfully	Post added
TC_o7	R 2.1	Delete post	Delete data	Post deleted	Post Deleted
TC_o8	R 3.1	Add Comment	Updated news	Comment Added	Comment Added
TC_o9	R 3.2	Delete comment	Delete	Comment deleted	Comment deleted

Screen Shots:

Login

The login page features a background image of a green field under a sunset sky. A stylized 'F' logo is centered in the upper half. Below the logo, the text 'Welcome to FarmDiscuss.' is displayed. The form includes two input fields for 'Email' and 'Password', followed by a 'Login' button. At the bottom, there is a link for users who do not have an account yet.

Welcome to FarmDiscuss.

Email

Password

Login

Don't have an account yet? [Signup](#)

Home page:

← Home

Farm Discuss



🔍 Search



FEEDS

POSTS

WEATHER



How to Make Rose Farm?

Site selected should receive good amount of sunshine as plenty of sunshine is required for the proper growth of t...



How Can Lemons Benefits Your Health?

The pulp, rind, and juice are rich with vitamins that stimulate immunity and reduce the risk of disease. Th...

← AddQues

Title

Description



Upload Image

Submit



ishita0202
ishita0202@gmail.com

7 steps for wheat Farming



- 1: Pick a dry area for cultivation: Wheat is both a spring and winter crop, and utilizes at least 8 hours of sun on a daily basis. Therefore, avoid picking a site that's too shady and doesn't allow enough sunlight.
- 2: Prepare the land well in advance: Growing wheat requires prepping the soil for strong root growth. Till your soil to a depth of at least 15 cm, and make sure that it's even.
- 3: Spread the wheat seeds in a semi-circular motion: You can always use a seed spreader too, as long as it spreads the seeds at approximately one seed per 2.5 sq. cm.
- 4: Use phosphorus, nitrogen & Potassium



← Details

3: Spread the wheat seeds in a semi-circular motion: You can always use a seed spreader too, as long as it spreads the seeds at approximately one seed per 2.5 sq. cm.

4: Use phosphorus, nitrogen & Potassium fertilization: Phosphorus and nitrogen collectively help develop strong roots which will help the wheat crop survive winter. Optimum dose of Mahadhan 12:32:16 (75 Kg/ac) can help wheat

✕

Comment

Put your thoughts

submit

to develop. Then on, follow a strict irrigation schedule.



ishita0202

Wow thanks



charvit

Great 🍷🍷🍷



← UserProfile



ishita0202

ishita0202@gmail.com

Following

2

Follower

1



7 steps for wheat Farming

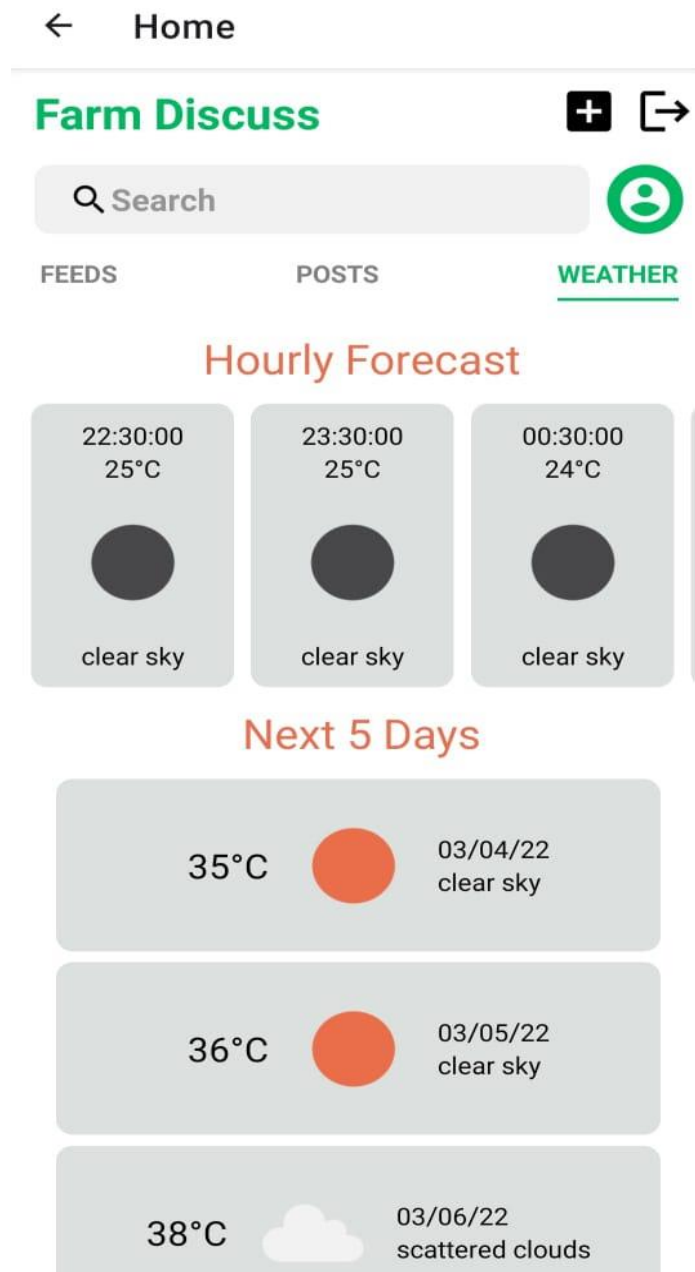
1: Pick a dry area for cultivation: Wheat is both a spring and winter crop, and utilizes at least 8 hours of sun o...



How to Make Rose Farm?

Site selected should receive good amount of sunshine as plenty of sunshine is required for the proper growt...





Conclusion:

Hereby, we can conclude that we have successfully implemented user module, post module, comment module and weather module.

Users can create account and post their queries. They can answer to others queries. Users can search their queries. They can follow other users to view their latest post. Users can check weather for upcoming days.

Limitations and Future extension:

Limitations:

1. There is no filtration for popular post in post section.
2. There is no category for post.

Future Extension:

1. Post will be displayed based on its popularity in post section.
2. Choose category before user post anything.
3. Mandi rates for market rates of fruits and vegetables.

Bibliography:

<https://reactnative.dev/docs/getting-started>

<https://stackoverflow.com/>

<https://docs.atlas.mongodb.com/getting-started/>

<https://www.youtube.com/>