**CHAPTER 1**

**INTRODUCTION**

* 1. **OVERVIEW**

The “Farm management system” has been develop to override the problems prevailing in the practicing manual system. This software is supported to eliminate and, in some cases, reduce the hardships faced by this existing system. Moreover, this system is designed for the particular need of the company to carry out operations in a smooth and effective manner.

The application is reduced and much as possible to avoid errors while entering the data. It also provides error message while entering invalid data. No formal knowledge is needed for user to use this system. Thus, by this all it proves it is user-friendly. Farm Management system. It can assist the user to concentrate on their other activities rather to concentrate on the record keeping. Thus, it will help organization in better utilization of resources.

Every organization, whether big or small, has challenges to overcome and managing the information of Crops, Farm, Insecticides, Equipment’s, Pesticides. Every Farm Management System has different Farm needs; therefore, we design exclusive employee management systems that are adapted to your managerial requirements. This is designed to assist in strategic planning, and will help you ensure that your organization is equipped with the right level of information and details for your future goals. Also, for those busy executive who are always on the go, our systems come with remote access features, which will allow you to manage your workforce anytime, at all times. These systems will ultimately allow you to better manage resources.

* 1. **OBJECTIVE**
* A lot farmers in India are getting the fertilizers and equipment's required at overpriced.
* Therefore our solution to above problem is to remove the middleman and provide utilities required by farmers at cheaper affordable price directly supplying from producers.
* The application developed will help farmers to buy fertilizers and equipment's on time without the involvement of middlemen directly through online. Also, people can know from where the utilities are coming.

**Functionalities provided by Farm Management System are as follows**:

* Provides the searching facilities based on various factors. Such as Fertilizers, Equipment’s, Insecticides, Pesticides
* Farm Management System also manage the Cost range details online for Insecticides details, Pesticides details
* It tracks all the information of Crops, Cost Range, Insecticides etc
* Manage the information of Fertilizers, Equipment’s
* Shows the information and description of the Fertilizers, Equipment’s
* Editing, adding and updating of Records is improved which results in proper resource management of Farm data.
  1. **PROPOSED SYSTEM**

It may help collecting perfect management in details. In a very short time, the collection will be obvious, simple and sensible. It will help a person to know the management of passed year perfectly and vividly. It also helps in current all works relative to Farm Management System. It will be also reduced the cost of collecting the management & collection procedure will go on smoothly.

Our project aims at Business process automation, i.e. we have tried to computerize various processes of Farm Management System.

* In computer system the person has to fill the various forms & number of copies of the forms can be easily generated at a time.
* In computer system, it is not necessary to create the manifest but we can directly print it which saves our time.
* To assist the staff in capturing the effort spent on their respective working areas.
* To utilize resources in an efficient manner by increasing their productivity through automation.
* The system generates types of information that can be used for various purposes.
* It satisfies the user requirement
* Be easy to understand by the user and operator
* Be easy to operate
* Have a good user interface
* Be expandable
* Delivered on schedule within the budget
  1. **APPLICATION**

Various applications of a database are – storage, retrieval, upgradation, alteration and deletion. An attendance management system can be used for daily student attendance in schools, collages and institutes. If facilitates to access the attendance information of a particular student in a particular class.

The attendance management systems are simple and easy to use for the teacher as well as the admin. The system provides online storage of data. The system is flexible and can be adapted to the needs of the institution. The web based attendance management system software can be easily accessed from anywhere and at any time. The teacher can search for student from classrooms, computer labs, home and places where the internet is available. The system provides access from smartphones and tablets to search books and resources. Digital system eliminate the need for extensive paperwork and too many staff. Maintenance overheads and operation costs are reduced.

**CHAPTER 2**

**HARDWARE AND SOFTWARE REQUIREMENTS**

**2.1 HARDWARE REQUIREMENTS**

Processor: Intel Pentium 4( 1.50 GHZ) or above

RAM: 1GB

Hard Disk: 128GB

**2.2 SOFTWARE REQUIREMENTS**

Operating System: Microsoft Windows 10

Language: HTML, CSS, JS

IDE: Visual Studio Code

Back End: Node(Express), MySQL

**2.3 INTRODUCTION TO LANGUAGE**

**Hyper Text Mark-up Language (HTML):**

**HTML** stands for Hyper Text Mark-up Language, which is the most widely used language on Web to develop web pages. It was created by Berners-Lee in 1991 but "HTML2.0" was the first standard HTML specification published in 1995. HTML 4.01 was a major version of HTML and it was published in 1999. Though HTML 4.01 version is widely used but currently we are having HTML-5 version which is an extension to HTML 4.01, and this version was published in 2012.

Further development under the auspices of the IETF was stalled by competing interests. Since1996, the HTML specifications have been maintained, with input from commercial software. Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers. Now, HTML is being widely used to format web pages with the help of different tags available in HTML language

**Cascading Style sheet (CSS):**

**CSS** is the language we use to style an HTML document. CSS is the acronym for Cascading Style Sheet . CSS describes how HTML elements should be displayed. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. CSS is among the core languages of the open web and is standardized across Web browsers. Development of various parts of CSS specification was done synchronously, which allowed versioning of the latest recommendations like CSS1, CSS2, and CSS3, etc.

CSS is a MUST for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain. The Advantages of CSS include:

* Create Stunning Websites
* Control web
* Become efficient web designer, etc.

**JavaScript**:

JAVASCRIPT is a scripting or programming language that allows you to implement complex features on web pages – every tie a web page does more than just sit there and display static information for you to look at – displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc – you can bet that JavaScript is probably involved. It is the third layer of the layer cake of standard web technologies, two of which (HTML and CSS) we have covered in much more detail in other parts of the Learning Area.

The core client-side JavaScript language consists of some common programming features that allow you to do things like:

* Store useful values inside variables.
* Operations on pieces of text (known as "strings" in programming)
* Running code in response to certain events occurring on a web page.
* And much more!

**2.4 INTRODUCTION TO IDE**

**Visual Studio Code (VS Code):**

**Visual Studio Code** is a code editor redefined and optimized for building and debugging modern web and cloud applications. Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging.

First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit- build-debug cycle means less time fiddling with your environment, and more time executing on your ideas. It supports macOS, Linux, and Windows - so you can hit the ground running, no matter the platform.

At its heart, Visual Studio Code features a lightning fast source code editor, perfect for day-to- day use. With support for hundreds of languages, VS Code helps you be instantly productive with syntax highlighting, bracket-matching, auto-indentation, box-selection, snippets, and more. Intuitive keyboard shortcuts, easy customization and community-contributed keyboard shortcut mappings let you navigate your code with ease. For serious coding, you'll often benefit from tools with more code understanding than just blocks of text.

Visual Studio supports different programming languages by means of language services, which allow the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Visual Studio also includes a web-site editor and designer that allows web pages to be authored by dragging and dropping widgets. It is used for developing VB.NET application efficiently to get input and output design easiest one. It will be run at windows application based services provide the user.

**2.5 INTRODUCTION TO BACKEND**

**Node**

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

const http = require('http');

const hostname = '127.0.0.1';

const port = 3000;

const server = http.createServer((req, res) => {

res.statusCode = 200;

res.setHeader('Content-Type', 'text/plain');

res.end('Hello World');

});

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

This is in contrast to today's more common concurrency model, in which OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. Almost no function in Node.js directly performs I/O, so the process never blocks except when the I/O is performed using synchronous methods of Node.js standard library. Because nothing blocks, scalable systems are very reasonable to develop in Node.js.

Node.js is similar in design to, and influenced by, systems like Ruby's [Event Machine](https://github.com/eventmachine/eventmachine)  and Python's [Twisted](https://twistedmatrix.com/trac/). Node.js takes the event model a bit further. It presents an [event loop](https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/) as a runtime construct instead of as a library. In other systems, there is always a blocking call to start the event-loop. Typically, behavior is defined through callbacks at the beginning of a script, and at the end a server is started through a blocking call like EventMachine::run(). In Node.js, there is no such start-the-event-loop call. Node.js simply enters the event loop after executing the input script. Node.js exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user.

HTTP is a first-class citizen in Node.js, designed with streaming and low latency in mind. This makes Node.js well suited for the foundation of a web library or framework.

Node.js being designed without threads doesn't mean you can't take advantage of multiple cores in your environment. Child processes can be spawned by using our [child\_process.fork()](https://nodejs.org/api/child_process.html#child_process_child_process_fork_modulepath_args_options) API, and are designed to be easy to communicate with. Built upon that same interface is the [cluster](https://nodejs.org/api/cluster.html) module, which allows you to share sockets between processes to enable load balancing over your cores.

**MySQl**:

**MySQL** is one of the most recognizable technologies in the modern big data ecosystem. Often called the most popular database and currently enjoying widespread, effective use regardless of industry, it’s clear that anyone involved with enterprise data or general IT should at least aim for a basic familiarity of MySQL.

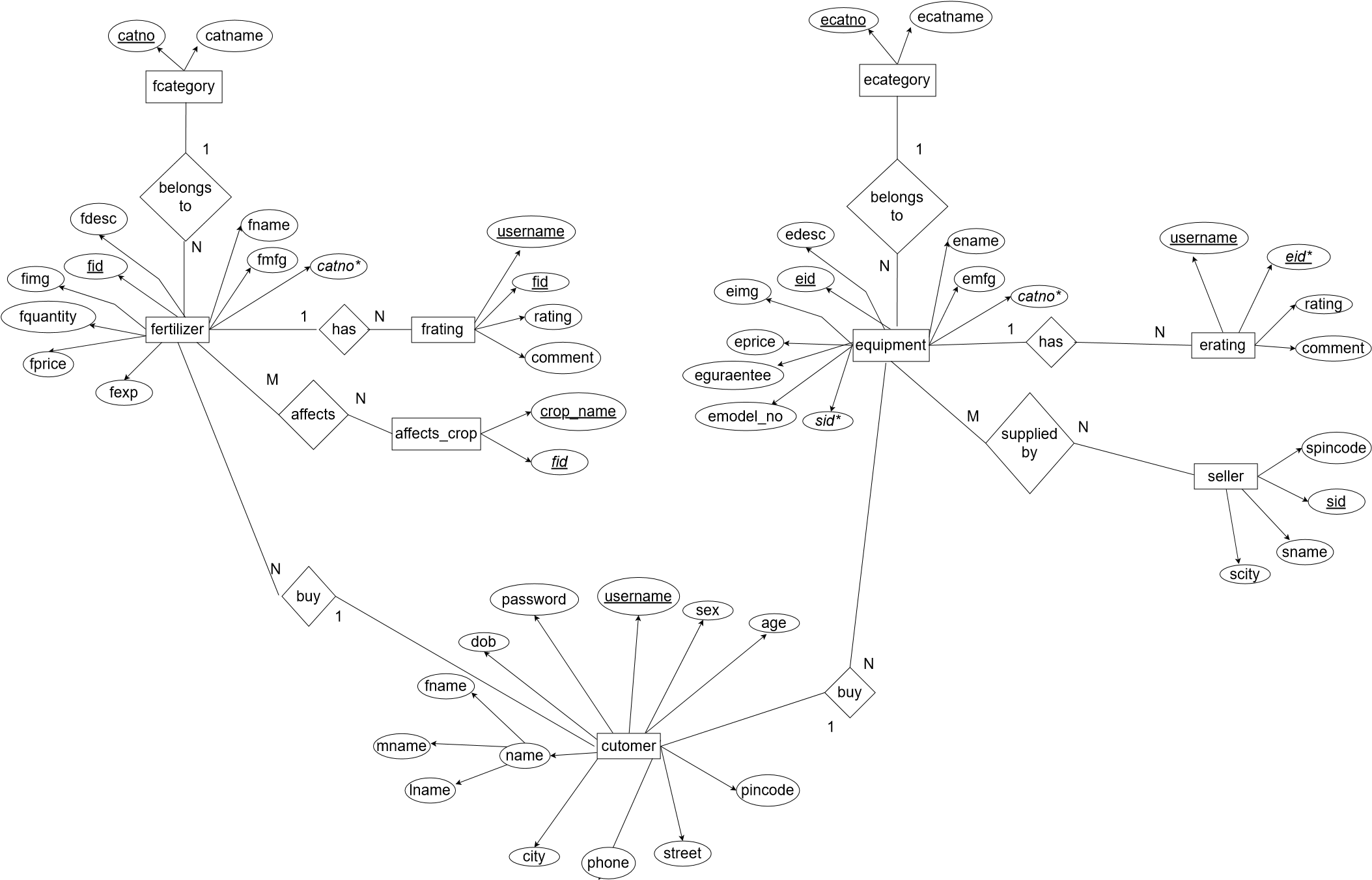
It is a relational database management system based on the Structured Query Language, which is the popular language for accessing and managing the records in the database [9]. MySQL is open-source and free software under the GNU license. It is supported by Oracle Company. It is ideal for both small and large applications. Xampp is one of the local hosts which supports SQL.

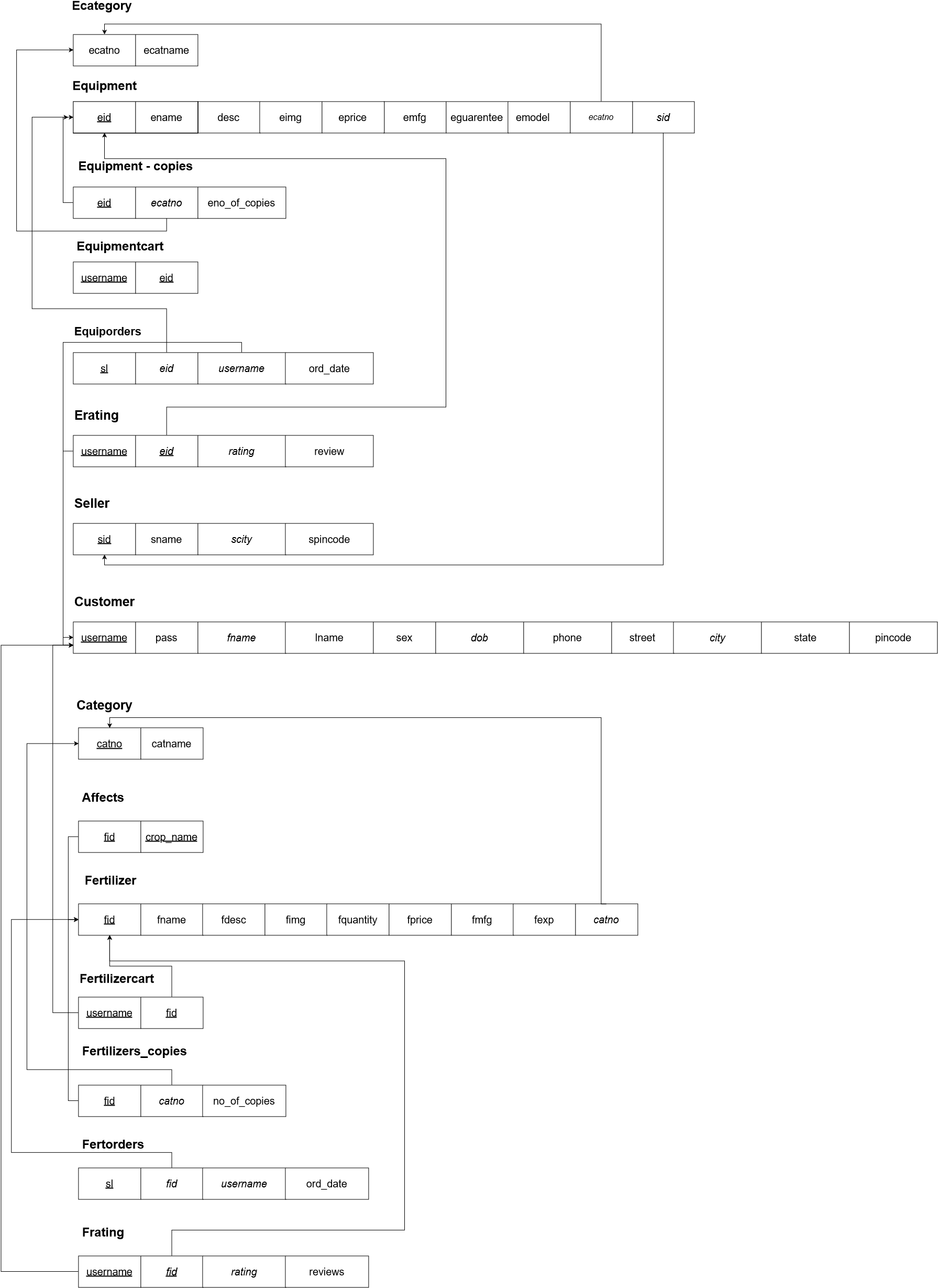
MySQL is integral to many of the most popular software stacks for building and maintaining everything from customer-facing web applications to powerful data-driven B2B services. Its open-source nature, stability, and rich feature set, paired with ongoing development and support from Oracle, have meant that internet-critical organizations such as Facebook, Flickr, Twitter, Wikipedia, and YouTube all employ MySQL backends

**CHAPTER 3**

**DESIGN AND IMPLEMENTATION**

**3.1 ER DIAGRAM**



 **3.2 SCHEMA DIAGRAM**

**3.3 PSUEDO CODE**

**BACKEND:**

*const* express = require('express')

*const* mysql = require('mysql')

*const* app = express()

*const* bodyParser = require('body-parser')

*const* path = require('path')

*const* ejsMate = require('ejs-mate')

*const* methodOverride = require('method-override')

*const* ExpressError = require('./utils/ExpressError')

*const* bcrypt = require('bcrypt')

*const* session = require('express-session')

*const* flash = require('connect-flash')

*const* { nextTick } = require('process')

app.use(bodyParser.urlencoded({ extended: true }))

app.set('view engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'))

app.engine('ejs', ejsMate)

app.use(methodOverride('\_method'))

*const* sessionConfig = {

  secret: 'thisismysecret',

  resave: false,

  saveUninitialized: true,

  cookie: {

    httpOnly: true,

    expires: Date.now() + 10000 \* 60 \* 60 \* 24 \* 7,

    maxAge: 10000 \* 60 \* 60 \* 24 \* 7

  }

}

app.use(session(sessionConfig))

app.use(flash())

app.use((*req*, *res*, *next*) => {

  res.locals.username = req.session.username;

  res.locals.success = req.flash('success');

  res.locals.error = req.flash('error');

  next()

})

*const* requireLogin = (*req*, *res*, *next*) => {

  if (!req.session.username) {

    return res.redirect('/login')

  } else {

    next()

  }

}

*// console.log(path.join(\_\_dirname, '/../public'))*

*const* pathForPublic = path.join(\_\_dirname, '/../public');

app.use(express.static(pathForPublic))

*//create connection*

*const* db = mysql.createConnection({

  host: 'localhost',

  user: 'root',

  password: '',

  database: 'agriculture',

  multipleStatements: true

});

*//connecting to DB*

db.connect((*err*) => {

  if (err) throw err;

  console.log('Database connected');

});

*//landing page route*

app.get('/', (*req*, *res*, *next*) => {

  try {

*let* fertCount = 0;

*const* query1 = `SELECT COUNT(fid) FROM fertilizer`

    db.query(query1, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      fertCount = result[0]['COUNT(fid)']

    })

*let* equipCount = 0;

*const* query2 = `SELECT COUNT(eid) FROM equipment`

    db.query(query2, (*err1*, *result1*) => {

      if (err1) throw new ExpressError(err1.message, err1.statusCode);

      equipCount = result1[0]['COUNT(eid)']

    })

    setTimeout(() => { res.render('home/homepage', { fertCount, equipCount }) }, 100)

  } catch (e) {

    next(e)

  }

*// res.send([fertCount, equipCount])*

})

*//login page route*

app.get('/login', (*req*, *res*) => {

  res.render('Logins/login')

})

app.post('/login', *async* (*req*, *res*, *next*) => {

  try {

*const* { username, pass } = req.body;

*const* sql = `SELECT pass, fname FROM customer WHERE username = ?`

    db.query(sql, [username], *async* (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      if (result[0]) {

*const* bcryptResult = await bcrypt.compare(pass, result[0].pass)

*const* fname = result[0].fname;

        if (bcryptResult) {

          req.session.username = username;

          req.flash('success', `Welcome back! ${fname}`)

*// console.log(req.session.fname),*

          res.redirect('/')

        } else {

          req.flash('error', 'Try again!')

          res.redirect('/login')

        }

      } else {

        req.flash('error', "You dont't have an account create a account first")

        res.redirect('/login')

      }

    })

  } catch (e) {

    next(e)

  }

})

*//logout*

app.post('/logout', (*req*, *res*, *next*) => {

  req.session.username = null;

  req.flash('success', 'logged out successfull')

  res.redirect('/')

})

*// register page route*

app.get('/register', (*req*, *res*) => {

  res.render('Logins/register')

})

app.post('/register', *async* (*req*, *res*, *next*) => {

  try {

*const* { username, pass, fname, lname, sex, dob, phone, street, city, state, pincode } = req.body;

*const* hashedPw = await bcrypt.hash(pass, 12);

*const* sql = `INSERT into customer SET ?`

*const* value = {

      username,

      pass: hashedPw,

      fname,

      lname,

      sex,

      dob,

      phone,

      street,

      city,

      state,

      pincode

    }

    db.query(sql, value, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode)

      req.flash('success', 'Welcome back!')

      req.session.username = username;

      res.redirect('/fertilizers')

    })

  } catch (e) {

    next(e)

  }

})

app.get('/updateProfile', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* username = req.session.username;

*const* sql = `SELECT \* FROM customer WHERE username = ?`

    db.query(sql, [username], (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode)

      res.render('Logins/user', { user: result[0] })

*// res.send([result[0]])*

    })

  } catch (e) {

    next(e)

  }

})

app.put('/updateProfile', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* username = req.session.username;

*const* { fname, lname, sex, phone, street, city, pincode, state } = req.body

*const* sql = `UPDATE customer SET ? WHERE username = ?`

*const* value = {

      fname,

      lname,

      sex,

      phone,

      street,

      city,

      state,

      pincode

    }

    db.query(sql, [value, username], (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode)

      req.flash('success', 'Changes saved!')

      res.redirect('/updateProfile')

    })

  } catch (e) {

    next(e)

  }

})

*// FERTILIZER MODULE:*

*// All fertilizers(get) => /fertilizers*

app.get('/fertilizers', (*req*, *res*, *next*) => {

  try {

*const* sql = 'SELECT \* FROM fertilizer f, category c WHERE c.catno = f.catno'

    db.query(sql, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

*// console.log(result)*

      res.render('Fertilizers/allFertilizers', { fertilizers: result })

    })

  } catch (e) {

    next(e)

  }

})

*// Add(post) => /fertilizer/new*

app.get('/fertilizer/new', requireLogin, (*req*, *res*) => {

  res.render('Fertilizers/fertilizerNew')

})

app.post('/fertilizer/new', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid, fname, fprice, fquantity, fmfg, fdesc, fexp, fimg, catno, crop\_name, no\_of\_copies } = req.body

*const* fvalue = {

      'fid': fid,

      'fname': fname,

      'fdesc': fdesc,

      'fprice': fprice,

      'fquantity': fquantity,

      'fmfg': fmfg,

      'fexp': fexp,

      'fimg': fimg,

      'catno': catno

    }

*const* crvalue = {

      'crop\_name': crop\_name,

      'fid': fid

    }

*const* fcvalue = {

      'fid': fid,

      'catno': catno,

      'no\_of\_copies': no\_of\_copies

    }

*let* fSql = "INSERT INTO fertilizer SET ? "

    db.query(fSql, fvalue, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

    })

*let* crSql = "INSERT INTO affects SET ? "

    db.query(crSql, crvalue, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

    })

*let* fcSql = "INSERT INTO fertilizers\_copies SET ? "

    db.query(fcSql, fcvalue, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

    })

    req.flash('success', 'New fertilizer added!')

    res.redirect(`/fertilizer/${fid}`)

  } catch (e) {

    next(e)

  }

})

*// Edit(put) => /fertilizer/fid*

app.get('/fertilizer/edit/:fid', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid } = req.params

*let* sql = `SELECT \* FROM affects WHERE fid = ${fid};

    SELECT \* FROM fertilizer f, category c WHERE f.catno = c.catno AND f.fid = ${fid};

    SELECT \* FROM fertilizers\_copies WHERE fid = ${fid};`

    db.query(sql, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

*// console.log(result)*

*const* cropNames = getCropsNames(result[0])

*const* no\_of\_copies = getNoOfCopies(result[2][0])

*const* fertilizerDetails = getFertilizerDetails(result[1][0])

      res.render('Fertilizers/fertilizerEdit', { cropNames, no\_of\_copies, fertilizerDetails })

    })

  } catch (e) {

    next(e)

  }

})

*const* getCropsNames = (*items*) => {

*let* arr1 = []

  for (*let* item of items) {

    arr1.push(item.crop\_name)

  }

  return arr1;

}

*const* getNoOfCopies = (*item*) => {

  return item.no\_of\_copies

}

*const* getFertilizerDetails = (*item*) => {

*let* obj = {};

  obj['fid'] = item.fid

  obj['fname'] = item.fname

  obj['fdesc'] = item.fdesc

  obj['fprice'] = item.fprice

  obj['fquantity'] = item.fquantity

  obj['fmfg'] = item.fmfg

  obj['fexp'] = item.fexp

  obj['catno'] = item.catno

  obj['catname'] = item.catname

  obj['fimg'] = item.fimg

  return obj;

}

app.put('/fertilizer/:fid', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid, fname, fdesc, fprice, fquantity, fimg, catno, no\_of\_copies } = req.body

*const* fvalue = {

      fname,

      fdesc,

      fimg,

      fquantity,

      fprice,

      catno

    }

*const* fcvalue = {

      catno,

      no\_of\_copies

    }

*let* sql = `UPDATE fertilizer SET ? WHERE fid = ${fid}`

    db.query(sql, fvalue, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

    })

*let* sql1 = `UPDATE fertilizers\_copies SET ? WHERE fid = ${fid}`

    db.query(sql1, fcvalue, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      req.flash('success', 'Update saved')

      res.redirect(`/fertilizer/${fid}`)

    })

  } catch (e) {

    next(e)

  }

})

*// show(get) => /fertilizer/fid*

app.get('/fertilizer/:fid', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid } = req.params

*const* query = `SELECT reviews, username, rating FROM frating WHERE fid = ${fid};`

*let* reviews = []

    db.query(query, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      reviews = result

    })

*let* sql = `select \* from fertilizer f, category c where c.catno = f.catno and f.fid = ${fid};

    SELECT no\_of\_copies FROM fertilizers\_copies WHERE fid = ${fid};

    SELECT \* FROM affects WHERE fid = ${fid};

    SELECT AVG(rating) AS rating FROM frating WHERE fid = ${fid}`

    db.query(sql, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      if (reviews) {

        res.render('Fertilizers/fertilizerShow', { fertilizer: result[0][0], fertCount: result[1][0], crops: result[2], reviews, rating: result[3][0].rating })

      } else {

        res.render('Fertilizers/fertilizerShow', { fertilizer: result[0][0], fertCount: result[1][0], crops: result[2], rating: result[3][0].rating })

      }

    });

  } catch (e) {

    next(e)

  }

})

*//ADD review*

app.post('/fertilizer/:fid/review', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid } = req.params

*const* { rating, reviews } = req.body;

*const* value = {

      username: req.session.username,

      fid,

      rating,

      reviews

    }

*let* sql = `INSERT INTO frating SET ?`

    db.query(sql, value, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      req.flash('success', 'review added Successfully!')

      res.redirect(`/fertilizer/${fid}`);

    })

  } catch (e) {

    next(e)

  }

})

*// DELETE review*

app.delete('/fertilizer/:fid/reviews/:username', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid } = req.params

*// console.log(username, fid)*

*let* sql = `DELETE from frating where username = ? and fid = ?;`

    db.query(sql, [req.session.username, fid], (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode)

      req.flash('success', 'review deleted Successfully!')

      res.redirect(`/fertilizer/${fid}`)

    })

*// res.send('delete')*

  } catch (e) {

    next(e)

  }

})

*// Delete(delete) => /fertilizer/fid*

app.delete('/fertilizer/:fid', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid } = req.params

*let* sql = `delete from fertilizer where fid = ${fid};`;

    db.query(sql, (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      req.flash('success', 'Fertilizer deleted!')

      res.redirect('/fertilizers')

    })

  } catch (e) {

    next(e)

  }

})

*// cart remove item*

app.delete('/ferilizer/removeItem/:fid', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* { fid } = req.params;

*const* username = req.session.username

*const* sql = `DELETE FROM fertilizercart WHERE username = ? AND fid = ?`

    db.query(sql, [username, fid], (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      res.redirect('/cartItems')

    })

  } catch (e) {

    next(e)

  }

})

*//invoice route*

app.get('/getInvoice', requireLogin, (*req*, *res*, *next*) => {

  try {

*const* username = req.session.username;

*let* userInfo = ''

*const* sql = `SELECT \* FROM customer WHERE username = ?`

    db.query(sql, [username], (*err*, *result*) => {

      if (err) throw new ExpressError(err.message, err.statusCode);

      userInfo = result[0]

    })

*let* totalPrice = 0;

*const* sql1 = `SELECT \* FROM fertilizer f, fertilizercart fo WHERE f.fid = fo.fid AND fo.username = ?`

*let* fertOrders = ''

    db.query(sql1, [username], (*err1*, *result1*) => {

      if (err1) throw new ExpressError(err1.message, err1.statusCode);

      for (*let* i of result1) {

        totalPrice += (i.fprice + 35);

      }

      fertOrders = result1;

    })

app.use('\*', (*req*, *res*, *next*) => {

  next(new ExpressError('Page Not Found', 404))

});

app.use((*err*, *req*, *res*, *next*) => {

*const* { statusCode = 500 } = err

  if (!err.message) err.message = 'Oh no, Something went wrong!'

  res.status(statusCode).render('error', { err })

});

app.listen(5000, (*req*, *res*) => {

  console.log('LISTENING TO PORT NUMBER 5000')

});

**DATABASE:**

|  |
| --- |
| **Affects**  CREATE TABLE `affects` (   `crop\_name` varchar(30) NOT NULL,   `fid` int(11) NOT NULL,   PRIMARY KEY (`crop\_name`,`fid`),   KEY `fid` (`fid`),   CONSTRAINT `affects\_ibfk\_1` FOREIGN KEY (`fid`) REFERENCES `fertilizer` (`fid`) ON DELETE CASCADE) |
|  |

**Category**:

|  |
| --- |
| CREATE TABLE `category` (   `catno` int(11) NOT NULL,   `catname` varchar(30) DEFAULT NULL,   PRIMARY KEY (`catno`)) |

**Fertilizer\_copies:**

CREATE TABLE `fertilizers\_copies` (  
  `fid` int(11) NOT NULL,  
  `catno` int(11) DEFAULT NULL,  
  `no\_of\_copies` int(11) DEFAULT NULL,  
  PRIMARY KEY (`fid`),  
  KEY `catno` (`catno`),  
  CONSTRAINT `fertilizers\_copies\_ibfk\_1` FOREIGN KEY (`catno`) REFERENCES `category` (`catno`) ON DELETE CASCADE,  
  CONSTRAINT `fertilizers\_copies\_ibfk\_2` FOREIGN KEY (`fid`) REFERENCES `fertilizer` (`fid`) ON DELETE CASCADE)

|  |
| --- |
| **Frating**:  CREATE TABLE `frating` (   `username` varchar(50) NOT NULL,   `fid` int(11) NOT NULL,   `rating` int(11) DEFAULT NULL,   `reviews` varchar(1000) DEFAULT NULL,   PRIMARY KEY (`username`,`fid`),   KEY `fid` (`fid`),   CONSTRAINT `frating\_ibfk\_1` FOREIGN KEY (`username`) REFERENCES `customer` (`username`) ON DELETE CASCADE,   CONSTRAINT `frating\_ibfk\_2` FOREIGN KEY (`fid`) REFERENCES `fertilizer` (`fid`) ON DELETE CASCADE ) |

**Fertilizercart:**

CREATE TABLE `fertilizercart` (  
  `username` varchar(30) NOT NULL,  
  `fid` int(11) NOT NULL,  
  PRIMARY KEY (`username`,`fid`),  
  KEY `fid` (`fid`),  
  CONSTRAINT `fertilizercart\_ibfk\_1` FOREIGN KEY (`username`) REFERENCES  `customer` (`username`)

ON DELETE CASCADE,  
  CONSTRAINT `fertilizercart\_ibfk\_2` FOREIGN KEY (`fid`) REFERENCES `fertilizer` (`fid`) ON DELETE CASCADE)

**Fertorders:**

|  |
| --- |
| CREATE TABLE `fertorders` (   `sl` int(11) NOT NULL AUTO\_INCREMENT,   `fid` int(11) DEFAULT NULL,   `username` varchar(30) DEFAULT NULL,   `ord\_date` varchar(50) DEFAULT NULL,   PRIMARY KEY (`sl`),   KEY `fid` (`fid`),   KEY `username` (`username`),   CONSTRAINT `fertorders\_ibfk\_1` FOREIGN KEY (`fid`) REFERENCES `fertilizer` (`fid`) ON DELETE CASCADE,   CONSTRAINT `fertorders\_ibfk\_2` FOREIGN KEY (`username`) REFERENCES `customer` (`username`) ON DELETE CASCADE) |

**Customer:**

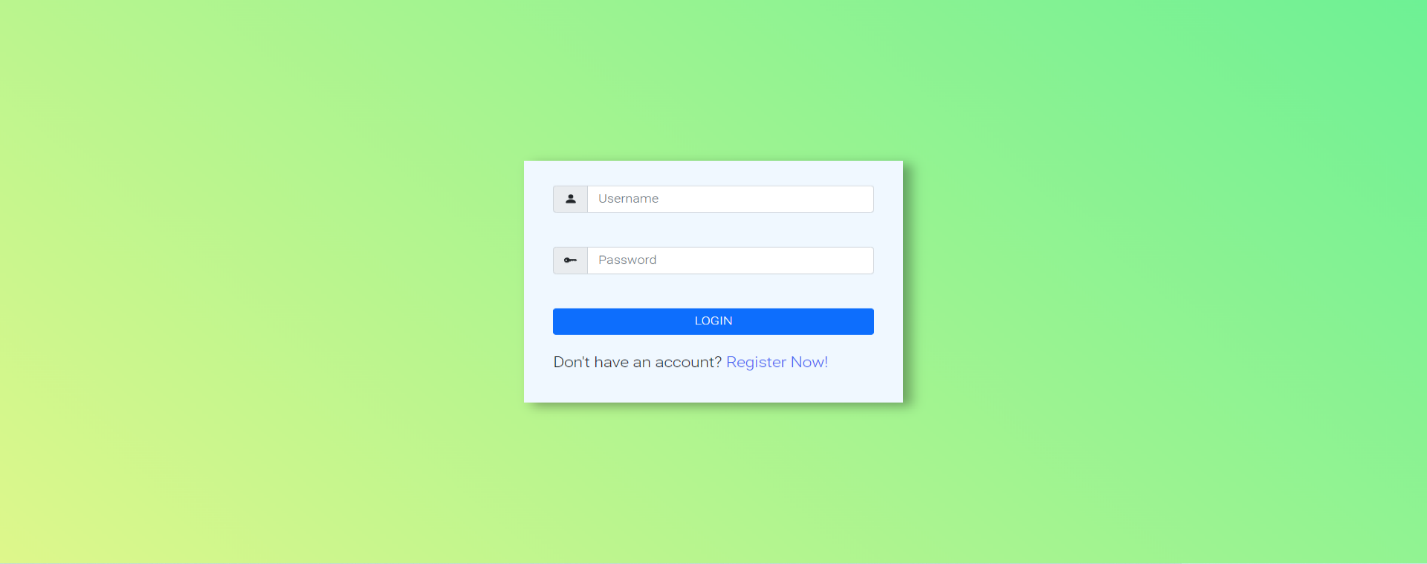
**C**REATE TABLE `customer` (  
  `username` varchar(30) NOT NULL,  
  `pass` varchar(255) DEFAULT NULL,  
  `fname` varchar(30) DEFAULT NULL,  
  `lname` varchar(30) DEFAULT NULL,  
  `sex` varchar(10) DEFAULT NULL,  
  `dob` date DEFAULT NULL,  
  `phone` bigint(20) DEFAULT NULL,  
  `street` varchar(30) DEFAULT NULL,  
  `city` varchar(30) DEFAULT NULL,  
  `state` varchar(30) DEFAULT NULL,  
  `pincode` int(11) DEFAULT NULL,  
  PRIMARY KEY (`username`)  
)

**CHAPTER 4**

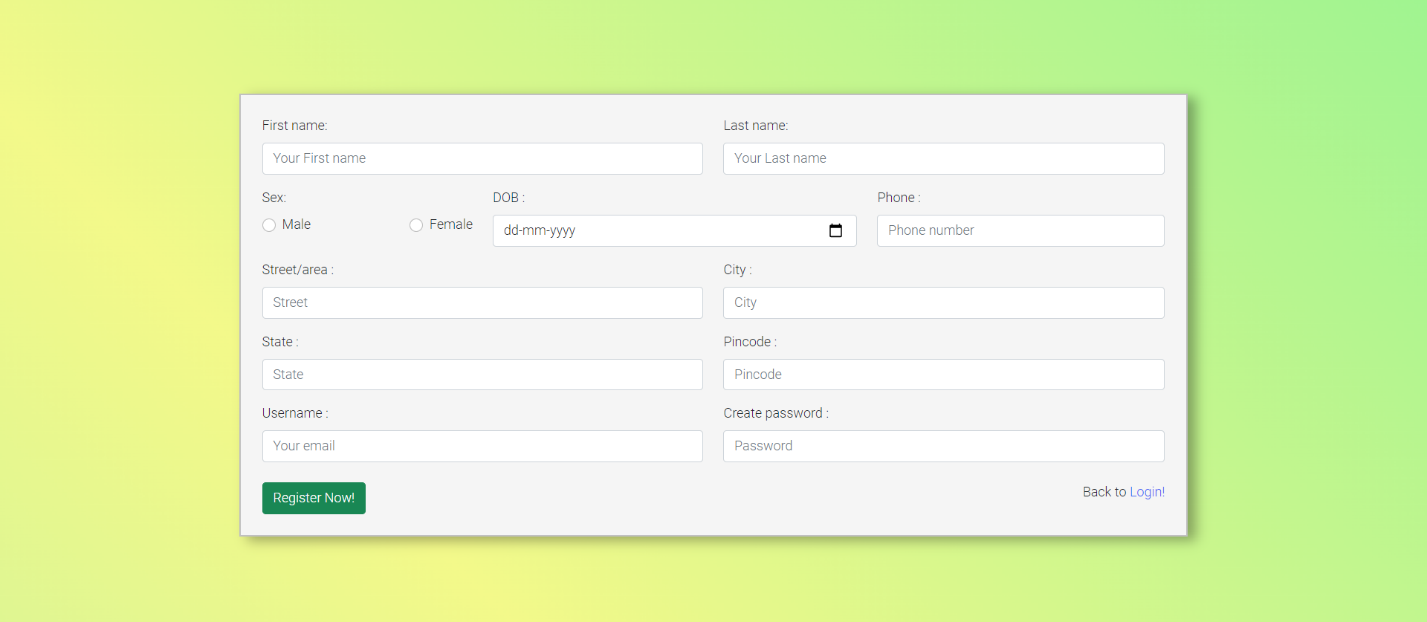
**RESULTS**

**4.1 SNAPSHOTS**

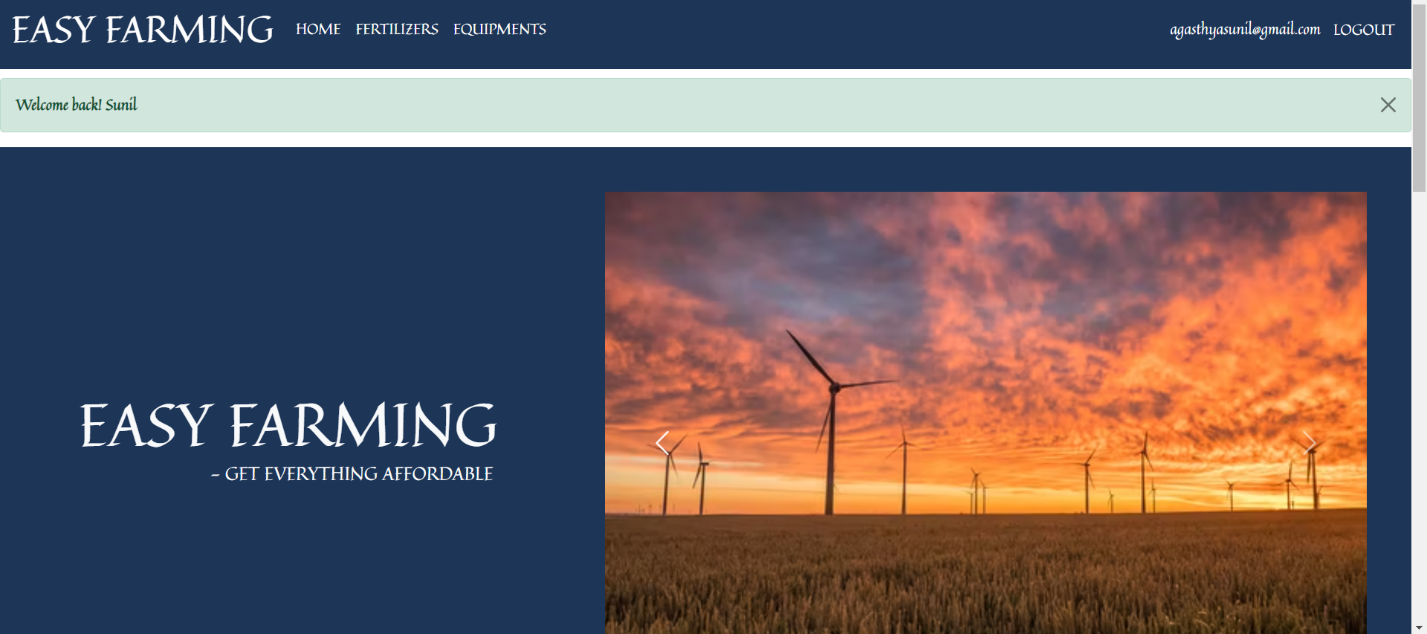
**Login Page:**

****

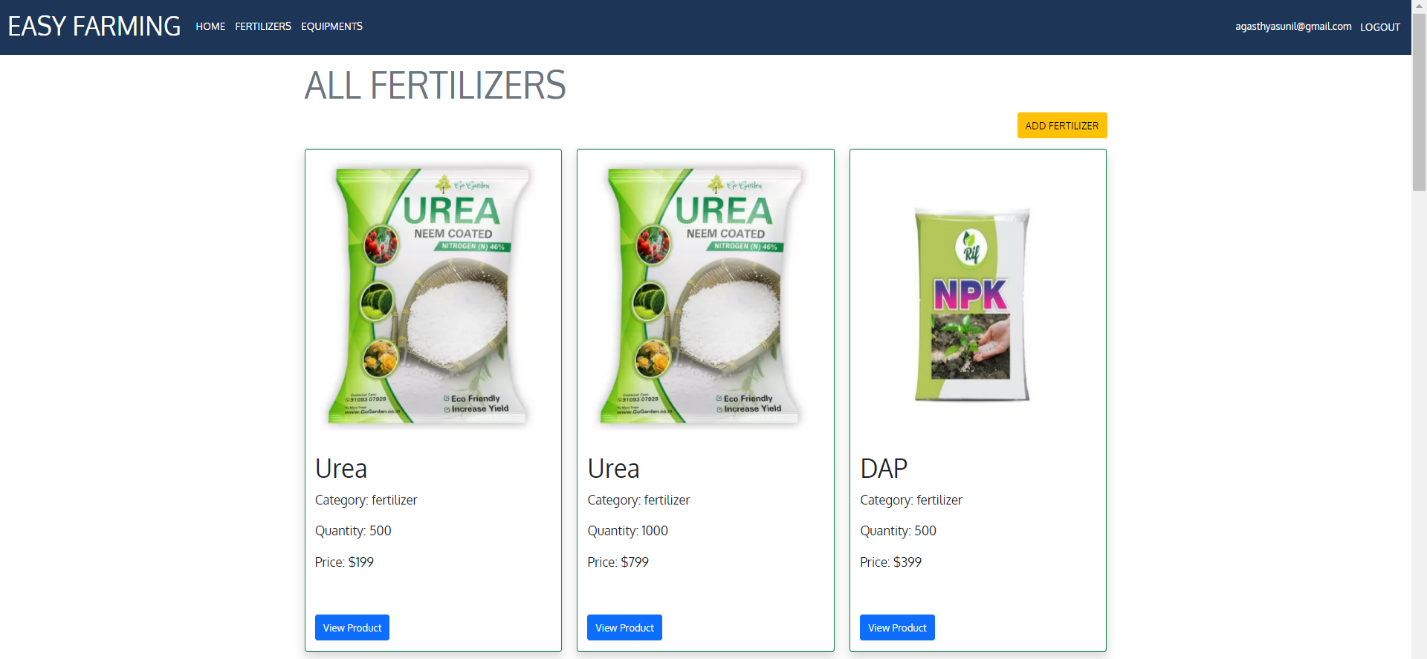
**Register Page:**

****

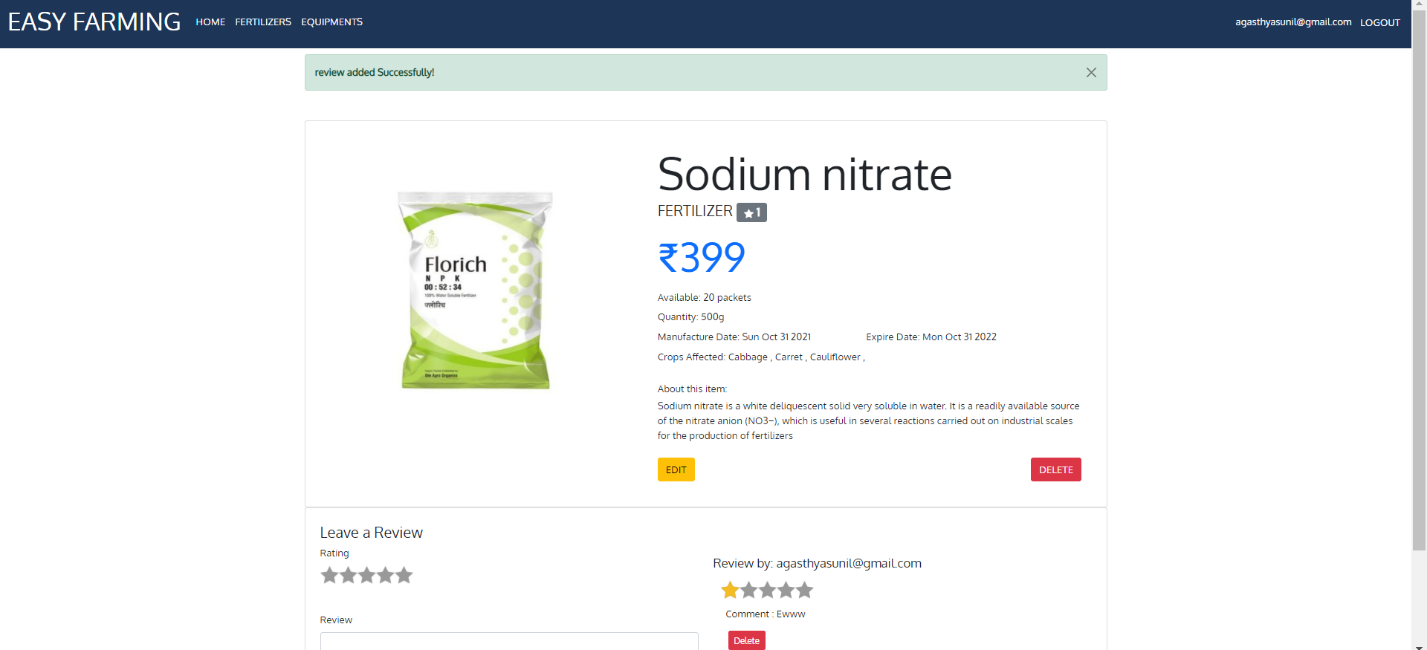
**Home Page:**

****

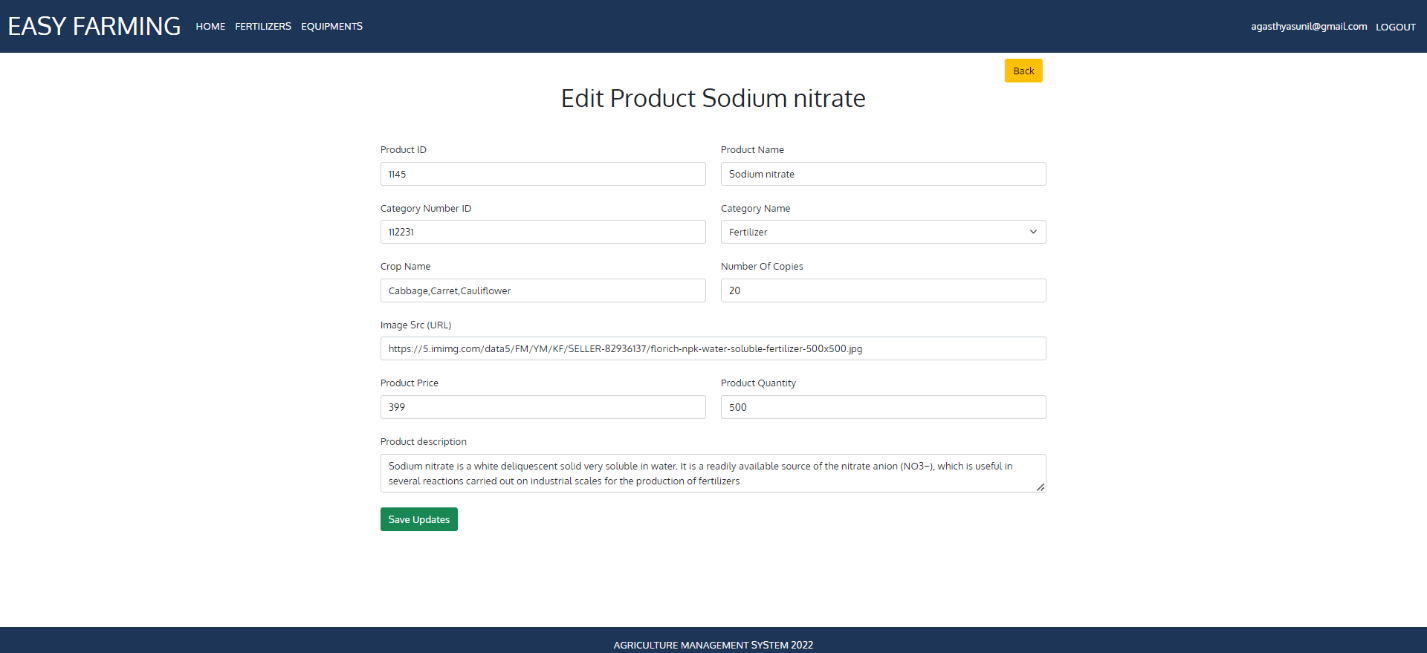
**Fertilizer Index Page:**

****

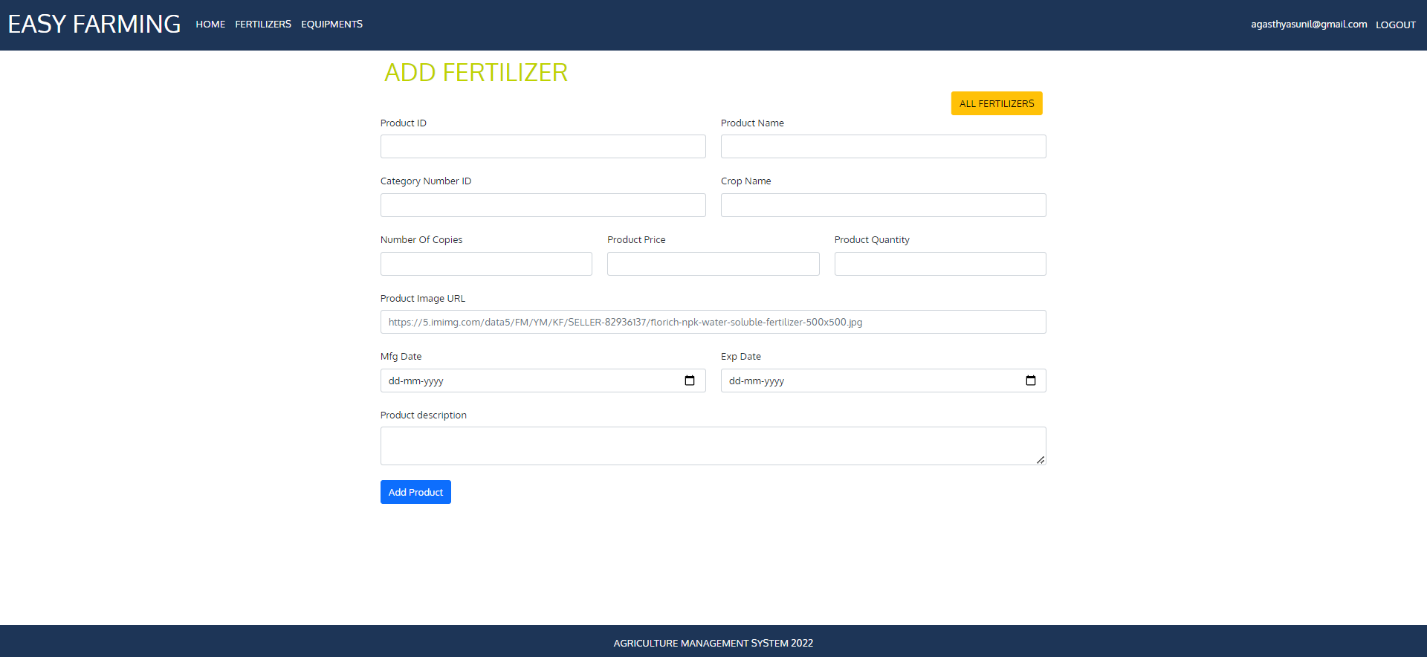
**Fertilizer Show Page:**

****

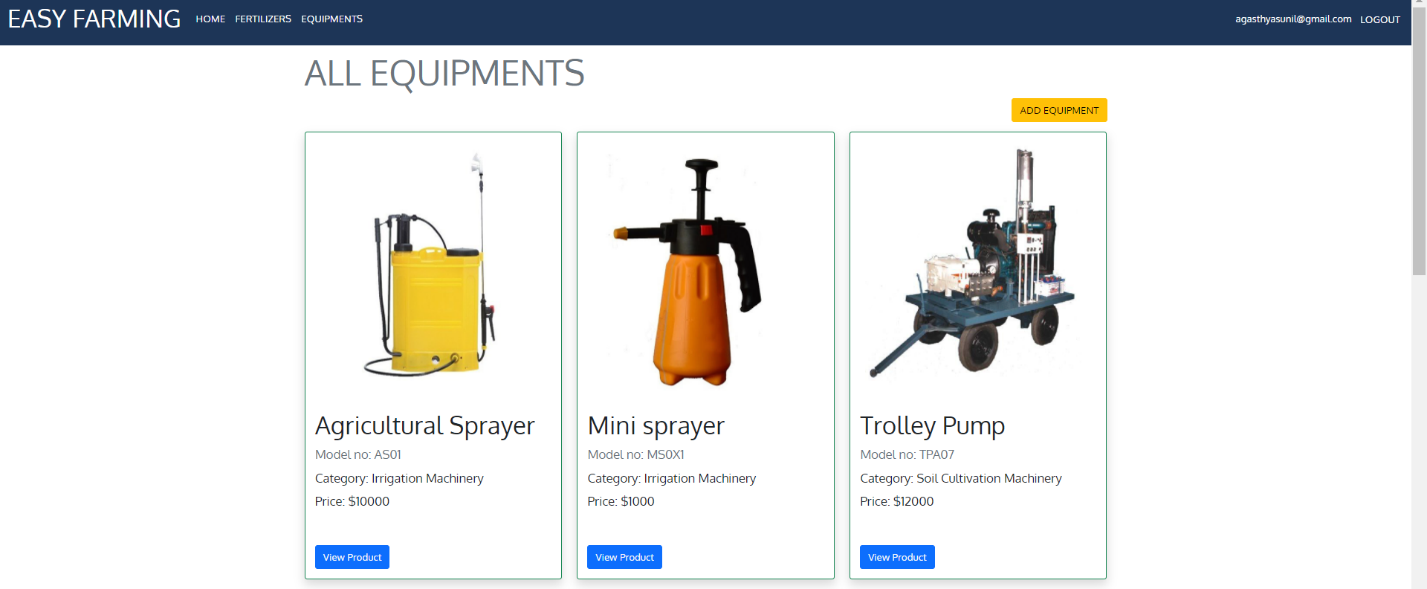
**Fertilizer Edit Page:**

****

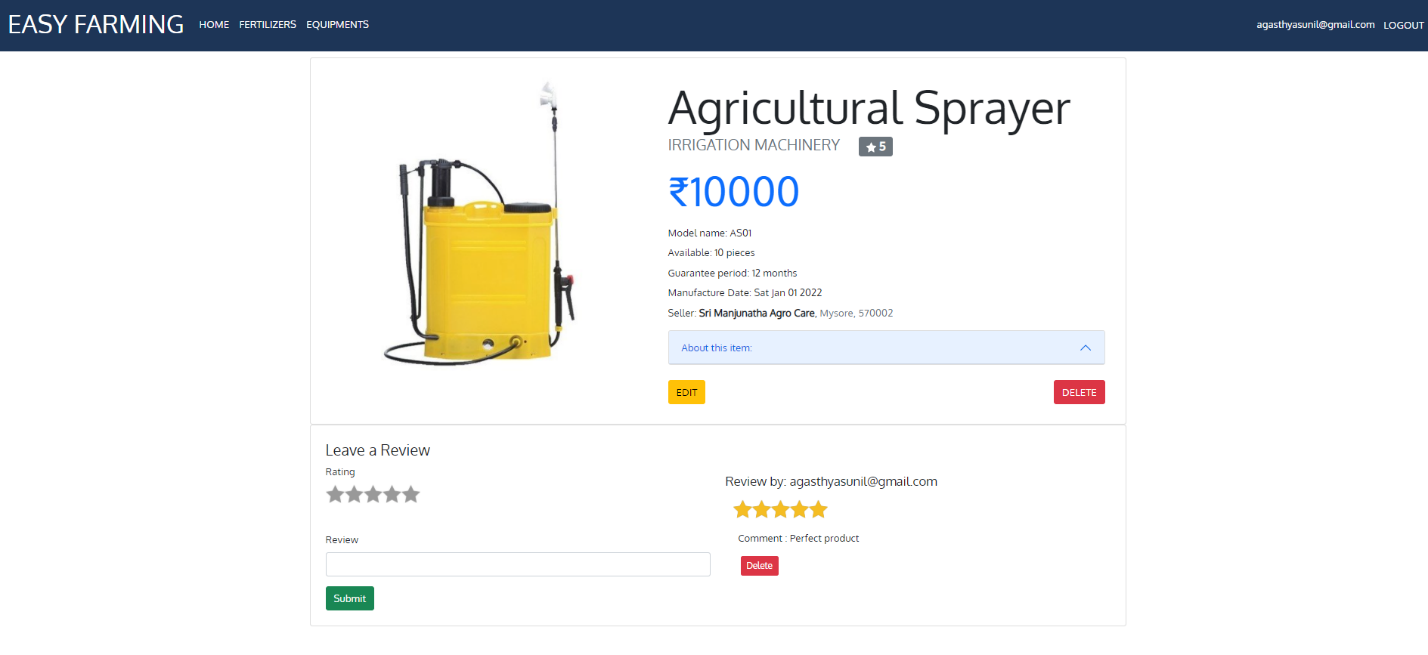
**Fertilizer Add Page:**

****

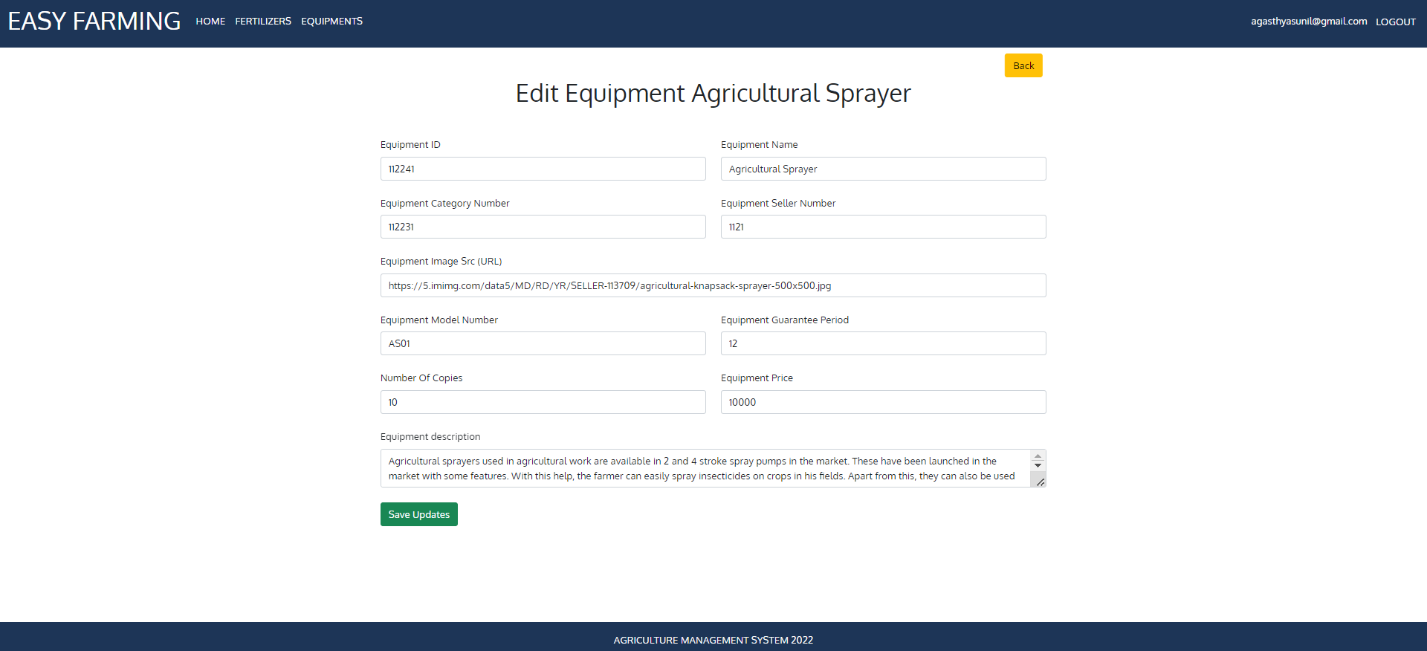
**Equipment Index Page:**

****

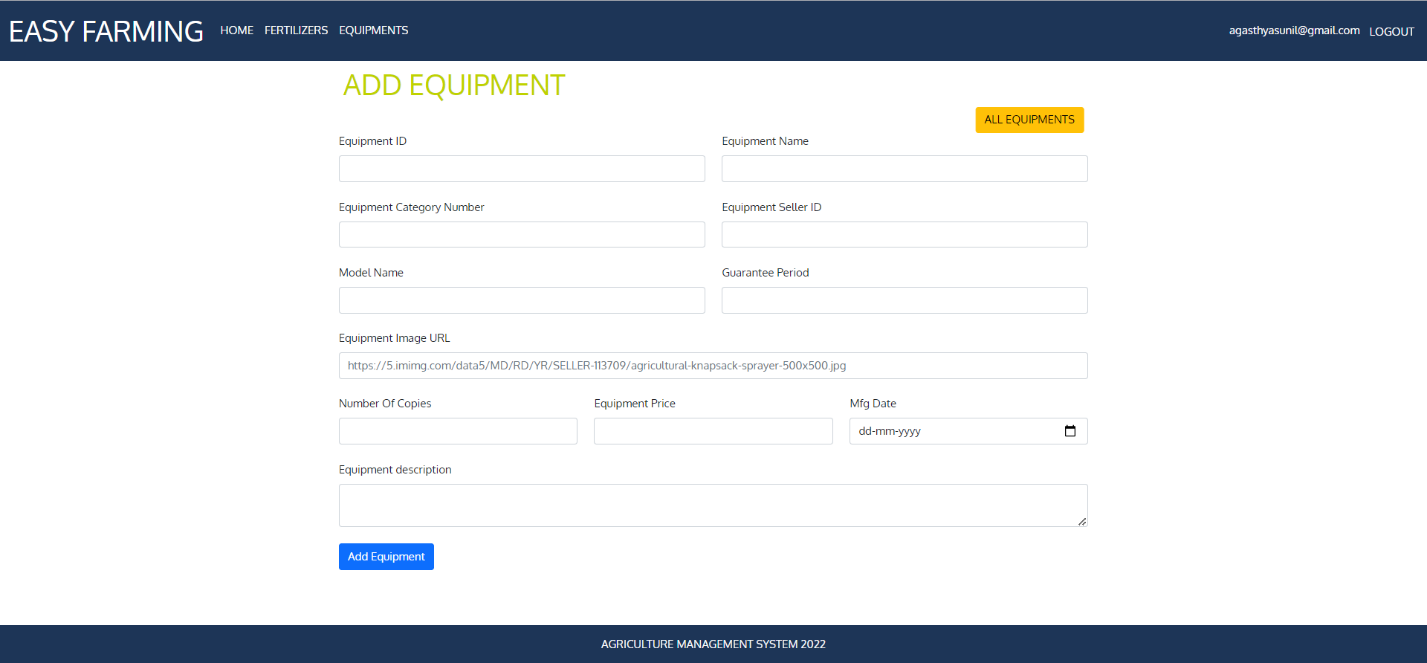
**Equipment Show page:**

****

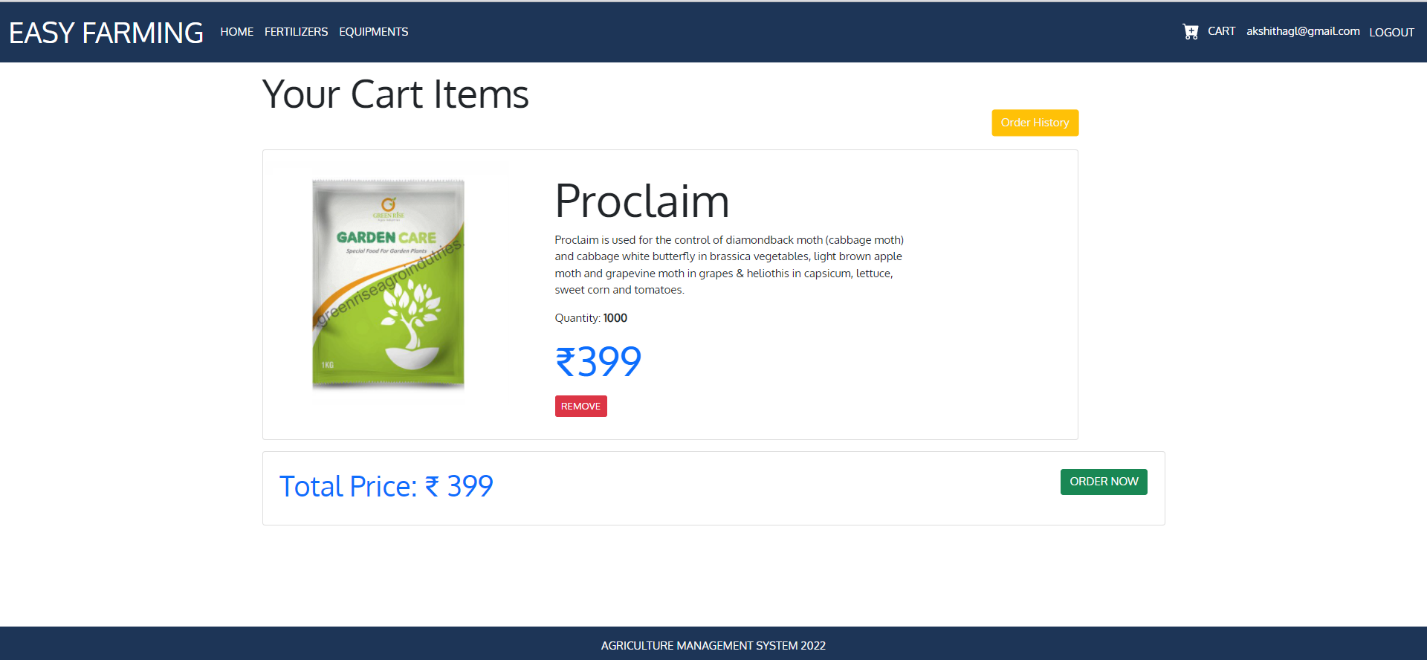
**Equipment Edit Page:**

****

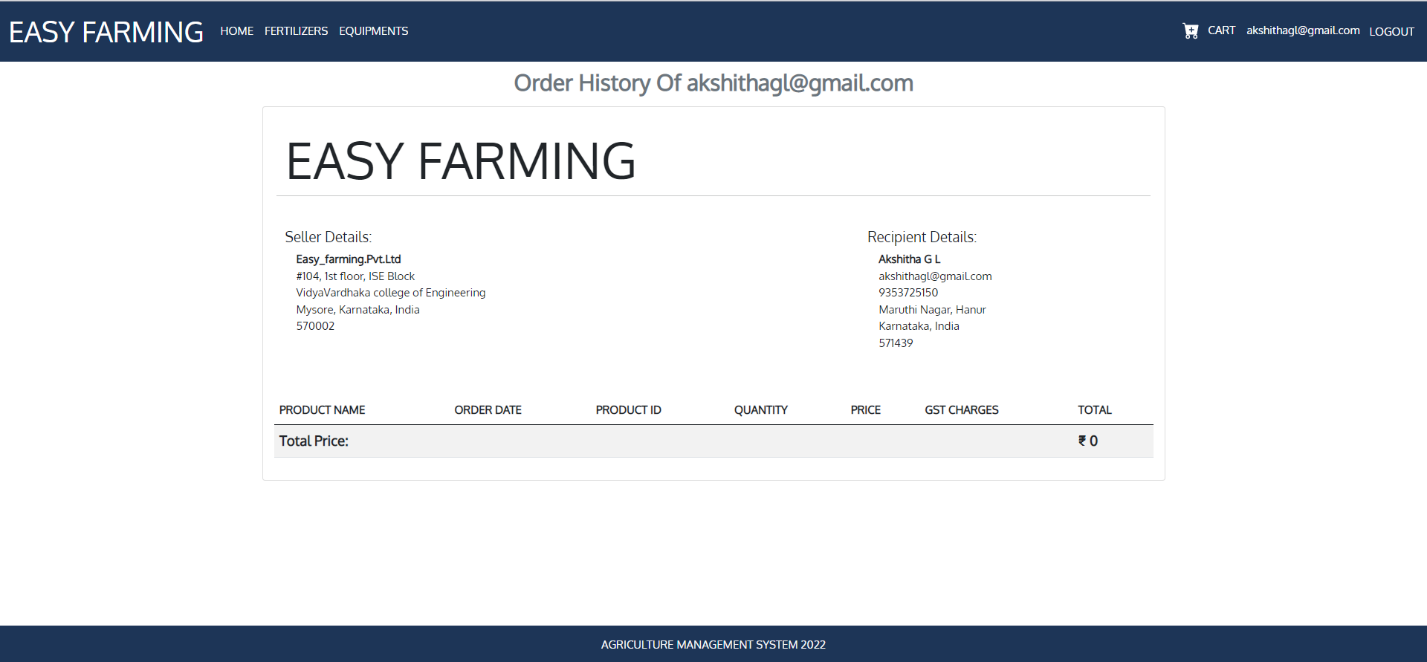
**Equipment Add page:**

****

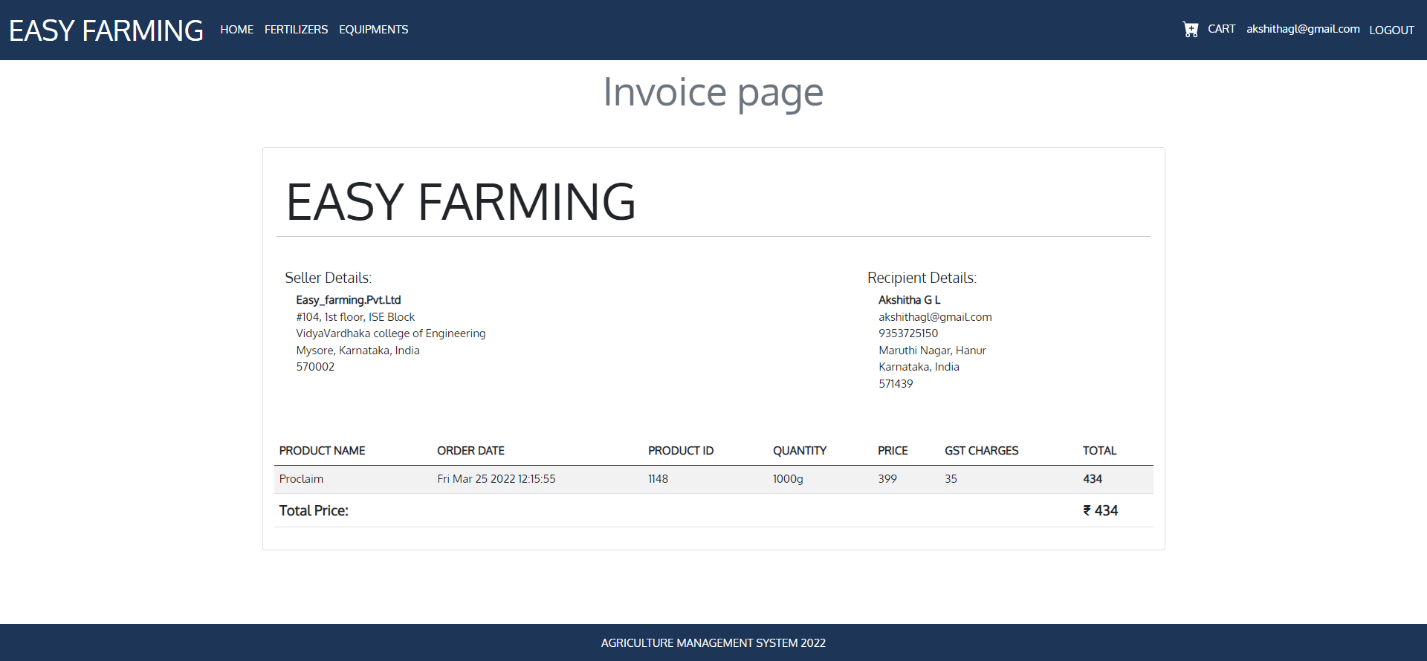
**User Cart Page:**

****

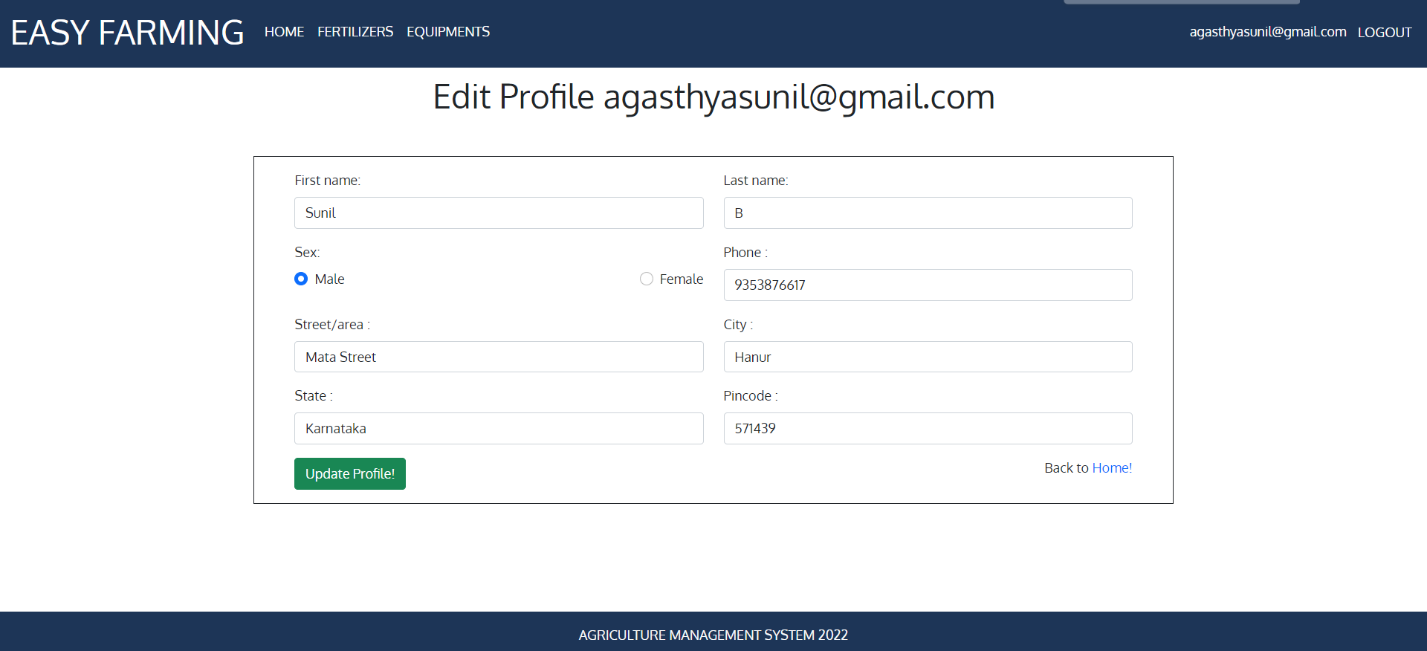
**User Order History:**

****

**Invoice Page:**

****

**User Profile:**

****

**CONCLUSION AND FUTURE ENHANCEMENT**

**CONCLUSION:**

Agriculture Management System is farmer management website application which helps farmers to give best-practice farming processes. It helps farmers to improve their productivity and profitability. It enables farmers to sell their productions through online and farmers can purchase tools and seeds directly from seller.

**FUTURE ENHANCEMENT:**

The project has a vast scope in future. Project can be updated in near future as and when required, as it is very flexible in terms of expansion. With the proposed software of database ready and fully functional, the admin is now able to manage the activities and hence run the entire work in a much better, accurate and error

**REFERENCES**

[1] About HTML:

URL: <https://en.wikipedia.org/wiki/HTML>

[2] About CSS:

URL: <https://en.wikipedia.org/wiki/CSS>

[3] About Visual Code:

URL: <https://code.visualstudio.com/docs/editor/whyvscode>

[4] About SQL - W3 Schools:

URL: <https://www.w3schools.com/sql/>

[5] About Xampp Server:

URL: <https://www.youtube.com/watch?v=-f8N4FEQWyY>

[6] About JS:

URL: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

[7] About Node:

URL: <https://nodejs.org/en/>

[8] About Colors palate:

URL: <https://coolors.co/palettes/trending>

[9] About Node MySQL:

URL: https://github.com/mysqljs/mysql

[10] About Express:

URL: <https://expressjs.com/>