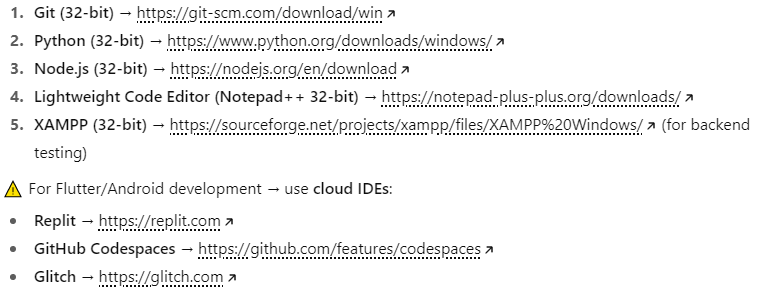
Required software:



**🔹 Step 1: Create Repo on GitHub**

1. Go to [GitHub](https://github.com).
2. Log in → Click on **New Repository**.
3. Fill in:
   * **Repository name** → krishisethu-app
   * Description → Platform connecting farmers with buyers (livestock, produce, etc.)
   * Choose **Public** (or Private if you prefer).
   * ✅ Check **“Add a README.md”** (recommended).
   * Leave .gitignore and license empty (we can add later).
4. Click **Create Repository**.

**🔹 Step 2: Clone Repo to Your Laptop**

Open your terminal/command prompt:

git clone https://github.com/<your-username>/krishisethu-app.git

cd krishisethu-app

(replace <your-username> with your GitHub username).

**🔹 Step 3: Setup Folder Structure**

Inside the repo:

mkdir backend frontend docs

cd backend

npm init -y

npm install express

Now create a file index.js inside backend/ with this code:

const express = require('express');

const app = express();

const PORT = 3000;

app.use(express.json());

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

res.send('Welcome to KrishiSethu API 🚜');

});

app.listen(PORT, () => {

console.log(`KrishiSethu backend running on http://localhost:${PORT}`);

});

**🔹 Step 4: Save and Push to GitHub**

Go back to repo root:

cd ..

git add .

git commit -m "Initial commit - setup backend with Express"

git push origin main

✅ Done! Now your krishisethu-app repo is created, and it already has your first backend server code.  
You can run it locally:

cd backend

node index.js

Visit → <http://localhost:3000> → you’ll see **“Welcome to KrishiSethu API 🚜”**.

**Possible Error to commit code:**

**C:\Users\Administrator\krishisethu-app>git commit -m "Initial commit - setup backend with Express"**

**Author identity unknown**

**\*\*\* Please tell me who you are.**

**Run**

**git config --global user.email "you@example.com"**

**git config --global user.name "Your Name"**

**to set your account's default identity.**

**Omit --global to set the identity only in this repository.**

**fatal: unable to auto-detect email address (got 'Administrator@Prasad-PC.(none)')**

**how to fix above**

## 🔹 Fix Steps

Run these commands in your terminal:

git config --global user.name "Your Name"

git config --global user.email "your-github-email@example.com"

👉 Replace with:

* "Your Name" → Your real name (or GitHub username, e.g., "Uday Reddy")
* "your-github-email@example.com" → The email linked to your GitHub account

### ✅ Example

If your GitHub account email is udayreddy123@gmail.com:

git config --global user.name "Uday Reddy"

git config --global user.email "udayreddy123@gmail.com"

## 🔹 Verify

Check if it’s saved correctly:

git config --list

You should see something like:

user.name=Uday Reddy

user.email=udayreddy123@gmail.com

## 🔹 Commit Again

Now retry:

git add .

git commit -m "Initial commit - setup backend with Express"

git push origin main

⚠️ **Note:** Make sure you use the **same email that is verified in GitHub**. Otherwise, commits won’t show

**Note:** run” node index.js” if you can’t reach <http://localhost:3000/>

### 5. (Optional) Auto-Restart on Changes

Install nodemon so you don’t have to restart manually every time:

npm install -g nodemon

nodemon index.js

⚡ So right now the issue is: **the server isn’t running**. Once you run node index.js inside the backend folder, it will work.

👉 Do you want me to also give you a **package.json start script** (npm start) so you don’t have to type node index.js every time?

**✅ Step 1: Open backend/package.json**

When you ran npm init -y, a file package.json was created.  
Open it → look for "scripts" section.

By default, it looks like this:

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

}

**✅ Step 2: Add Start Script**

Change it to:

"scripts": {

"start": "node index.js",

"dev": "nodemon index.js"

}

👉 Explanation:

* npm start → runs your server normally
* npm run dev → runs your server with **nodemon** (auto restart on changes)

**✅ Step 3: Run**

Inside backend/ folder:

npm start

Output:

KrishiSethu backend running on http://localhost:3000

If you installed nodemon:

npm run dev

Now the server auto-restarts whenever you change code.

⚡ Done! Now you don’t need to remember node index.js each time.

👉 Do you want me to also give you the **next step API route** (e.g., /livestock to return sheep, goats, hens data) so you can see your first real endpoint?

**🔹 How to Commit package.json Update**

1. Make sure you’re inside your repo root (krishisethu-app/).
2. Stage the changes:
3. git add backend/package.json
4. Commit the change:
5. git commit -m "Added start and dev scripts in package.json"
6. Push to GitHub:
7. git push origin main

✅ Done! Now your repo on GitHub will also have the updated package.json.

👉 Do you want me to also show you how to **ignore unnecessary files** (like nod

Perfect 👍 let's make your **first real API endpoint** in your backend/ folder using **Express**.  
We’ll start with just one module → **Livestock**. Later you can easily add grains, vegetables, etc.

**1️⃣ Install Express**

Run inside your backend/ folder:

npm install express

**2️⃣ Create index.js (main entry file)**

// backend/index.js

const express = require('express');

const app = express();

const PORT = process.env.PORT || 3000;

// Middleware (to parse JSON request body if needed later)

app.use(express.json());

// Livestock route

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

const data = [

{ id: 1, type: 'Sheep', count: 50 },

{ id: 2, type: 'Goat', count: 30 },

{ id: 3, type: 'Hen', count: 200 }

];

res.json(data);

});

// Root route (optional)

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

res.send('KrishiSethu API is running 🚜');

});

// Start server

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

**3️⃣ Run Your API**

👉 Development mode (with auto-restart):

npm run dev

👉 Production mode:

npm start

**4️⃣ Test in Browser / Postman**

* Open <http://localhost:3000/livestock>  
  You’ll see:

[

{ "id": 1, "type": "Sheep", "count": 50 },

{ "id": 2, "type": "Goat", "count": 30 },

{ "id": 3, "type": "Hen", "count": 200 }

]

✅ This gives you your **first working endpoint**.  
Next, we can split it into **modules** (routes/livestock.js, routes/grains.js, etc.) to keep it clean as your app grows.

👉 Do you want me to show you the **modular folder structure** (separate routes for each category) right now, or keep everything in index.js for the moment?

Perfect 🚀, let’s make it **modular**. That way when you later add grains, vegetables, etc., it will be clean and scalable.

Here’s the structure we’ll use:

krishisethu-app/

│── backend/

│ │── index.js

│ │── routes/

│ │ └── livestock.js

│ │── package.json

│ └── node\_modules/

### 1. Create a routes/livestock.js file

// routes/livestock.js

const express = require('express');

const router = express.Router();

// Example livestock data

const livestockData = [

{ id: 1, type: 'Sheep', count: 50 },

{ id: 2, type: 'Goat', count: 30 },

{ id: 3, type: 'Hen', count: 200 }

];

// GET /livestock

router.get('/', (req, res) => {

res.json(livestockData);

});

module.exports = router;

### 2. Update index.js to use this router

// index.js

const express = require('express');

const app = express();

const PORT = process.env.PORT || 3000;

// Middleware

app.use(express.json());

// Import routes

const livestockRoutes = require('./routes/livestock');

// Use routes

app.use('/livestock', livestockRoutes);

// Root endpoint

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

res.send('🌱 KrishiSethu API is running');

});

// Start server

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

### 3. Run your server

npm run dev

Now check 👉 <http://localhost:3000/livestock>  
You should see the livestock JSON.

Would you like me to also **add placeholder modules** (grains.js, vegetables.js) now with sample data, so you’re ready to expand?

Your new structure will look like this:

krishisethu-app/

│── backend/

│ │── index.js

│ │── routes/

│ │ ├── livestock.js

│ │ ├── grains.js

│ │ └── vegetables.js

│ │── package.json

│ └── node\_modules/

## 1️⃣ routes/livestock.js (already done)

// routes/livestock.js

const express = require('express');

const router = express.Router();

const livestockData = [

{ id: 1, type: 'Sheep', count: 50 },

{ id: 2, type: 'Goat', count: 30 },

{ id: 3, type: 'Hen', count: 200 }

];

router.get('/', (req, res) => {

res.json(livestockData);

});

module.exports = router;

## 2️⃣ routes/grains.js

// routes/grains.js

const express = require('express');

const router = express.Router();

const grainsData = [

{ id: 1, type: 'Rice', quantity: '100 kg' },

{ id: 2, type: 'Wheat', quantity: '75 kg' },

{ id: 3, type: 'Millet', quantity: '50 kg' }

];

router.get('/', (req, res) => {

res.json(grainsData);

});

module.exports = router;

## 3️⃣ routes/vegetables.js

// routes/vegetables.js

const express = require('express');

const router = express.Router();

const vegetablesData = [

{ id: 1, name: 'Tomato', quantity: '40 kg' },

{ id: 2, name: 'Potato', quantity: '60 kg' },

{ id: 3, name: 'Carrot', quantity: '25 kg' }

];

router.get('/', (req, res) => {

res.json(vegetablesData);

});

module.exports = router;

## 4️⃣ Update index.js to use all routes

// index.js

const express = require('express');

const app = express();

const PORT = process.env.PORT || 3000;

app.use(express.json());

// Import routes

const livestockRoutes = require('./routes/livestock');

const grainsRoutes = require('./routes/grains');

const vegetablesRoutes = require('./routes/vegetables');

// Use routes

app.use('/livestock', livestockRoutes);

app.use('/grains', grainsRoutes);

app.use('/vegetables', vegetablesRoutes);

// Root endpoint

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

res.send('🌱 KrishiSethu API is running');

});

// Start server

app.listen(PORT, () => {

console.log(`Server is running on http://localhost:${PORT}`);

});

## 5️⃣ Test in Browser

* <http://localhost:3000/livestock> → Sheep, Goats, Hens
* <http://localhost:3000/grains> → Rice, Wheat, Millet
* <http://localhost:3000/vegetables> → Tomato, Potato, Carrot

👉 Do you also want me to show you how to **structure this with a database (like MySQL/SQLite/Databricks)** instead of static JSON, so that farmers can add/update products later?

By default, Express doesn’t generate a **homepage UI** — it only exposes **API endpoints**.

* If you go to **http://localhost:3000/**, you’ll only see "Welcome to KrishiSethu API" (or whatever message you coded in / route).
* Your /api/livestock module is available at **http://localhost:3000/api/livestock**, but it won’t “list itself” on the homepage automatically.

👉 If you want modules visible on the home page, you have 2 options:

1. **Manually add a landing route** (e.g., return JSON listing available modules).
2. app.get("/", (req, res) => {
3. res.json({
4. message: "Welcome to KrishiSethu API",
5. modules: ["/api/livestock", "/api/grains", "/api/vegetables"]
6. });
7. });
8. **Later build a frontend (React, Angular, etc.)** that consumes these APIs and shows modules in a UI.

So right now you **can hit the endpoints directly in the browser or Postman**, but won’t see them auto-listed on the homepage unless you add that JSON response.

Do you want me to update your home route to show all available modules?

Here’s how you can update your index.js so that your **home page shows available modules**:

const express = require("express");

const app = express();

// Import routes

const livestockRoutes = require("./routes/livestock");

// Middleware

app.use(express.json());

// Routes

app.use("/api/livestock", livestockRoutes);

// Home route with module listing

app.get("/", (req, res) => {

res.json({

message: "Welcome to KrishiSethu API 🚜",

availableModules: {

livestock: "/api/livestock",

grains: "/api/grains",

vegetables: "/api/vegetables"

}

});

});

// Start server

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => console.log(`Server running on port ${PORT}`));

👉 Now, when you open [**http://localhost:3000/**](http://localhost:3000/) in your browser, you’ll see:

{

"message": "Welcome to KrishiSethu API 🚜",

"availableModules": {

"livestock": "/api/livestock",

"grains": "/api/grains",

"vegetables": "/api/vegetables"

}

}

And if you go to [**http://localhost:3000/api/livestock**](http://localhost:3000/api/livestock), you’ll see your sheep, goat, hen data.

Do you want me to also add **dummy modules for grains and vegetables** (just like livestock) so the homepage links won’t break?

Right now your index.js just sends text like "Welcome to KrishiSethu API". If you want clickable hyperlinks (at least when opened in the browser), you just need to return a small HTML page instead of plain text.

Here’s an updated **index.js** with hyperlinks to your modules (livestock, grains, vegetables).

const express = require("express");

const app = express();

const PORT = 3000;

// Import routes

const livestockRoutes = require("./routes/livestock");

// Middleware

app.use(express.json());

// Routes

app.use("/livestock", livestockRoutes);

// Home page with hyperlinks

app.get("/", (req, res) => {

res.send(`

<h1>Welcome to KrishiSethu App</h1>

<p>Select a category:</p>

<ul>

<li><a href="/livestock">Livestock</a></li>

<li><a href="/grains">Grains (coming soon)</a></li>

<li><a href="/vegetables">Vegetables (coming soon)</a></li>

</ul>

`);

});

// Start server

app.listen(PORT, () => {

console.log(`Server running on http://localhost:${PORT}`);

});

✅ This way when you open [**http://localhost:3000/**](http://localhost:3000/) in the browser:

* You’ll see a small HTML page
* With clickable links: /livestock, /grains, /vegetables

Right now only /livestock will work because you have that route. The other two will show "Cannot GET /grains" until we create those modules.

👉 Do you want me to also create **placeholder routes** for grains and vegetables so that they show some sample JSON instead of an error?

yes, just put coming soon inside the grains and vegetables

Perfect ✅ we’ll add **placeholder routes** for grains and vegetables so they don’t give errors, just return "Coming soon" until you build them.

Here’s the updated structure:

### 1. routes/livestock.js (already done)

const express = require("express");

const router = express.Router();

router.get("/", (req, res) => {

res.json([

{ id: 1, type: "Sheep", price: 5000 },

{ id: 2, type: "Goat", price: 7000 },

{ id: 3, type: "Hen", price: 300 }

]);

});

module.exports = router;

### 2. routes/grains.js (new file)

const express = require("express");

const router = express.Router();

router.get("/", (req, res) => {

res.send("Grains module coming soon!");

});

module.exports = router;

### 3. routes/vegetables.js (new file)

const express = require("express");

const router = express.Router();

router.get("/", (req, res) => {

res.send("Vegetables module coming soon!");

});

module.exports = router;

### 4. Updated index.js

const express = require("express");

const app = express();

const PORT = 3000;

// Import routes

const livestockRoutes = require("./routes/livestock");

const grainsRoutes = require("./routes/grains");

const vegetablesRoutes = require("./routes/vegetables");

// Middleware

app.use(express.json());

// Routes

app.use("/livestock", livestockRoutes);

app.use("/grains", grainsRoutes);

app.use("/vegetables", vegetablesRoutes);

// Home page with hyperlinks

app.get("/", (req, res) => {

res.send(`

<h1>Welcome to KrishiSethu App</h1>

<p>Select a category:</p>

<ul>

<li><a href="/livestock">Livestock</a></li>

<li><a href="/grains">Grains</a></li>

<li><a href="/vegetables">Vegetables</a></li>

</ul>

`);

});

// Start server

app.listen(PORT, () => {

console.log(`Server running on http://localhost:${PORT}`);

});

👉 Now when you open [**http://localhost:3000/**](http://localhost:3000/) you’ll see 3 hyperlinks.

* /livestock → shows JSON of sheep, goats, hens
* /grains → shows "Grains module coming soon!"
* /vegetables → shows "Vegetables module coming soon!"

Would you like me to also show you how to **style this home page with a simple HTML+CSS page** (so it looks more like a real app UI), or for now keep it just plain hyperlinks?

Best approach 👉 **start with backend (database + API) first, then UI.**

Reason (short):

* If you build UI first without schema, you won’t know what data to display.
* With schema + APIs ready, the UI can just plug into them.
* It avoids redesigning UI again and again.

✅ So: **Step 1 → define schema & APIs in Databricks (or any DB)**  
➡️ Step 2 → connect UI (web/app) to those APIs.

since you want **free of cost**, here’s the practical path:

### ✅ Best Free Options for Your App

1. **SQLite**
   * Comes bundled with Python & Node.js.
   * File-based, no server needed.
   * Perfect for **starting small (livestock orders, farmers, buyers)**.
   * Zero hosting cost.
2. **PostgreSQL / MySQL (Free on Cloud tiers)**
   * Free on **Azure (12 months free)**, AWS, or Heroku.
   * Good if you want multi-user transactions, better than SQLite.
   * Needs little more setup.
3. **Databricks (Free Community Edition)**
   * Only good for **analytics / ML**, not for your live transactional app.
   * Use later when you scale and want insights like demand forecast.

### 🚀 Suggested Approach for You (free + practical)

* **Start with SQLite** → simplest, no cost, quick coding.
* Once you need **real hosting & multi-user**, migrate to **Postgres/MySQL free cloud tier**.
* Later, connect **Databricks Community Edition** just for analytics, not as main DB.

⚖️ So, **for your app’s current stage (livestock module, small town pilot)**:  
👉 **Use SQLite (free, local, no infra cost).**

Do you want me to show you a **SQLite schema design** (tables for Farmers, Buyers, Products, Orders) so you can start coding right away?