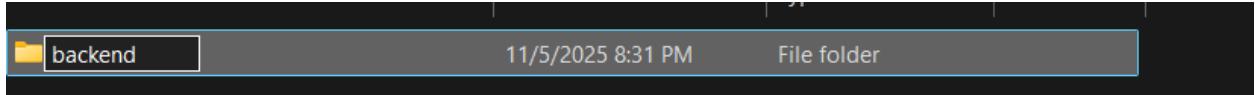


MYSQL + EXPRESS + VUE + NODE.JS

STEP 1 — Project Structure

- Create a parent folder anywhere on your PC.
- Inside, make two subfolders:



Note. Later we will create our frontend folder through vite-vue installation.

STEP 2 — Set up the Backend (Node + Express + MySQL)

The backend exposes REST API routes:

- GET /api/users → get all users
- POST /api/users → create user
- PUT /api/users/:id → update user
- DELETE /api/users/:id → delete user

2.1 — Initialize Node project

```
PS C:\Users\asus\Desktop\full-stack\backend> npm init -y
  >> npm install express cors dotenv mysql2
  >> npm install -D nodemon
```

After:

A screenshot of a terminal window within the Visual Studio Code interface. The window has a dark theme. On the left, there's an 'EXPLORER' sidebar showing a folder named 'BACKEND' containing 'node_modules', 'package-lock.json', and 'package.json'. The main terminal area shows the command 'npm init -y' being run, followed by the output of the initialization process. The output includes details about packages added, vulnerabilities found, and audit results. The terminal tabs at the bottom include 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'.

- **express** -creates web server
- **cors** - allows requests from your Vue app

- **dotenv** - reads config from .env
- **mysql2** - connects to MySQL
- **nodemon** - restarts automatically when code changes (for development)

2.2 — Database connection file

Create db.js file.

```
const mysql = require('mysql2/promise');
require('dotenv').config();

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  port: process.env.DB_PORT,
  waitForConnections: true,
  connectionLimit: 10
});

module.exports = pool;
```

Explanation:

const mysql = require('mysql2/promise');

- This line imports the mysql2 library, which is a faster and more modern version of the original mysql library.
- The /promise version allows you to use async/await syntax instead of callbacks, making code cleaner and easier to manage.

require('dotenv').config();

- This loads your .env file so that environment variables (like DB username, password, etc.) are available via process.env.

```
const pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  database: process.env.DB_NAME,
  port: process.env.DB_PORT,
  waitForConnections: true,
  connectionLimit: 10
});
```

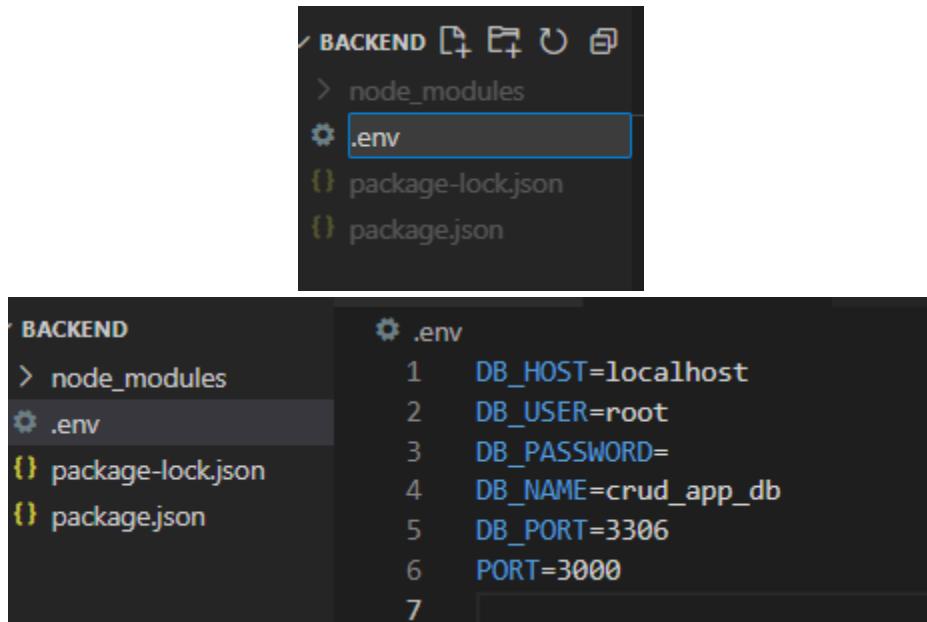
- mysql.createPool() creates a pool of database connections instead of just one connection.
- This means Node.js can reuse connections efficiently for multiple database queries.
- It's better for performance and scalability.

- The options:
 - host: Database server address (usually localhost or an IP).
 - user: MySQL username.
 - password: MySQL password.
 - database: Database name you want to connect to.
 - port: Port number (default for MySQL is 3306).
 - waitForConnections: When all connections are in use, new requests will wait until one is free.
 - connectionLimit: Maximum number of simultaneous connections (10 in this case).

```
module.exports = pool;
```

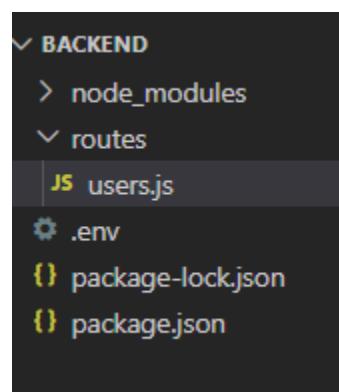
- This exports the pool so that other files in your project can use the same connection.

2.3 — Environment configuration



Why use .env: Keeps credentials separate so you can change them without editing code.

2.4 — Create REST API routes



Inside the backend folder, create a new folder and add a users.js file to contain all CRUD logic for organized code under api/users.

```
const express = require('express');
const router = express.Router();
const pool = require('../db');

// GET all users
router.get('/', async (req, res, next) => {
  try {
    const [rows] = await pool.execute('SELECT * FROM users ORDER BY id DESC');
    res.json(rows);
  } catch (err) {
    next(err);
  }
});

// POST create
router.post('/', async (req, res, next) => {
  try {
    const { name, email, age } = req.body;
    const [result] = await pool.execute(
      'INSERT INTO users (name, email, age) VALUES (?, ?, ?)',
      [name, email, age || null]
    );
    const [rows] = await pool.execute('SELECT * FROM users WHERE id=?',
[ result.insertId ]);
    res.status(201).json(rows[0]);
  } catch (err) {
    next(err);
  }
});

// PUT update
router.put('/:id', async (req, res, next) => {
  try {
    const { name, email, age } = req.body;
    await pool.execute(
      'UPDATE users SET name=?, email=?, age=? WHERE id=?',
      [name, email, age || null, req.params.id]
    );
    const [rows] = await pool.execute('SELECT * FROM users WHERE id=?',
[ req.params.id ]);
    res.json(rows[0]);
  } catch (err) {
    next(err);
  }
});
```

```

// DELETE
router.delete('/:id', async (req, res, next) => {
  try {
    await pool.execute('DELETE FROM users WHERE id=?', [req.params.id]);
    res.json({ message: 'User deleted' });
  } catch (err) {
    next(err);
  }
});

module.exports = router;

```

Explanation:

```

const express = require('express');
const router = express.Router();
const pool = require('../db');

```

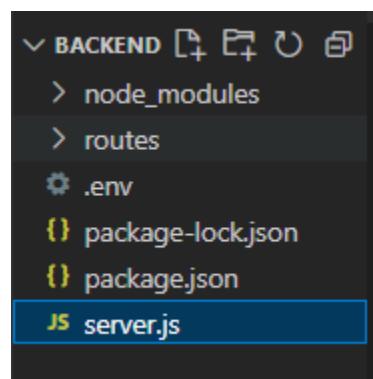
- `express.Router()` - lets you define routes separately from the main server.
- `pool` - MySQL connection pool from `db.js`, used for executing queries.

Final API endpoints

- **GET /api/users**
- **POST /api/users**
- **PUT /api/users/:id**
- **DELETE /api/users/:id**

2.5 — Main server file

Create a `server.js` file under `backend` folder.



```

require('dotenv').config();
const express = require('express');
const cors = require('cors');
const usersRouter = require('./routes/users');

const app = express();
app.use(cors());
app.use(express.json());

app.get('/', (req, res) => res.send({ message: 'API running...' }));
app.use('/api/users', usersRouter);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Backend running at http://localhost:${PORT}`));

```

Explanation:

```
const express = require('express');
const cors = require('cors');
const usersRouter = require('./routes/users');
```

- express - the main web framework for building APIs.
- cors - allows your frontend (Vue) and backend (Node) to communicate, even though they run on different ports (5173 & 3000).
- usersRouter - imports all CRUD routes from routes/users.js.

```
const app = express();
```

- This creates an Express application instance — basically your API server.

```
app.use(cors());
app.use(express.json());
```

- cors() - enables Cross-Origin Resource Sharing. Without it, your browser would block API calls from Vue (different port).
- express.json() - tells Express to automatically parse JSON bodies in requests (like when Vue sends POST or PUT).

```
app.get('/', (req, res) => res.send({ message: 'API running...' }));
```

```
app.use('/api/users', usersRouter);
```

- Define routes

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

```
app.listen(PORT, () => console.log(`Backend running at
http://localhost:${PORT}`));
```

- Reads the port from .env, or uses 3000 by default.
- app.listen() starts the web server and listens for incoming requests.
- Logs a message to the console when the server is up and running.

2.6 — package.json scripts

Before:

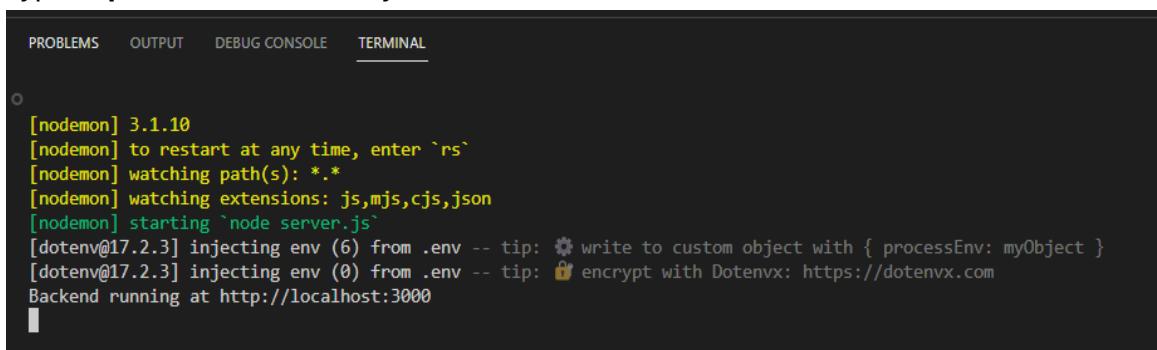
```
0 package.json > ...
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9 }
```

Updated:

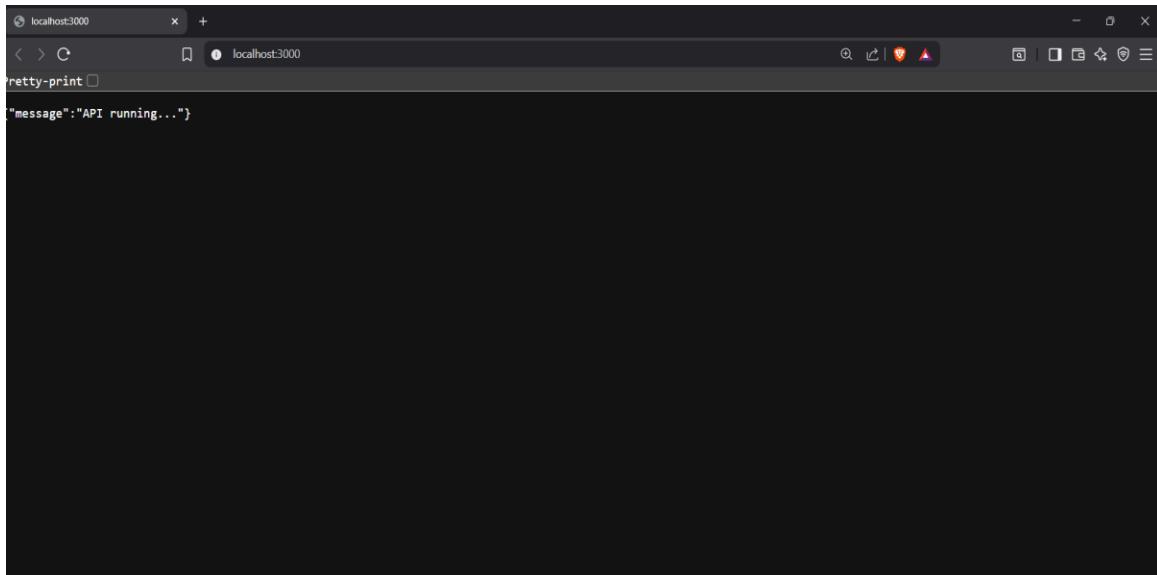
```
0 package.json > ...
1 {
2   "name": "backend",
3   "version": "1.0.0",
4   "main": "server.js",
5   "scripts": {
6     "dev": "nodemon server.js",
7     "start": "node server.js"
8   },
9 }
```

Testing:

- Type “**npm run dev**” to test your backend.



```
[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
[dotenv@17.2.3] injecting env (6) from .env -- tip: ⚡ write to custom object with { processEnv: myObject }
[dotenv@17.2.3] injecting env (0) from .env -- tip: 🔒 encrypt with Dotenvx: https://dotenvx.com
Backend running at http://localhost:3000
```

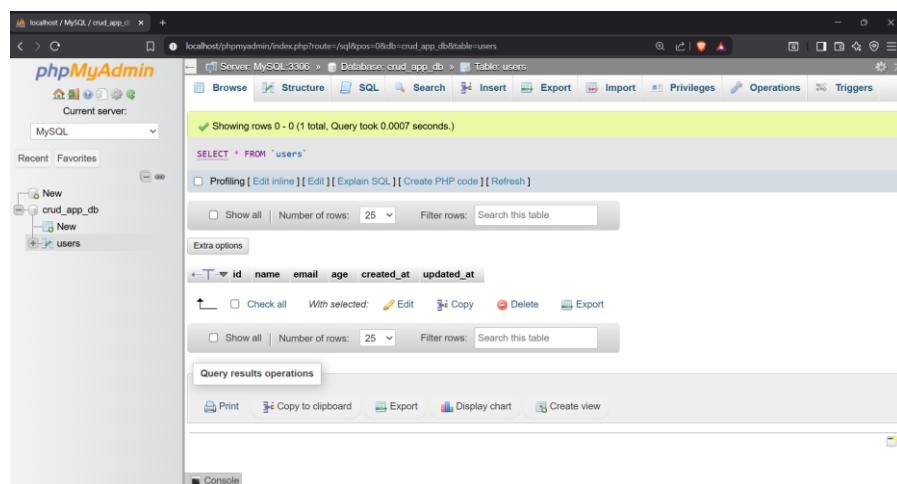



```
{"message": "API running..."}
```

2.7 — Create MySQL database

```
CREATE DATABASE crud_app_db;
USE crud_app_db;

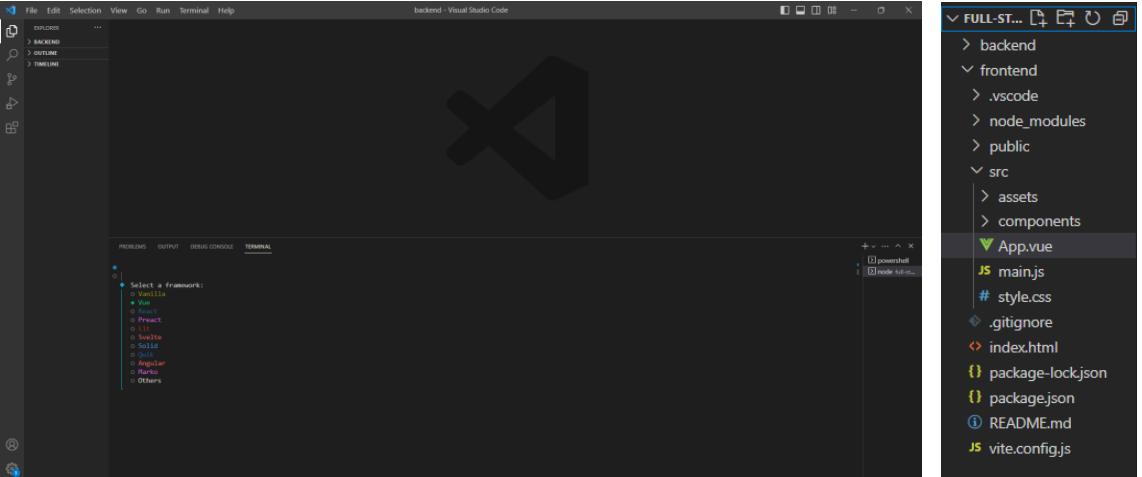
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(150) UNIQUE,
    age INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```



STEP 3 — Frontend (Vue 3 + Vite)

- Install Vue using vite.

```
PS C:\Users\asus\Desktop\full-stack\backend> cd ..
PS C:\Users\asus\Desktop\full-stack> npm create vite@latest frontend
```



- Then install dependencies.

```
PS C:\Users\asus\Desktop\full-stack\frontend> npm install
>> npm install axios
```

3.2 — Configure Vite for backend proxy

```
vite.config.js X

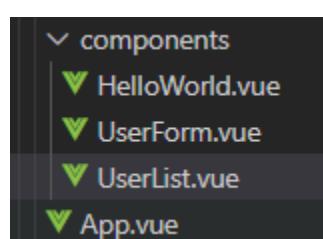
frontend > vite.config.js > ...
1 import { defineConfig } from 'vite'
2 import vue from '@vitejs/plugin-vue'
3
4 // https://vite.dev/config/
5 export default defineConfig({
6   plugins: [vue()],
7   server: {
8     proxy: { '/api': 'http://localhost:3000' }
9   }
10})
11|
```

Add that line in vite.config.js file so Vue dev server automatically forwards /api/... requests to Express.

3.3 — Add environment variable

```
frontend > .env
1 VITE_API_BASE=http://localhost:3000/api
```

3.4 – Create your components.



STEP 4 — Combine Frontend & Backend

- At the project root (full-stack/), create this file:

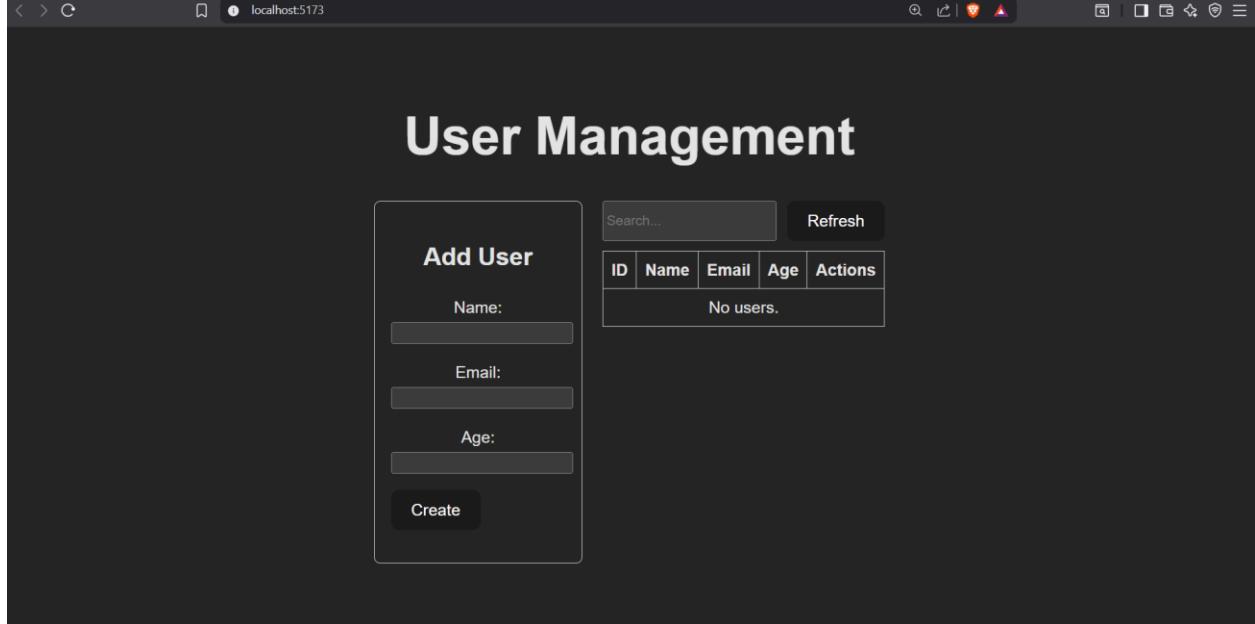
```
FULL-STACK          package.json > ...
> backend
> frontend
{} package.json
1  {
2    "name": "full-stack",
3    "version": "1.0.0",
4    "scripts": {
5      "start": "concurrently \"npm run dev --prefix backend\" \"npm run dev --prefix frontend\""
6    },
7    "devDependencies": {
8      "concurrently": "^9.0.0"
9    }
10 }
11
```

- Install and run:

```
PS C:\Users\asus\Desktop\full-stack> npm install
>> npm start
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
node + v 🌐 ⏪ ... ^ x
> concurrently "npm run dev --prefix backend" "npm run dev --prefix frontend"

[0]
[0] > backend@1.0.0 dev
[0] > nodemon server.js
[0]
[1]
[1] > frontend@0.0.0 dev
[1] > vite
[1]
[0] [nodemon] 3.1.10
[0] [nodemon] to restart at any time, enter `rs`
[0] [nodemon] watching path(s): *
[0] [nodemon] watching extensions: js,mjs,cjs,json
[0] [nodemon] starting `node server.js`
[0] [dotenv@17.2.3] injecting env (6) from .env -- tip: ⚡ encrypt with Dotenvx: https://dotenvx.com
[0] [dotenv@17.2.3] injecting env (0) from .env -- tip: ⚡ run anywhere with `dotenvx run -- yourcommand`
[0] Backend running at http://localhost:3000
[1] 3:41:16 PM [vite] (client) Re-optimizing dependencies because lockfile has changed
[1]
[1] VITE v7.2.0 ready in 1077 ms
[1]
[1] → Local: http://localhost:5173/
[1] → Network: use --host to expose
```



The screenshot displays two windows side-by-side, illustrating a full-stack application development environment.

Frontend Application (User Management):

- The title bar says "frontend".
- The address bar shows "localhost:5173".
- The main content area features a large heading "User Management".
- To the left is a form titled "Add User" with fields for Name (Anna), Email (anna@gmail.com), and Age (20). A "Create" button is present below the fields.
- To the right is a table with columns ID, Name, Email, Age, and Actions. It contains one row with ID 4, Name Anna, Email anna@gmail.com, Age 20, and Actions "Edit" and "Delete".
- A green "Saved!" message is displayed at the bottom of the form.

Backend Database (phpMyAdmin):

- The title bar says "localhost / MySQL / crud_app_db".
- The address bar shows "localhost/phpmyadmin/index.php?route=/sql&pos=0&db=crud_app_db&table=users".
- The sidebar shows the database structure with "crud_app_db" selected, containing "New" and "users".
- The main content shows the "users" table with the following data:

	<th>name</th> <th>email</th> <th>age</th> <th>created_at</th> <th>updated_at</th>	name	email	age	created_at	updated_at
	4	Anna	anna@gmail.com	20	2025-11-05 21:42:43	2025-11-05 21:42:43

- Below the table are "Query results operations" buttons for Print, Copy to clipboard, Export, Display chart, and Create view.