

1. Create a GitHub OAuth App

- Go to GitHub → Settings → Developer Settings → OAuth Apps → New OAuth App.
- Set:
 - **Application Name:** TaskLedger GitHub Integration
 - **Homepage URL:** Your frontend URL (<https://taskledger.com> or <http://localhost:3000>)
 - **Authorization callback URL:** Backend endpoint to handle OAuth response (<https://api.taskledger.com/github/callback/>)
- Required scopes:
 - `repo` → access all repos created by the user, including private repos.
 - `read:user` → fetch user info (username, email).

2. User Connects GitHub (Frontend)

- Add a “Connect GitHub” button in the profile page.
- Button redirects the user to GitHub OAuth consent URL:

```
https://github.com/login/oauth/authorize  
?client\_id=YOUR\_CLIENT\_ID  
&scope=repo%20read:user  
&redirect\_uri=YOUR\_CALLBACK\_URL
```

- The user logs in to GitHub and authorizes access.

3. GitHub Redirects with Code

- GitHub sends a **temporary code** to your backend via the callback URL:

```
GET /github/callback/?code=TEMP_CODE
```

4. Backend Exchanges Code for Access Token

- Your backend sends a POST request to GitHub:

```
POST https://github.com/login/oauth/access_token
```

Headers:

```
Accept: application/json
```

Body:

```
client_id=YOUR_CLIENT_ID  
client_secret=YOUR_CLIENT_SECRET  
code=TEMP_CODE
```

- GitHub responds with:

```
{  
  "access_token": "USER_ACCESS_TOKEN",  
  "scope": "repo,read:user",  
  "token_type": "bearer"  
}
```

- Save **access_token securely** in your DB (encrypt at rest). Link it to the **User** record.
-

5. Fetch User Repositories

- Backend uses the stored access token to call GitHub API:

GET user repos created by the user:

```
GET https://api.github.com/user/repos
```

Headers:

Authorization: Bearer USER_ACCESS_TOKEN

- Filter only repos where `owner.login == github_username` and `fork == false` to analyze original repos created by the user.
 - Include private repos, since `repo` scope allows it.
-

6. Analyze Contributions

- For each repo, fetch:
 - Commits: `GET /repos/{owner}/{repo}/commits`
 - Pull requests: `GET /repos/{owner}/{repo}/pulls`
 - Issues: `GET /repos/{owner}/{repo}/issues`
- Aggregate per user, store in `GitHubContribution` table:

```
class GitHubContribution(models.Model):  
    user = models.ForeignKey(User, on_delete=models.CASCADE)  
    repo_name = models.CharField(max_length=255)  
    commit_count = models.IntegerField(default=0)  
    pull_request_count = models.IntegerField(default=0)  
    issues_count = models.IntegerField(default=0)  
    last_synced = models.DateTimeField(auto_now=True)
```

7. Sync Strategy

- Don't fetch contributions on every request — use **periodic sync**:
 - Celery task or cron job (daily/hourly)
 - Fetch new commits since `last_synced`

- Update `GitHubContribution` table
-

8. Frontend Integration

- Show:
 - GitHub connection status (Connected / Not Connected)
 - User contribution stats: commits, PRs, issues
 - Optional: contribution graphs (weekly/monthly)
-

Flow Diagram Summary

1. **Frontend:** User clicks “Connect GitHub”
2. **GitHub:** OAuth consent page → user authorizes
3. **Backend:** Receives code → exchanges for access token → stores token
4. **Backend:** Fetches repos created by the user → aggregates commits/PRs/issues
5. **Database:** Stores contribution data
6. **Frontend:** Displays analytics in dashboard