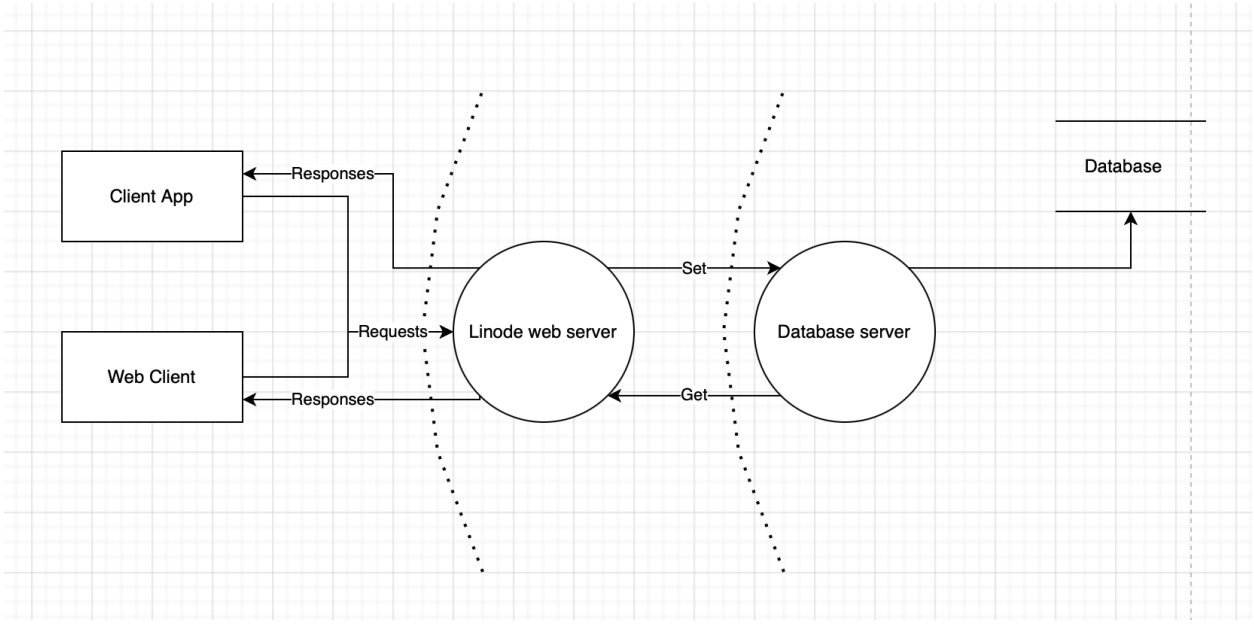
**Authors: Charlie Roslansky and Silas Rhyneer**

**Data flow diagram:**




**Threats:**

- **Spoofing**
    - A bad actor might get into the database server by impersonating your home computer or the web server. This could be fixed by requiring some sort of authentication when a client wants to access data.
    - A bad actor could impersonate another user. This could be fixed by requiring login credentials. They could also steal their username and password, which could be mitigated but not totally prevented by requiring (or strongly encouraging) 2FA.
    - A bad actor catfishing other users by posing to be them on the site. This could be fixed by requiring authentication (A user would have to send proof of their identity in order to register)
- **Tampering**
    - A bad actor could edit the database. This could be prevented by requiring login access for administrative control of the database. For a user to edit the database, their “get” or “set” request would have to come with some sort of validation key verifying that that user is authorized to edit that data.
    - Changing source code. Divide access to source code between developers, and require an authentication process for new code to be pushed to the production branch.
    - The linode server could alter requests or responses made. Requests and responses should be encrypted and hashed so that they cannot be altered by the server.
- **Repudiation**

    - Someone could pose as a user they aren't. This can be fixed by requiring login credentials.
    - Another website could pretend to be tapir.com. Getting a certificate for the site would help.
- **Information disclosure**
    - An eavesdropper might be listening in on the HTTP connection between the client apps and the web server. A mitigation might be to have this connection over HTTPS instead.
    - An eavesdropper could likewise be listening to the connection between the linode server and the database. This should be encrypted as well, since sensitive information such as passwords and other personal data are being transmitted.
    - Jeff has access to all users' passwords, credit card information, etc. This should be stored in a way such that Jeff cannot access the info, just authenticate (like using hashing to store passwords). Likewise, he should not be able to access their personal information, so that should be encrypted as well.
- **Denial of service**
    - The web server or database could be overloaded with spam requests. A possible mitigation is to backup all the data, and possibly to limit traffic between the web server and database server, to prevent the database from getting overloaded.
    - A web client could be spammed with responses from the server. This could be prevented by limiting the traffic from the server to an individual client (this might be more in the scope of the browser's security)
    - Someone might break into Jeff's home and destroy his computer/home network. Jeff should get better locks for his home.
    - Someone could break into Linode's server and destroy it. Linode should improve their physical security.
- **Elevation of privilege**
    - A user might attempt to edit the source code. This can be prevented by implementing strict permissions for users not to be able to edit any data besides their own.
    - User shouldn't be able to alter (or see) other people's data
    - Developers may read whatever other systems allow them to read, but only should be allowed to write within their department of code.