Based on the provided code, it seems like you're building a backend system for managing properties, users, rental agreements, and other related functionalities. Let's break down the components:

1. \*\*Backend:\*\*

- The majority of your code is focused on defining structs and functions related to managing properties, users, rental agreements, and canister state initialization and upgrade.

- You have structs like `User`, `Role`, `Property`, `PropertyListing`, `RentalAgreement`, `RentalContract`, `CanisterState`, etc., which represent the core data structures and entities in your system.

- Functions like `register\_user`, `create\_property\_listing`, `sign\_rental\_agreement`, etc., are used for interacting with these entities and performing actions such as user registration, property listing creation, and rental agreement signing.

2. \*\*Frontend:\*\*

- The second part of your code seems to be more focused on simulating user interactions and displaying information, particularly in a command-line interface (CLI) environment.

- Structs like `Buyer` and `Renter` represent users who are interacting with the system.

- Functions like `get\_user\_input` and `main` are involved in collecting user input and displaying information about available houses based on user interactions.

So, to classify it, the first part of your code primarily constitutes the backend logic for managing properties and related operations, while the second part simulates user interactions and serves as a frontend in a CLI environment. If you're planning to build a complete application, you might need to implement a separate frontend using web technologies like HTML, CSS, and JavaScript to interact with the backend APIs you've defined.

To link the provided backend code to a frontend built with web technologies, you'll need to create an API layer that exposes endpoints for the frontend to communicate with. Here's a general guide on how to achieve this:

1. \*\*Define API Endpoints:\*\*

- Identify the operations that your frontend needs to perform, such as user registration, property listing retrieval, rental agreement signing, etc.

- Design RESTful API endpoints that correspond to these operations. For example:

- `POST /api/users/register`: Register a new user.

- `GET /api/properties`: Retrieve a list of available properties.

- `POST /api/agreements`: Sign a rental agreement.

2. \*\*Implement API Handlers:\*\*

- Within your backend codebase, implement handlers for each API endpoint. These handlers will process incoming requests, interact with your existing backend logic, and return appropriate responses.

- Use a web framework like Rocket, Actix, or Warp (for Rust) to handle routing and request handling. These frameworks simplify the process of building web servers and handling HTTP requests.

- Integrate your existing backend logic (such as user registration, property management, etc.) with the API handlers.

3. \*\*Expose APIs:\*\*

- Start a web server that listens for incoming HTTP requests on the defined API endpoints.

- Ensure that your server is accessible to the frontend. If you're developing locally, you can use tools like ngrok to expose your local server to the internet for testing purposes.

4. \*\*Frontend Integration:\*\*

- In your frontend codebase (built with HTML, CSS, and JavaScript), use asynchronous HTTP requests (e.g., Fetch API or Axios) to interact with the backend APIs.

- When the user performs actions in the frontend (e.g., registering as a new user, viewing property listings, signing rental agreements), trigger the corresponding API requests to the backend.

- Process the responses from the backend APIs and update the frontend UI accordingly.

5. \*\*Cross-Origin Resource Sharing (CORS):\*\*

- If your frontend and backend are hosted on different domains, you may need to enable Cross-Origin Resource Sharing (CORS) on your backend server to allow the frontend to make requests to it.

- Configure CORS settings in your backend server to whitelist the domains from which requests are allowed.

6. \*\*Security Considerations:\*\*

- Implement authentication and authorization mechanisms to secure your APIs. This may involve using techniques like token-based authentication (e.g., JWT) and role-based access control (RBAC).

- Sanitize and validate user inputs to prevent security vulnerabilities such as injection attacks (e.g., SQL injection, XSS).

By following these steps, you can link your backend code to a frontend built with web technologies and enable communication between the two using APIs.