Documentación Proyecto

Gevora hotel

NRC: 2092

Equipo No. 05

Integrantes:

Camilo Andrés Castellanos Herrera

URL del proyecto

http://r4d3o.pythonanywhere.com/

https://github.com/camilo6castell/GEVORA-HOTEL-MisionTIC-poject

Descripción de roles del equipo (*Sprint 1*)

Rol	Integrante	Descripción	Tareas
Planeación	**Camilo Castellanos**	**40**	**FullStack**

Definición de artefactos (*Sprint 1*)

Backlog Sprint 1

User story	Descripción	Estimación (Horas)	Responsable
Desarrollo	**Camilo Castellanos**	**40**	**FullStack**

Backlog Sprint 2

User story	Descripción	Estimación (Horas)	Responsable
Diseño de aplicación - vistas	**Camilo Castellanos**	**40**	**FullStack**

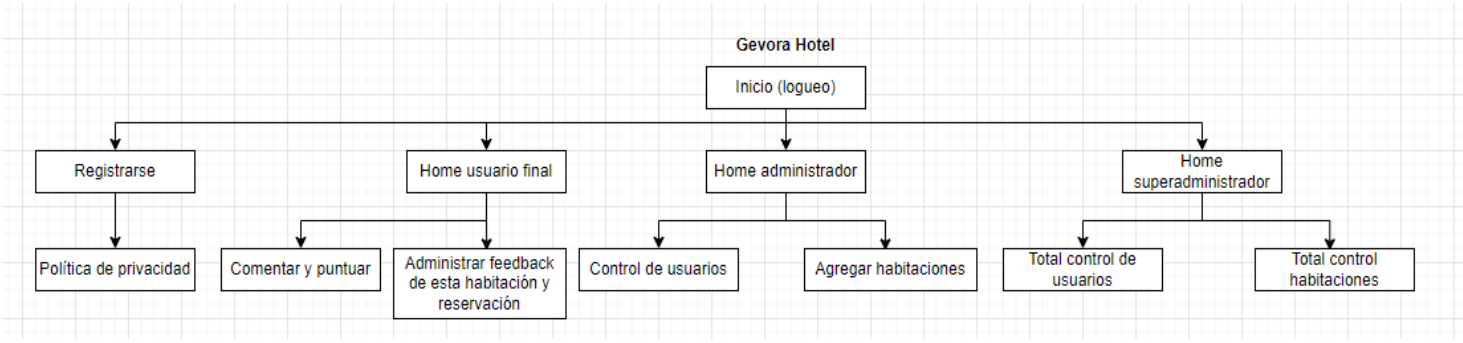
Backlog Sprint 3

User story	Descripción	Estimación (Horas)	Responsable
Elaboración de aplicación	**Camilo Castellanos**	**40**	**FullStack**

Backlog Sprint 4

User story	Descripción	Estimación (Horas)	Responsable
Despliegue	**Camilo Castellanos**	**20**	**FullStack**

Mapa de navegabilidad

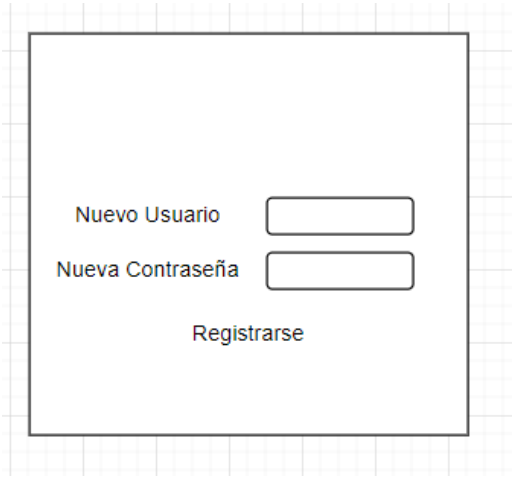


El mapa de navegabilidad muestra los diferentes caminos que existen para acceder a los servicios que ofrece. Los servicios disponibles dependerán del rol que el usuario tenga una vez ingrese al sistema y se identifique por el servidor.

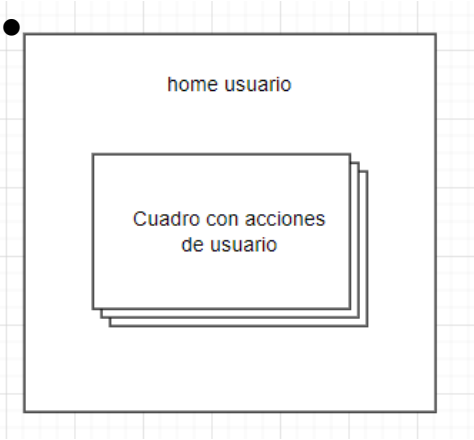
Vistas de la aplicación (*Sprint 2*)

- Inicio de aplicación

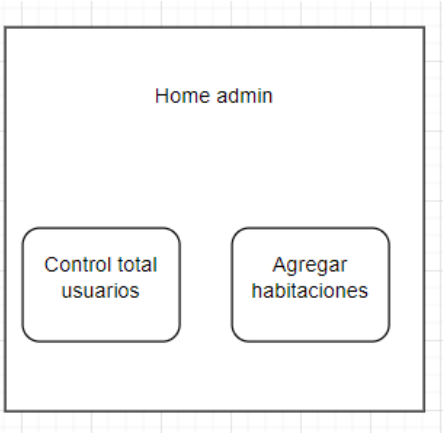
- Registrarse



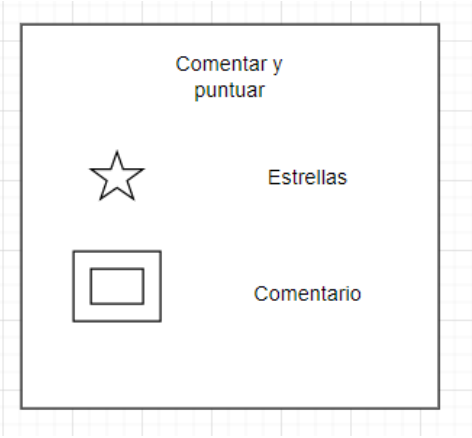
- **Home usuario**



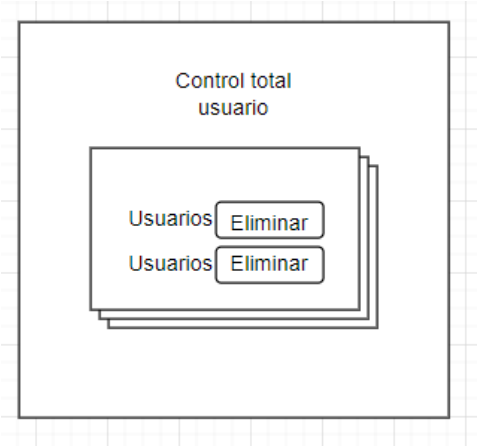
- **Home administrador**



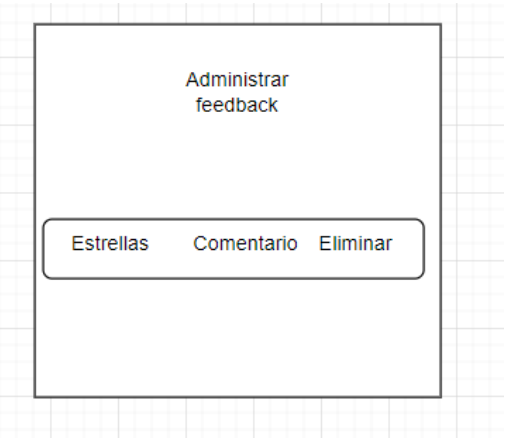
- **Comentar y puntuar**



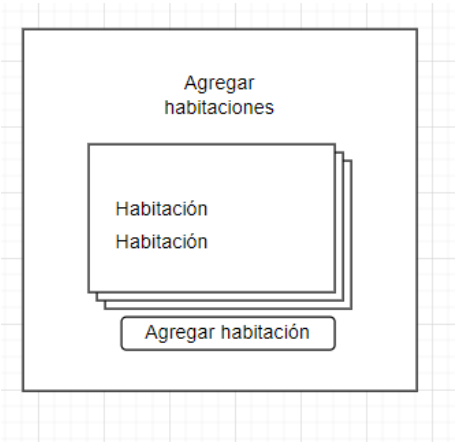
- **Control total de usuarios**



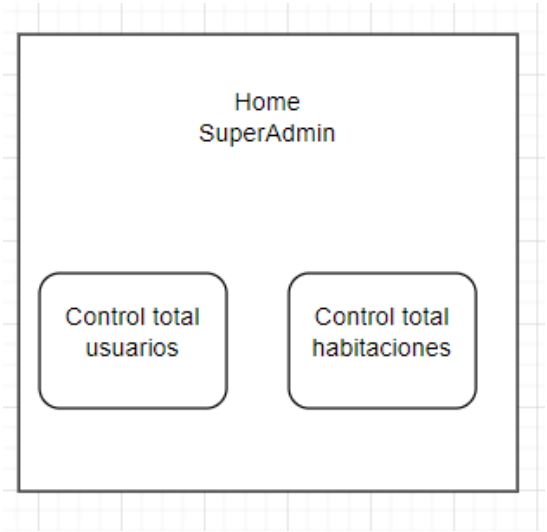
- **Administrar feedback**



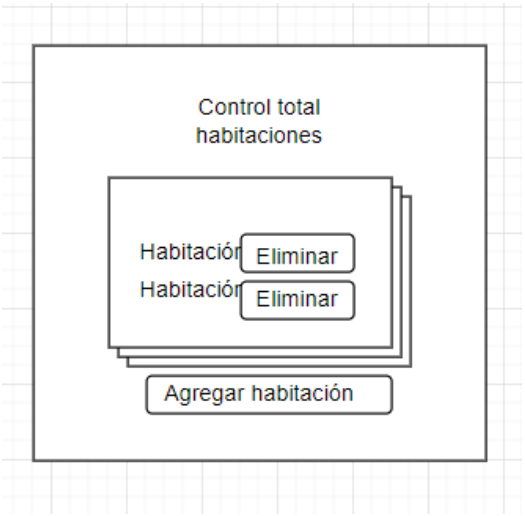
- **Agregar habitaciones**



- **Home superadministrador**



- **Control total habitaciones**



Métodos controladores del Back-End

- index()
  - Ruta: /
  - Métodos HTTP permitidos: GET
  - Lógica algorítmica:
    Devuelve la página de inicio-principal
- log_in()
  - Ruta: /
  - Métodos HTTP permitidos: POST
  - Lógica algorítmica:
    Captura información de formulario en index y la procesa. Seguidamente hace consulta y redirecciona a home de usuario final o home de admin o home de superadmin. Sino a vista de error o index si no se puede avanzar.
- registro()
  - Ruta: /registro/
  - Métodos HTTP permitidos: GET
  - Lógica algorítmica:
    Captura información de formulario en registro, procesa información y consulta en base de datos. Seguidamente devuelve a index para

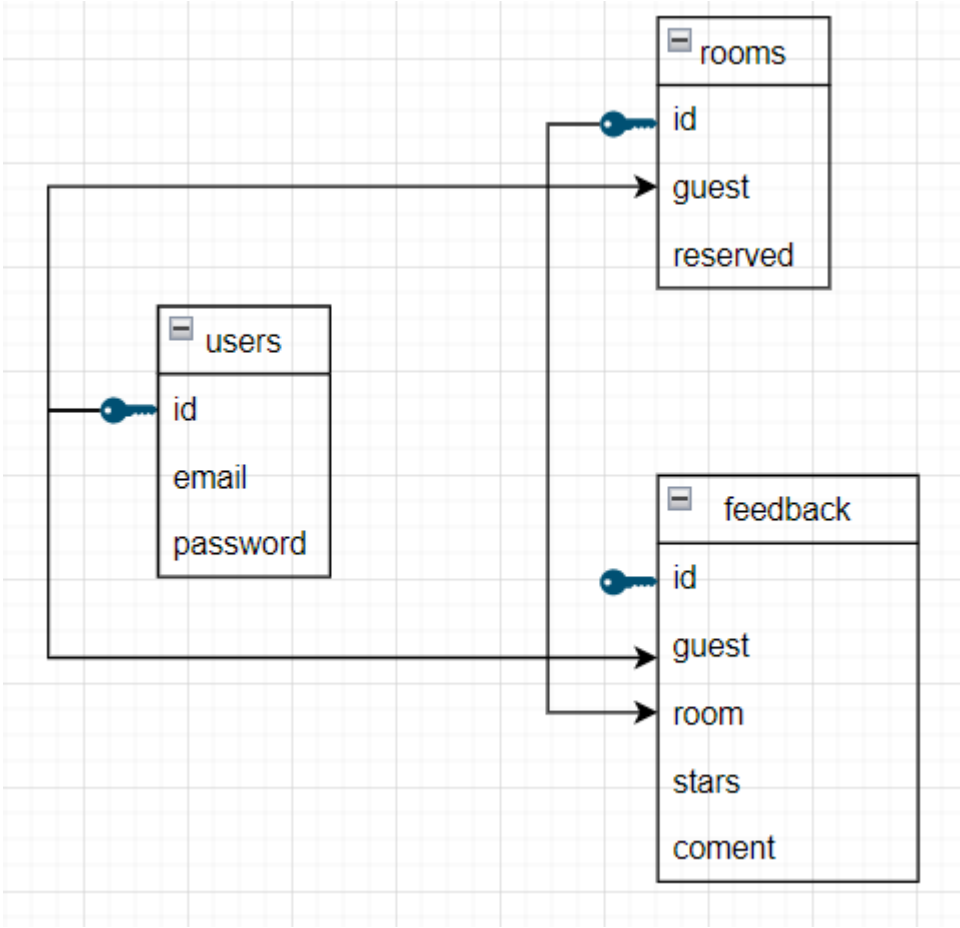
iniciar sesión en caso de ser usuario nuevo o un usuario ya registrado anteriormente.

- home()
    - Ruta: /home/
    - Métodos HTTP permitidos: GET, POST
    - Lógica algorítmica:
      Recibe información de usuario logueado y comienza con una consulta en la base de datos para poder tener la información lista para poder renderizar la página de home de usuario con toda la información necesaria.
- homesa(a)
    - Ruta: /homesa/<int:a>
    - Métodos HTTP permitidos: GET
    - Lógica algorítmica:
      Devuelve la página de home de Súperadministrador
- homea(a)
    - Ruta: /homea/<int:a>/
    - Métodos HTTP permitidos: GET
    - Lógica algorítmica:
      Devuelve la página de home de administrador
- controlu(a, b=0)
    - Ruta: /controlu/<int:a>/<int:b>/
    - Métodos HTTP permitidos: GET, POST
    - Lógica algorítmica:
      Devuelve la página de control total de usuarios. Con dos posibilidades de renderización, una inicial donde se carga la información actual según base de datos, y la segunda donde se ejecuta una eliminación de la base de datos de un usuario final y seguidamente devuelve a la opción uno donde consulta y actualiza la información para mostrar en la vista.
- controlh(a, b=0)
    - Ruta: /controlh/<int:a>/ - /controlh/<int:a>/<int:b>/
    - Métodos HTTP permitidos: GET, POST
    - Lógica algorítmica:

Consulta información para renderizar página con la información actual. Su segunda opción en donde se elimina un registro de habitaciones y se devuelve a la primera opción para renderizar información actual. Su tercera opción agrega habitación y devuelve a primer opción para renderizar información actual.

- feed_back(i,s=0,c=0)
    - Ruta: /feedback/<int:i>
    - Métodos HTTP permitidos: GET, POST
    - Lógica algorítmica:
      Recupera información del formulario y renderiza vista de feedback. En caso de haber cambios en el formulario actualiza la base de datos con una consulta. Devuelve un error de presentarse.
- editcom(i,c)
    - Ruta: /editcom/<int:i>/<int:c>/
    - Métodos HTTP permitidos: GET, POST
    - Lógica algorítmica:
      Comienza haciendo una consulta para mostrar información actual y renderizar, en las siguientes opciones se recupera información de los formularios y se hacen actualizaciones a la base de datos según se requiera y finalmente se devuelve a la primera opción para actualizar y renderizas información actual.
- log_out()
    - Ruta: /log_out/
    - Métodos HTTP permitidos: GET
    - Lógica algorítmica:
      Elimina la información en la cookie de sesión y devuelve a la pagina de inicio para volver a comenzar el proceso.

Base de datos



La base de datos se compone de 3 tablas: "users", la cual es la principal y contiene los usuarios finales y tiene como llave principal id, esta llave es una llave foránea en las tablas "rooms" y "feedback" en las cuales aparece como columna "guest". La tabla "rooms" tiene como llave principal "id" y esta llave aparece como llave foránea en la tabla "feedback" como room. En la tabla "feedback" la llave principal es "id".

Prácticas de programación segura (*Sprint 3*)

Mi proyecto cuenta con el uso de una "SECRET_KEY" administrada con un número aleatorio que prevalece en el tiempo y permite una seguridad más robusta al comunicarse con el cliente.

Además, dentro de mi proyecto importe la librería de markupsafe con la intención de usar su "escape" para proteger al servidor de las entradas recuperadas de los formularios de los HTML. También importe "re" y "validate_email" para validar los datos de email y contraseña desde el backend.

Una medida importante fue importar el "werkzeug.security" con sus funciones de "generate_password_hash" y "check_password_hash" para manejar de manera segura las contraseñas en mi servidor y almacenarlas en mi base de datos.

Con "session" y "g" de "flask" manejé de forma segura las sesiones de usuario final, administrador y super administrador en mi proyecto.

Para evitar la inyección de código SQL, implementé el uso de sentencias preparadas para toda vinculación realizada con la base de datos.

Despliegue

Para el despliegue de mi aplicación usé el servicios de plataforma como servicio (PaaS) que ofrece Pythoneverywhere. Para esto cree una cuenta gratuita, importé mi código, descargué e instalé módulos a python necesarios para que mi aplicación funcionara. Por ultimo revisé la funcionalidad y corregí errores como referencias a rutas de archivos y funciones en el código para que funcionaran óptimamente.

Revisé con detenimiento que todo estuviera perfecto. Por ultimo, complete este documento con la ruta y esata descripción del despliegue.

LINK: http://r4d3o.pythonanywhere.com/

DATOS DE ACCESO PARA PRUEBA

SúperAdmin
Usuario: Superadmin1@superadmin1.com
Contraseña: Superadmin1@superadmin1.com

Admin
Usuario: Admin1@admin1.com
Contraseña: Admin1@admin1.com

Usuario final
Usuario: Usuario5@usuario5.com
Contraseña: Usuario5@usuario5.com