Unitests

Reporte de Pruebas Unitarias

Este documento describe el setup, herramientas utilizadas y organización de las pruebas unitarias del proyecto.

---

1. Herramientas y Librerías

- **Vitest**: Framework de pruebas
- **Effect**: Para inyeccion de deppendencias (`Layer`) y manejor de errores
- **Drizzle**: Orm
- **vitest mocks**: Para mockear modulos externos en frontend y mockear callbacks en los componentes

---

2. Backend: Setup y Organización

Ubicación de las pruebas backend

Las pruebas del backend se encuentran en: `src/tests/`

Archivos actuales:

- `src/tests/user.test.ts`
- `src/tests/place.test.ts`
- `src/tests/util.ts`

Patrón de configuración

Cada prueba backend sigue el mismo patron:

1. **Composición de capas**: Se construye una `Layer` que provee implementaciones de repositorios, servicios, etc .

2. **Definición del programa**: Se usa `Effect.gen` definir las operaciones (crear usuario, crear lugar, consultar, actualizar, etc).
3. **Ejecución**: `Effect.runPromise(program.pipe(Effect.provide(testLayer)))` aplica el `Layer` de prueba y ejecuta el programa.
4. **Asserts**: Se hacen las validaciones

Pruebas Backend

- user.test.ts
  - Qué hace: verifica que un nuevo usuario se crea con el rol y email correctos en la base de datos.
  - Casos límite: validación de rol asignado
  - Código:

```
test("should create a new todo with correct default values",
async () => {
  const program = Effect.gen(function* () {
    const newUser = yield* UserRepository.create({
      clerk_id: "clerk-fake-id",
      email: "fake@email.com",
      fullName: "Fake User",
      role: "explorer"
    })
    return newUser
  })
  const testLayer =
Layer.provide(UserRepository.DefaultWithoutDependencies, DBTest)
  const result = await
Effect.runPromise(program.pipe(Effect.provide(testLayer)))
  expect(result.role).toBe("explorer")
})
```

- place.test.ts (crear lugar)
  - Qué hace: verifica que un lugar se crea correctamente asociado a un business y mantiene la relación de clave foránea.
  - Casos límite: relación business_id → place válida
  - Código:

```
test("creates a place for a business user", async () => {
  const program = Effect.gen(function* () {
    const business = yield* createBusinessUser("create")
    const place = yield* PlaceRepository.create({ business_id:
business.id, name: "Test Place", description: "Great place",
location: "https://maps.example.com/test" })
```

```
    return { business, place }
  })
  const { place, business } = await
Effect.runPromise(program.pipe(Effect.provide(testLayer)))
  expect(place.business_id).toBe(business.id)
})
```

- place.test.ts (filtrado por negocio)
  - Qué hace: verifica que la búsqueda por business_id retorna solo los lugares asociados a ese negocio, ignorando los de otros.
  - Casos límite: múltiples businesses con lugares propios
  - Código:

```
test("findByBusinessId returns only places for the given
business", async () => {
  const program = Effect.gen(function* () {
    const businessOne = yield* createBusinessUser("one")
    const businessTwo = yield* createBusinessUser("two")
    yield* PlaceRepository.create({ business_id: businessOne.id,
name: "Cafe Uno", description: "Coffee", location: "..." })
    yield* PlaceRepository.create({ business_id: businessOne.id,
name: "Cafe Dos", description: "Coffee2", location: "..." })
    yield* PlaceRepository.create({ business_id: businessTwo.id,
name: "Cafe Tres", description: "Coffee3", location: "..." })
    const placesForBusinessOne = yield*
PlaceRepository.findByBusinessId(businessOne.id)
    return { placesForBusinessOne }
  })
  const { placesForBusinessOne } = await
Effect.runPromise(program.pipe(Effect.provide(testLayer)))
  expect(placesForBusinessOne).toHaveLength(2)
})
```

- place.test.ts (update parcial)
  - Qué hace: comprueba que la actualización modifica solo los campos enviados sin afectar el id ni las relaciones existentes.
  - Casos límite: actualización parcial sin sobrescribir otros datos
  - Código:

```
test("update modifies provided fields", async () => {
  const program = Effect.gen(function* () {
    const business = yield* createBusinessUser("update")
    const place = yield* PlaceRepository.create({ business_id:
business.id, name: "Old Name", description: "Old", location:
"..." })
    const updated = yield* PlaceRepository.update(place.id, {
```

```
name: "New Name", description: "Updated" })
    return { updated }
  })
  const { updated } = await
Effect.runPromise(program.pipe(Effect.provide(testLayer)))
  expect(updated.name).toBe("New Name")
})
```

- place.test.ts (delete)
  - Qué hace: verifica que la eliminación remueve el registro de la base de datos y una búsqueda posterior retorna nulo.
  - Casos límite: registro eliminado no existe en consultas posteriores
  - Código:

```
test("delete removes a place and returns the deleted record",
async () => {
  const program = Effect.gen(function* () {
    const business = yield* createBusinessUser("delete")
    const place = yield* PlaceRepository.create({ business_id:
business.id, name: "To Be Deleted", description: "Temp",
location: "..." })
    const deleted = yield* PlaceRepository.delete(place.id)
    const afterDelete = yield* PlaceRepository.findById(place.id)
    return { deleted, afterDelete }
  })
  const { deleted, afterDelete } = await
Effect.runPromise(program.pipe(Effect.provide(testLayer)))
  expect(afterDelete).toBeNull()
})
```

Pruebas Frontend

Lista de pruebas (frontend):

- ExplorerHeader.test.tsx (ruta saved)
  - Qué hace: verifica que el componente muestra el título "Saved" cuando la ruta es `/explorer/saved/*`.
  - Casos límite: rutas anidadas en la sección saved

```
it("handles nested saved paths and shows Saved for
`/explorer/saved/*`", () => {
  setPathname("/explorer/saved/abc123"); setSignedIn(true);
render(<ExplorerHeader />);
```

```
  expect(screen.getByText("Saved")).toBeDefined();
})
```

- ExplorerHeader.test.tsx (trailing slash)
  - Qué hace: comprueba que una ruta con trailing slash `/explorer/` se interpreta como "Explorer" y no como "Discover".
  - Casos límite: rutas con slash final

```
it("treats '/explorer/' (trailing slash) as Explorer (not Discover)",
() => {
  setPathname("/explorer/"); setSignedIn(false);
render(<ExplorerHeader />);
  expect(screen.getByText("Explorer")).toBeDefined();
})
```

- ExplorerHeader.test.tsx (rerender dinámico)
  - Qué hace: verifica que el título se actualiza cuando la ruta cambia
  - Casos límite: cambios de ruta sin re renderizar el componente

```
it("updates title when pathname changes without remount (rerender)",
() => {
  setPathname("/explorer/saved/abc123"); const { rerender } =
render(<ExplorerHeader />);
  setPathname("/explorer/preferences/account");
rerender(<ExplorerHeader />);
  expect(screen.getByText("Preferences")).toBeDefined();
})
```

- SwipeDeck.test.tsx (guardar y avanzar)
  - Qué hace: verifica que al hacer click en "Save" se ejecuta el callback onSave y se avanza a la siguiente tarjeta.
  - Casos límite: transición tras guardar un lugar

```
it("renders the first place and advances after saving with the
button", async () => {
  const places = createPlaces(); const onSave = vi.fn(); const
onDiscard = vi.fn();
  render(<SwipeDeck places={places} onSave={onSave} onDiscard=
{onDiscard} />);
  fireEvent.click(screen.getAllByLabelText("Save")[0]);
  expect(onSave).toHaveBeenCalledTimes(1);
  expect(onSave).toHaveBeenCalledWith(places[0]);
})
```

- SwipeDeck.test.tsx (descartar y avanzar)

- Qué hace: asegura que al presionar "Discard" se llama a `onDiscard` con el lugar activo y el deck pasa al siguiente elemento.
- Casos límite: transición tras descartar sin disparar `onSave`

```
it("renders the first place and advances after discarding with the
button", async () => {
  const places = createPlaces(); const onSave = vi.fn(); const
onDiscard = vi.fn();
  render(<SwipeDeck places={places} onSave={onSave} onDiscard=
{onDiscard} />);
  fireEvent.click(screen.getAllByLabelText("Discard")[0]);
  expect(onDiscard).toHaveBeenCalledTimes(1);
  expect(onDiscard).toHaveBeenCalledWith(places[0]);
  await screen.findAllByText(places[1].name);
  expect(onSave).not.toHaveBeenCalled();
})
```

- SwipeDeck.test.tsx (lista vacía)
  - Qué hace: verifica que se muestra un mensaje de finalización cuando la lista de lugares está vacía.
  - Casos límite: renderizado con datos vacíos

```
it("renders completion message immediately when places is empty", ()
=> {
  render(<SwipeDeck places={[]} onSave={vi.fn()} onDiscard={vi.fn()}
/>);
  expect(screen.getAllByText("You're all caught up")
[0]).toBeDefined();
})
```

Ejecutar las pruebas

- Todas las pruebas (backend + frontend): `pnpm run test`
- Un solo archivo: `pnpm vitest run <runta_del_test>
- Un test case especifico: `pnpm vitest run <ruta_del_test> -t "<nombre del test>"`