En la arquitectura hexagonal, los **out ports** son interfaces que define el dominio para expresar **qué necesita hacer hacia afuera** sin acoplarse a la implementación. No importa si el flujo es *guardar datos* o *solo leer datos*, siempre que el dominio requiera interactuar con infraestructura externa, pasa por un out port.

Función principal

- Son **contratos de salida** que el dominio usa para pedir operaciones a un servicio externo (DB, API, cola de mensajes, etc.).
- El dominio **no sabe ni le importa** si la información proviene de MySQL, MongoDB o un archivo. Solo conoce el port.
- Los adaptadores de salida (por ejemplo, repositorios con Prisma) implementan estos ports.

¿Por qué incluso `get` necesita un out port?

Aunque `get` no "mande" información al exterior, **sí la solicita desde fuera del dominio** (base de datos). En hexagonal, leer y escribir son interacciones externas, por lo que ambas pasan por un port out.

Ejemplo:

```ts
// domain/colaboradores/colaboradores.repository.ts
import { Colaborador } from './colaboradores.entity';

export interface ColaboradorRepository {
  create(colaborador: Partial<Colaborador>): Promise<Colaborador>;
  findById(id: number): Promise<Colaborador | null>; // <- GET
  update(id: number, data: Partial<Colaborador>): Promise<Colaborador>;
  delete(id: number): Promise<void>;
}
```

El caso de uso llama al port:

```ts
const colaborador = await this.colaboradorRepo.findById(id);
```

Y el adaptador de salida lo implementa usando Prisma:

```ts
@Injectable()
export class PrismaColaboradorRepository implements ColaboradorRepository {
  constructor(private readonly prisma: PrismaService) {}
```

```
  async findById(id: number): Promise<Colaborador | null> {
    return this.prisma.colaborador.findUnique({ where: { id } });
  }
}
```

Resumen

- **Ports out** → definen lo que el dominio necesita de afuera.
- Incluyen **lecturas** (`get`) y **escrituras** (`create`, `update`, `delete`).
- Permiten cambiar la implementación sin tocar el dominio.
- Facilitan pruebas unitarias, ya que puedes **mockear** la implementación sin base de datos real.