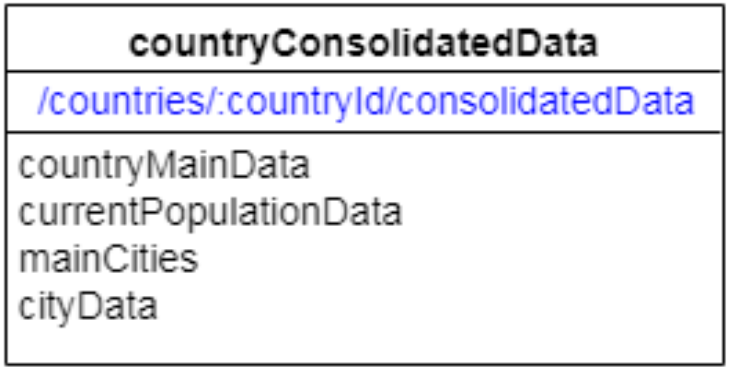
Microservicios – relacionamos conceptos con ejemplos

Carlos Lombardi

Universidad Nacional de Quilmes – Argentina

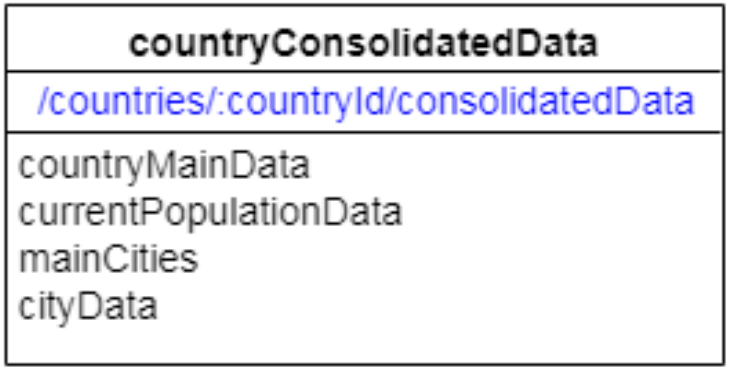
Quebrando el monolito – un poco

Antes

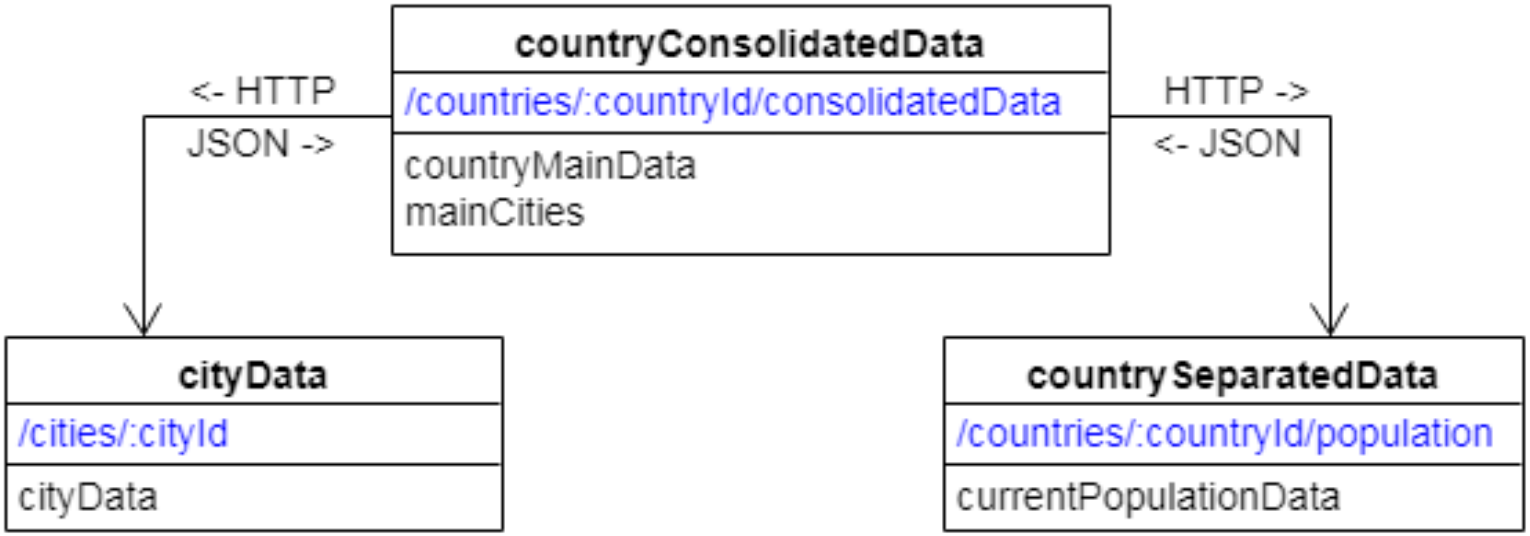


Quebrando el monolito – un poco

Antes



Después



Pautas – separación entre componentes

De qué se trata

Separar la aplicación en componentes, que se comunican solamente mediante servicios.

Fortalezas

- Componentes más manejables.
- Interfaces bien definidas entre componentes.
- Habilita descentralización del equipo de trabajo.

Debilidades/riesgos

- Cómo "cortar" una aplicación en componentes.
- Esfuerzo adicional para definir y desarrollar servicios.

Pautas – separación entre componentes

De qué se trata

Separar la aplicación en componentes, que se comunican solamente mediante servicios.

Fortalezas

- Componentes más manejables.
- Interfaces bien definidas entre componentes.
- Habilita descentralización del equipo de trabajo.

Debilidades/riesgos

- Cómo "cortar" una aplicación en componentes.
- Esfuerzo adicional para definir y desarrollar servicios.

Pautas – separación entre componentes

De qué se trata

Separar la aplicación en componentes, que se comunican solamente mediante servicios.

Fortalezas

- Componentes más manejables.
- Interfaces bien definidas entre componentes.
- Habilita descentralización del equipo de trabajo.

Debilidades/riesgos

- Cómo "cortar" una aplicación en componentes.
- Esfuerzo adicional para definir y desarrollar servicios.

Pautas – separación entre componentes

De qué se trata

Separar la aplicación en componentes, que se comunican solamente mediante servicios.

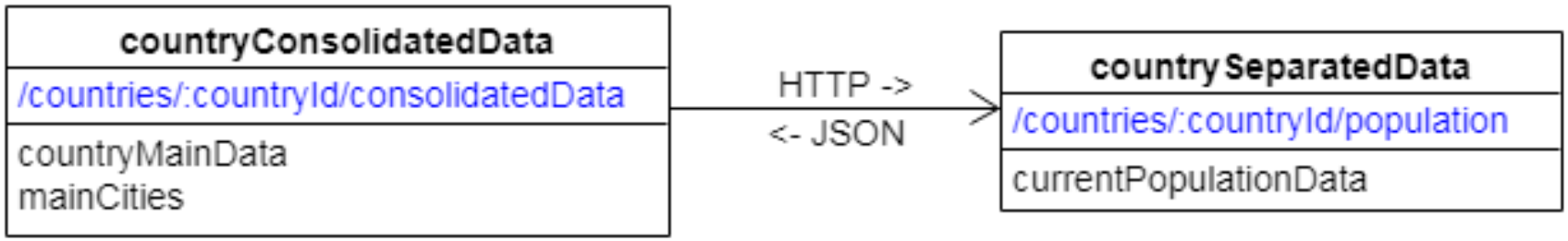
Fortalezas

- Componentes más manejables.
- Interfaces bien definidas entre componentes.
- Habilita descentralización del equipo de trabajo.

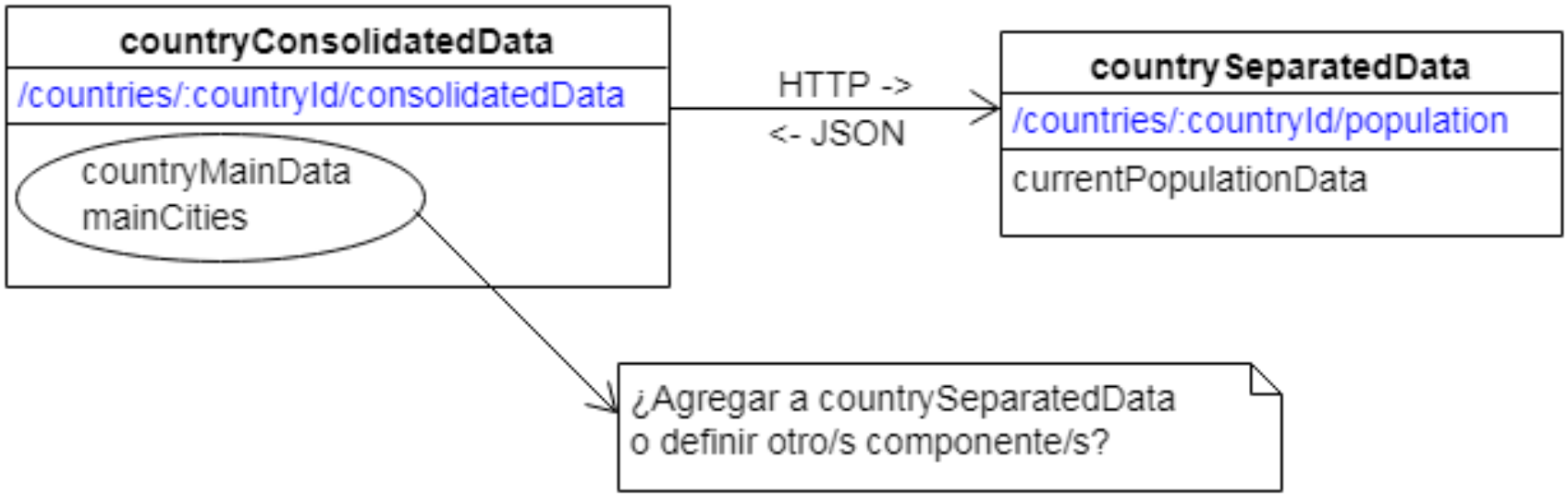
Debilidades/riesgos

- Cómo "cortar" una aplicación en componentes.
- Esfuerzo adicional para definir y desarrollar servicios.

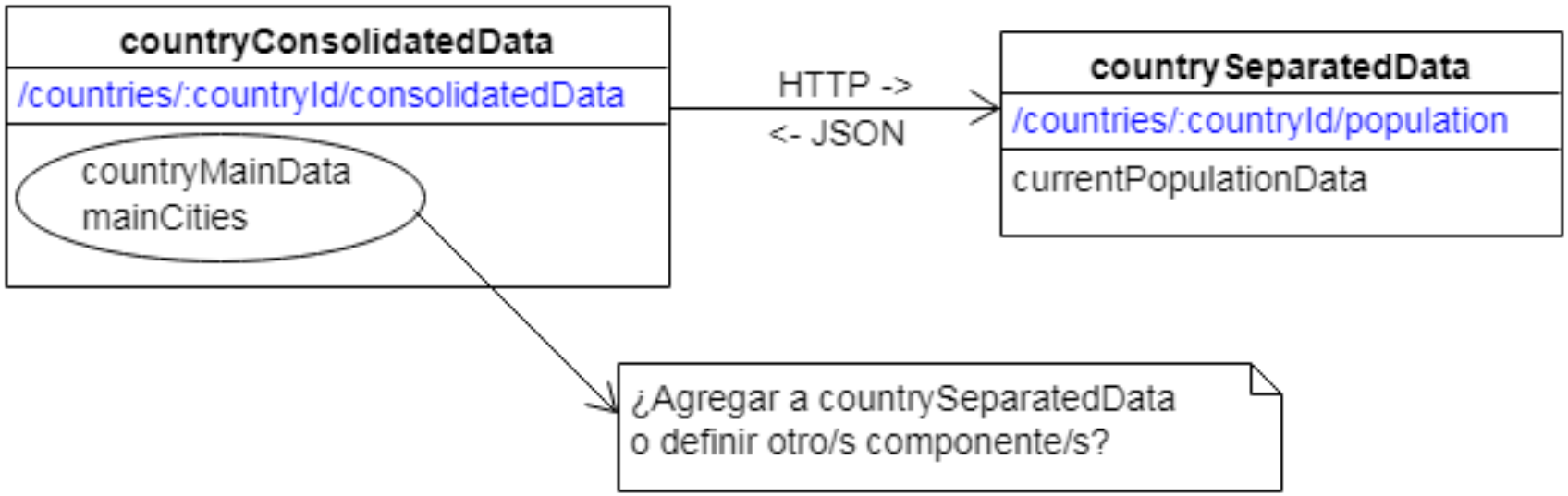
Quebrando el monolito – sigamos



Quebrando el monolito – sigamos

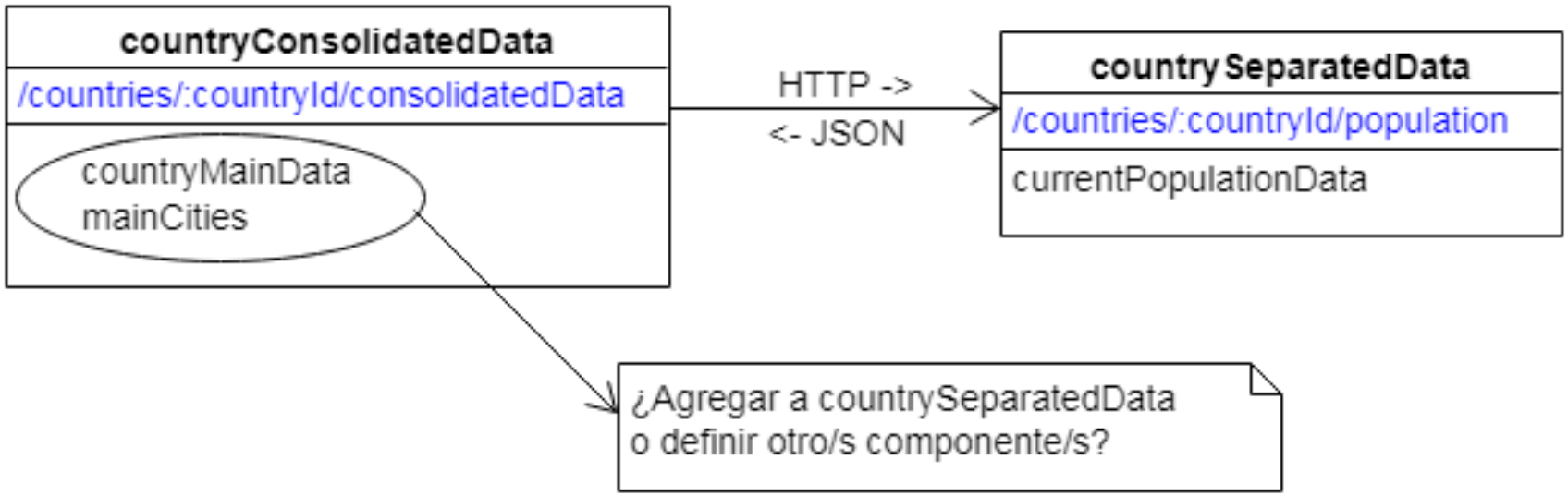


Quebrando el monolito – sigamos



Datos de países, ¿juntos o separados? Criterio:

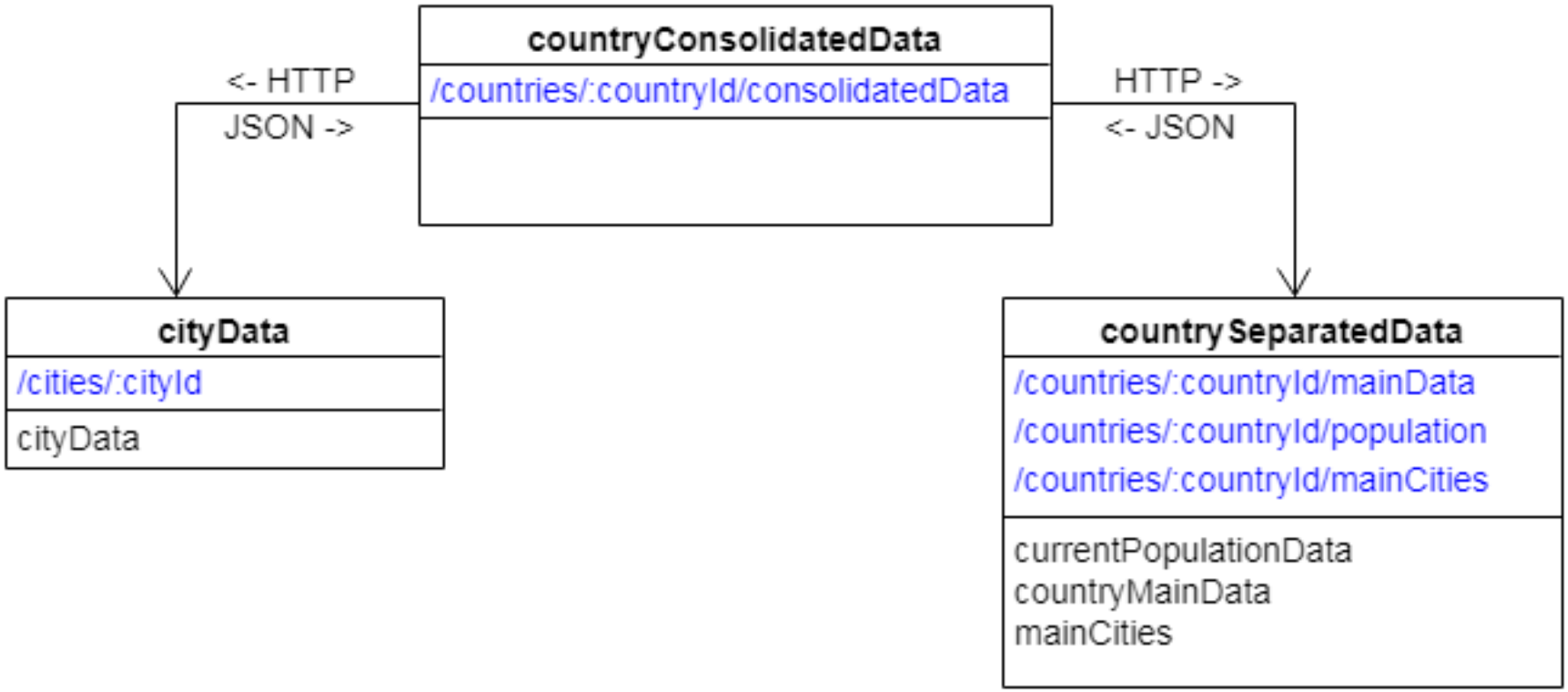
Quebrando el monolito – sigamos



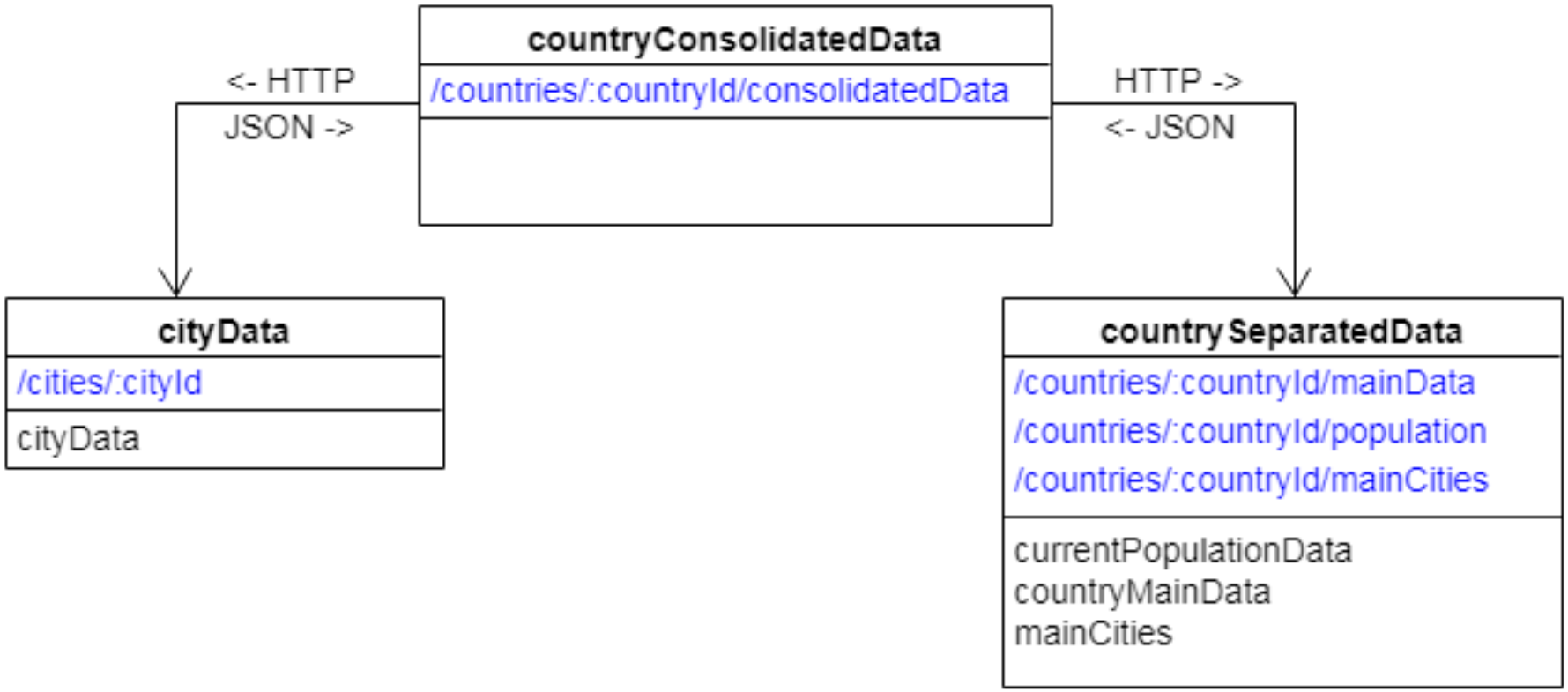
Datos de países, ¿juntos o separados? Criterio: evolución.

- Poner junto lo que cambia junto.
- Separar lo que puede escalar por separado.
- Separar lo que puede pasar a otro equipo.

Quebrando el monolito – componente de datos de países

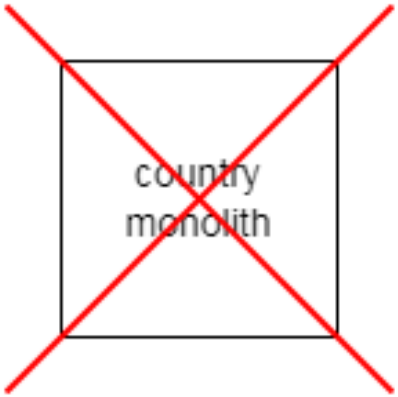


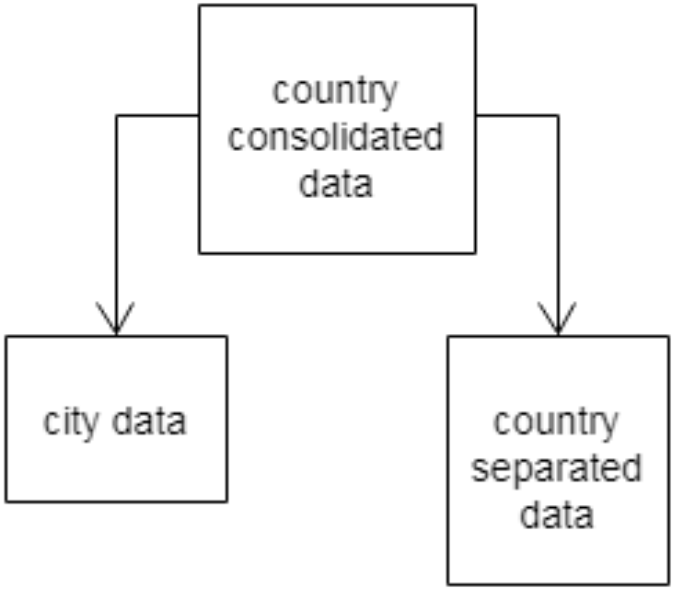
Quebrando el monolito – componente de datos de países



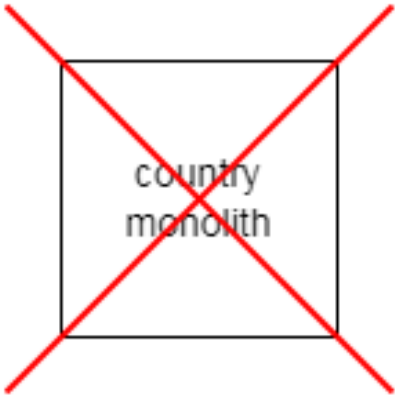
. . . veamos el código . . .

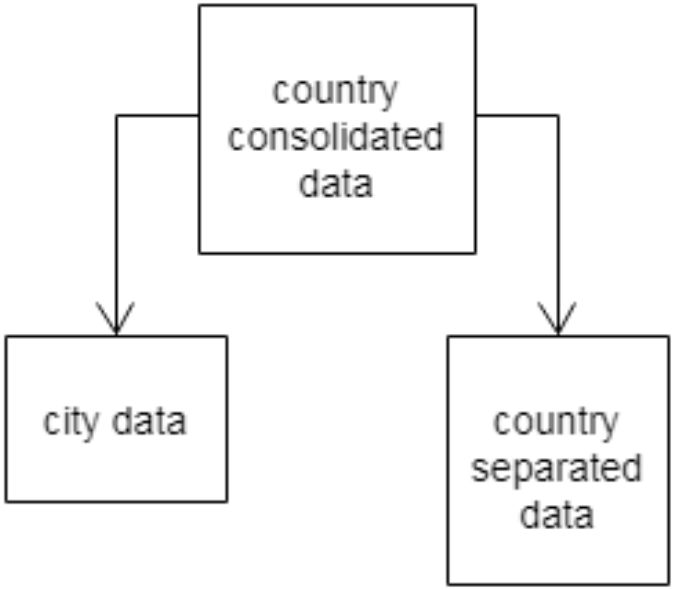
Repasamos las pautas





Repasamos las pautas





Separación entre componentes.

Deploys ágiles, descentralizados, automáticos.

Interfaces livianas.

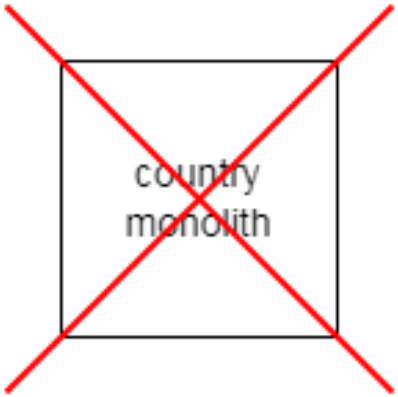
Organización del equipo por funcionalidad.

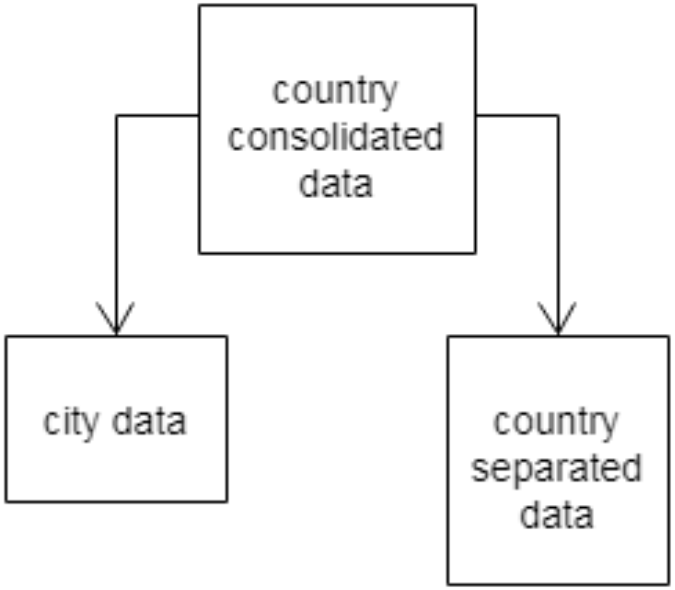
Gestión descentralizada.

Datos descentralizados.

Design for failure.

Repasamos las pautas





Separación entre componentes.

Deploys ágiles, descentralizados, automáticos.

Interfaces livianas.

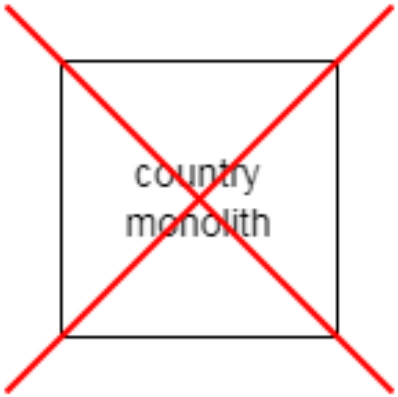
Organización del equipo por funcionalidad.

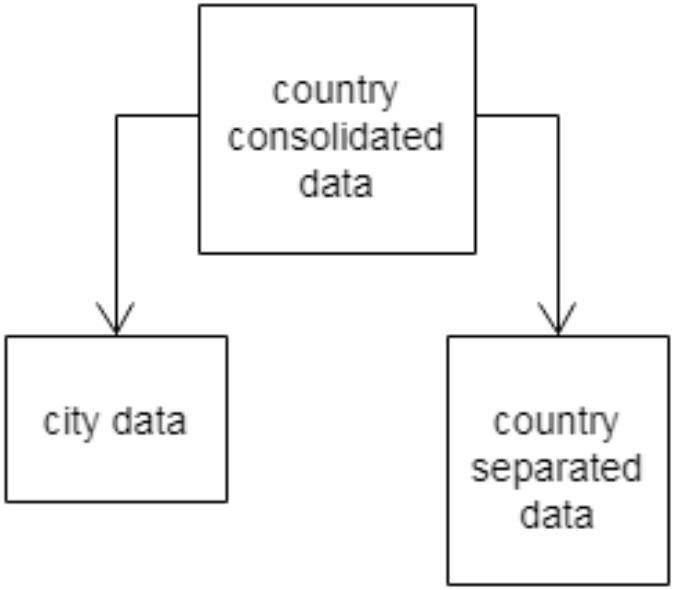
Gestión descentralizada.

Datos descentralizados.

Design for failure.

Repasamos las pautas





Separación entre componentes.

Deploys ágiles, descentralizados, automáticos.

Interfaces livianas.

Organización del equipo por funcionalidad.

Gestión descentralizada.

Datos descentralizados.

Design for failure.

La aplicación evoluciona - corregimos un bug

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

La aplicación evoluciona - corregimos un bug

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Encontramos un bug en un servicio.

La aplicación evoluciona - corregimos un bug

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Encontramos un bug en un servicio.

La aplicación evoluciona – se agrega funcionalidad

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

La aplicación evoluciona – se agrega funcionalidad

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Tours

Incorporamos funcionalidad nueva que usa servicios existentes.

Tour – información consolidada

```
{ "name": "Sudamérica full", "regularPrice": 2400, ...
  "journey": [
    { "cityId": 2002, "cityName": "Rio de Janeiro" ,
      "firstDay": 1, "lastDay": 4, ...  },
    { "cityId": 1001, "cityName": "Buenos Aires",
      "firstDay": 4, "lastDay": 6, ...  }, ...  ],
  "countries": [
    { "countryId": 2, "countryName": "Brazil",
      "dayCount": 4, "cityCount": 1, ...  },
    { "countryId": 1, "countryName": "Argentina",
      "dayCount": 9, "cityCount": 4, ...  }, ...  ]
}
```

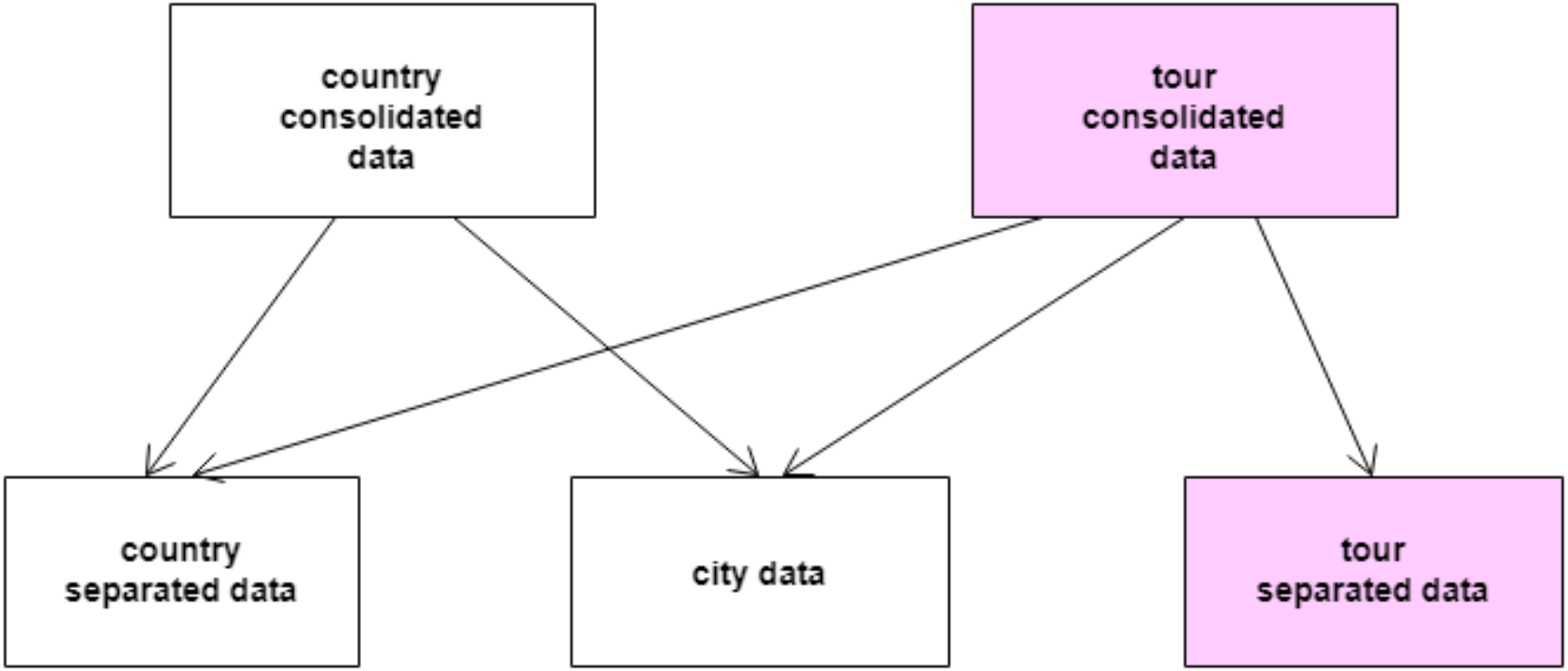
Tour – información consolidada

```
{ "name": "Sudamérica full", "regularPrice": 2400, ...
  "journey": [
    { "cityId": 2002, "cityName": "Rio de Janeiro" ,
      "firstDay": 1, "lastDay": 4, ... },
    { "cityId": 1001, "cityName": "Buenos Aires",
      "firstDay": 4, "lastDay": 6, ... }, ... ],
  "countries": [
    { "countryId": 2, "countryName": "Brazil",
      "dayCount": 4, "cityCount": 1, ... },
    { "countryId": 1, "countryName": "Argentina",
      "dayCount": 9, "cityCount": 4, ... }, ... ]
}
```

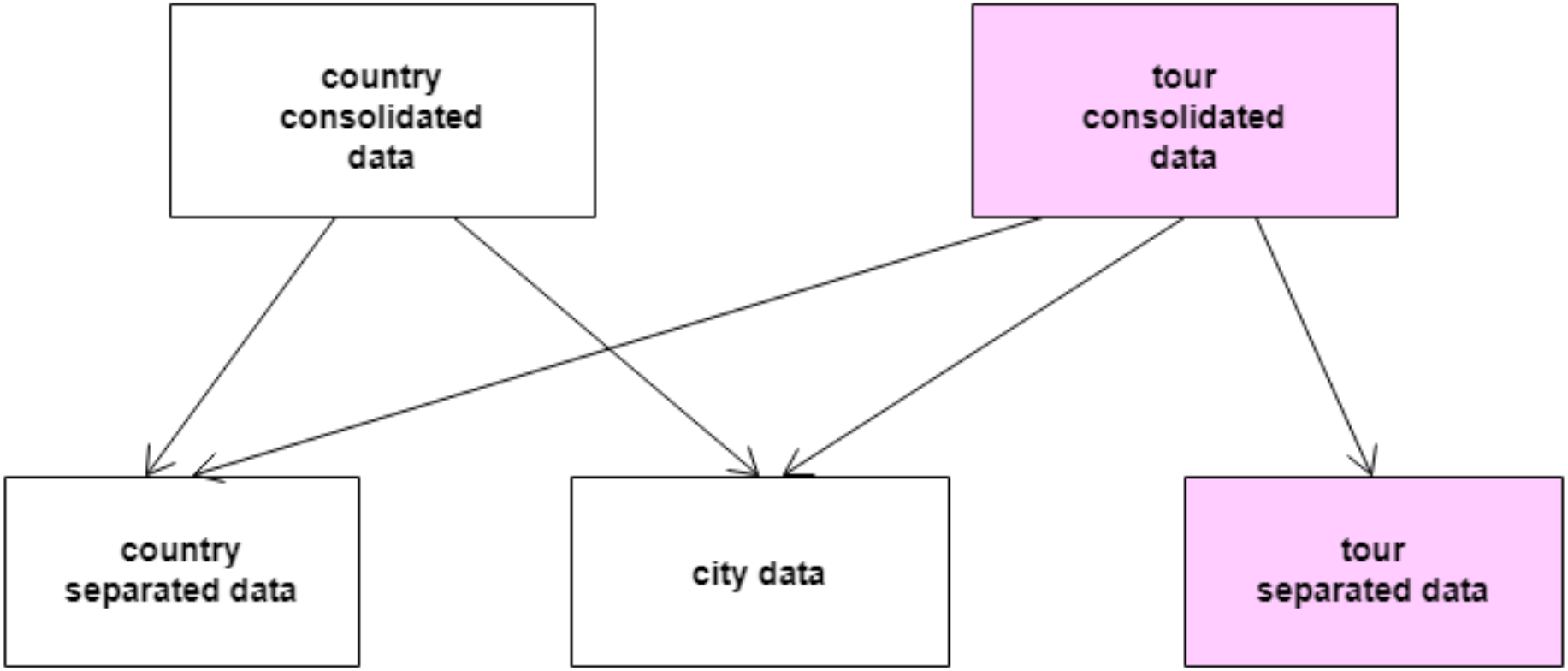
Tour – información consolidada

```
{ "name": "Sudamérica full", "regularPrice": 2400, ...
  "journey": [
    { "cityId": 2002, "cityName": "Rio de Janeiro" ,
      "firstDay": 1, "lastDay": 4, ... },
    { "cityId": 1001, "cityName": "Buenos Aires",
      "firstDay": 4, "lastDay": 6, ... }, ... ],
  "countries": [
    { "countryId": 2, "countryName": "Brazil",
      "dayCount": 4, "cityCount": 1, ... },
    { "countryId": 1, "countryName": "Argentina",
      "dayCount": 9, "cityCount": 4, ... }, ... ]
}
```

Tours – esquema de servicios



Tours – esquema de servicios



. . . veamoslo andando . . .

Tours – repasemos pautas

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Tours – repasemos pautas

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Tours

Incorporamos funcionalidad nueva que usa servicios existentes.

Tours – repasemos pautas

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Tours

Incorporamos funcionalidad nueva que usa servicios existentes.

Tours – repasemos pautas

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Tours

Incorporamos funcionalidad nueva que usa servicios existentes.

Tours – repasemos pautas

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Tours

Incorporamos funcionalidad nueva que usa servicios existentes.

La aplicación evoluciona – hay que escalar un servicio

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

La aplicación evoluciona – hay que escalar un servicio

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

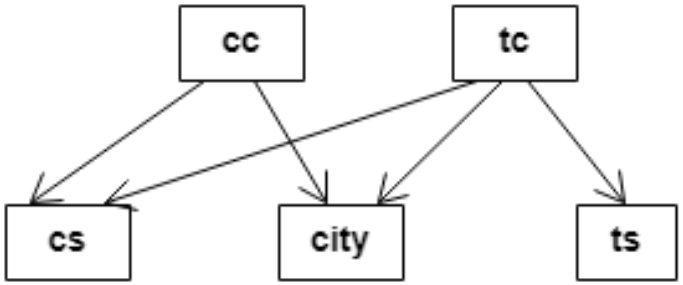
Se observa que el servicio `cities/:cityId` crece en tráfico.

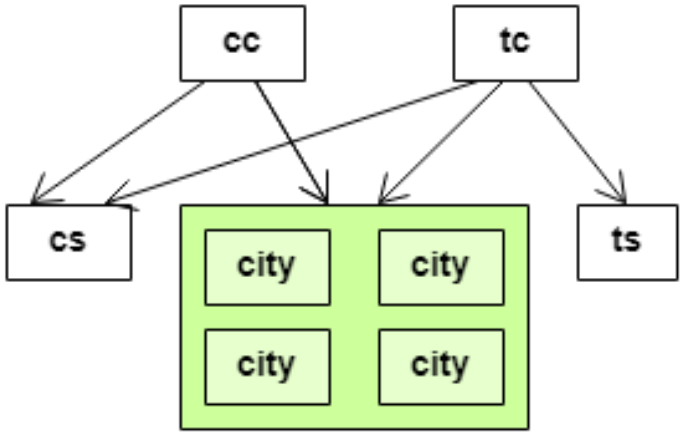
La aplicación evoluciona – hay que escalar un servicio

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Se observa que el servicio `cities/:cityId` crece en tráfico.

Se decide escalarlo, se levantan 4 instancias.



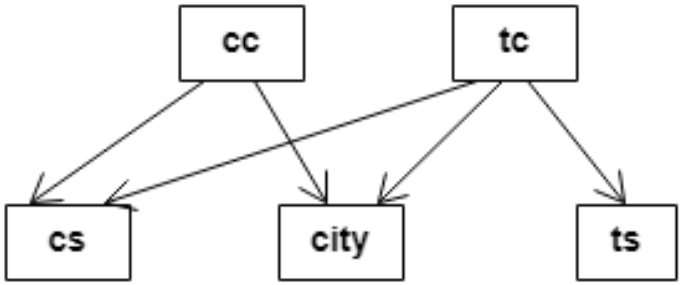


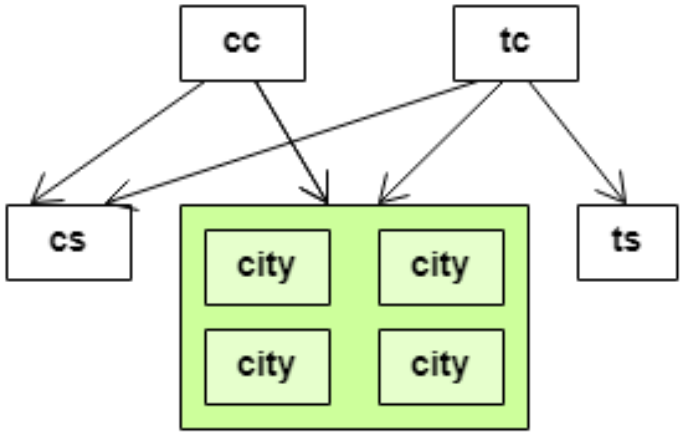
La aplicación evoluciona – hay que escalar un servicio

- Separación entre componentes.
- Deploys ágiles, descentralizados, automáticos.
- Interfaces livianas.

Se observa que el servicio `cities/:cityId` crece en tráfico.

Se decide escalarlo, se levantan 4 instancias.





La aplicación evoluciona – crece `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

La aplicación evoluciona – crece `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

Se agregan más requerimientos de información sobre ciudades:
`cities/:cityId/attractions` `cities/:cityId/hotels`
etc.

La aplicación evoluciona – crece `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

Se agregan más requerimientos de información sobre ciudades:
`cities/:cityId/attractions` `cities/:cityId/hotels`
etc.

- Se separa el equipo de desarrollo de `/cities`.

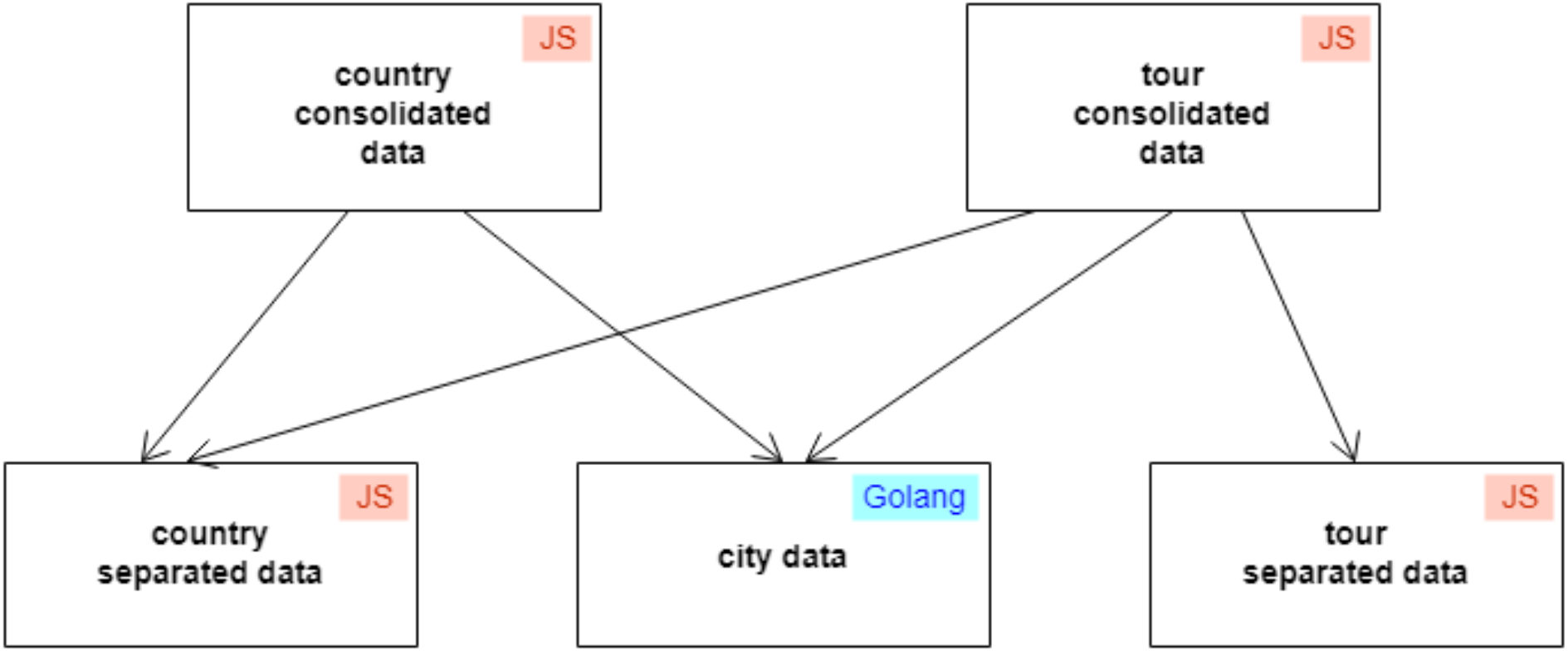
La aplicación evoluciona – crece `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

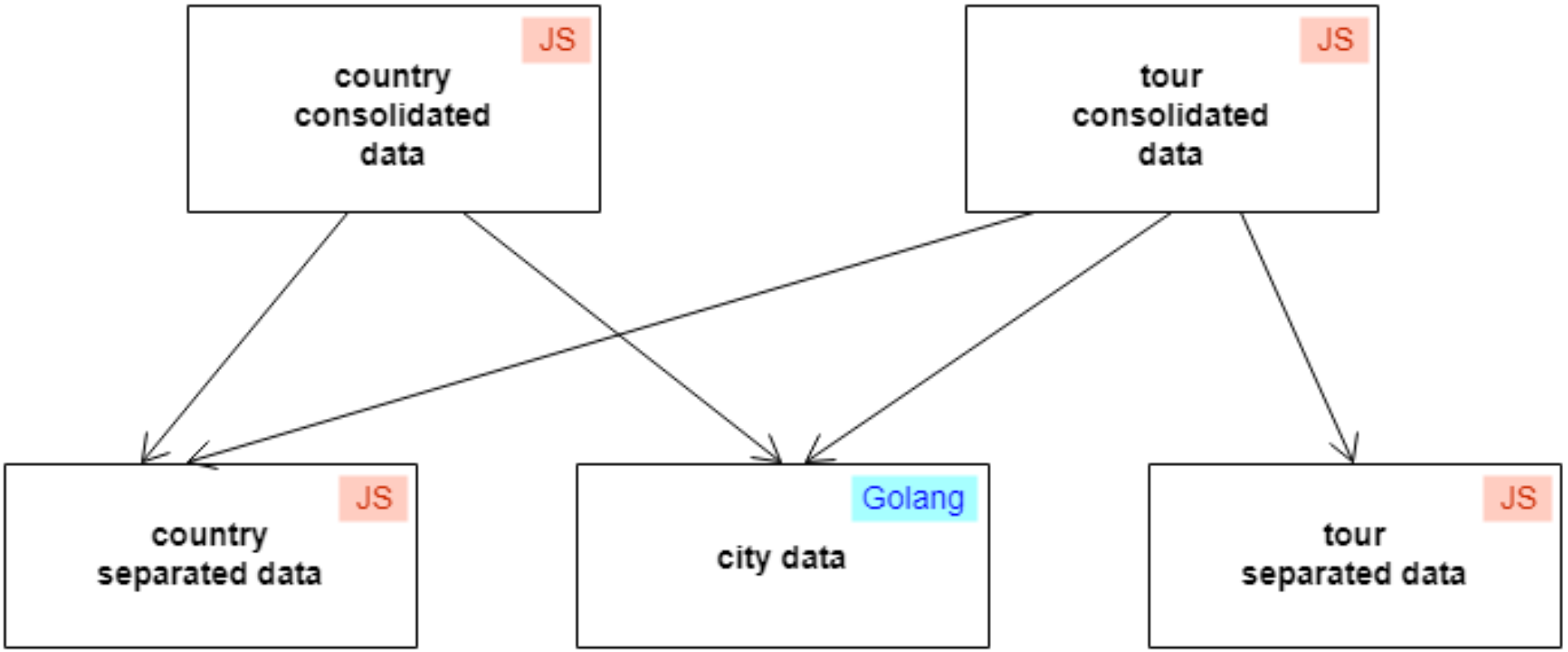
Se agregan más requerimientos de información sobre ciudades:
`cities/:cityId/attractions` `cities/:cityId/hotels`
etc.

- Se separa el equipo de desarrollo de `/cities`.
- El equipo separado decide reimplementar usando Golang.

Reimplementación de `/cities`



Reimplementación de `/cities`



. . . veamoslo andando . . .

Independencia y reimplementación de `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

Se agregan más requerimientos de información sobre ciudades:
`cities/:cityId/attractions` `cities/:cityId/hotels`
etc.

- Se separa el equipo de desarrollo de `/cities`.
- El equipo separado decide reimplementar usando Golang.

Independencia y reimplementación de `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

Se agregan más requerimientos de información sobre ciudades:
`cities/:cityId/attractions` `cities/:cityId/hotels`
etc.

- Se separa el equipo de desarrollo de `/cities`.
- El equipo separado decide reimplementar usando Golang.

Independencia y reimplementación de `/cities`

- Separación entre componentes.
- Organización del equipo por funcionalidad.
- Gestión descentralizada.

Se agregan más requerimientos de información sobre ciudades:
`cities/:cityId/attractions` `cities/:cityId/hotels`
etc.

- Se separa el equipo de desarrollo de `/cities`.
- El equipo separado decide reimplementar usando Golang.

Pautas – gestión descentralizada

Autonomía para elegir tecnologías, herramientas, forma de trabajo.

Fortalezas

- Facilita experimentación.
- Permite usar tecnologías específicas.
- Simplifica la incorporación de nuevos desarrolladores.

Debilidades/riesgos

- Hace más complejo el mantenimiento.
- Dificulta la introducción de mejoras globales.
- Menor facilidad para los cambios de equipo.

Pautas – gestión descentralizada

Autonomía para elegir tecnologías, herramientas, forma de trabajo.

Fortalezas

- Facilita experimentación.
- Permite usar tecnologías específicas.
- Simplifica la incorporación de nuevos desarrolladores.

Debilidades/riesgos

- Hace más complejo el mantenimiento.
- Dificulta la introducción de mejoras globales.
- Menor facilidad para los cambios de equipo.

Pautas – gestión descentralizada

Autonomía para elegir tecnologías, herramientas, forma de trabajo.

Fortalezas

- Facilita experimentación.
- Permite usar tecnologías específicas.
- Simplifica la incorporación de nuevos desarrolladores.

Debilidades/riesgos

- Hace más complejo el mantenimiento.
- Dificulta la introducción de mejoras globales.
- Menor facilidad para los cambios de equipo.

Pautas – gestión descentralizada

Autonomía para elegir tecnologías, herramientas, forma de trabajo.

Fortalezas

- Facilita experimentación.
- Permite usar tecnologías específicas.
- Simplifica la incorporación de nuevos desarrolladores.

Debilidades/riesgos

- Hace más complejo el mantenimiento.
- Dificulta la introducción de mejoras globales.
- Menor facilidad para los cambios de equipo.

La aplicación evoluciona

- Corrección de bugs.
- Aparición de nueva funcionalidad.
- Necesidad de escalar un servicio.
- División del equipo de desarrollo.
- Experimentación con nuevas herramientas.

Un minuto para hablar sobre – aspectos

Un minuto para hablar sobre – aspectos

Standards

Un minuto para hablar sobre – aspectos

Standards

Operaciones

Un minuto para hablar sobre – aspectos

Standards

Operaciones

Documentación

Un minuto para hablar sobre – características

Se lleva bien con

- Desarrollo ágil.
- Equipos autónomos.
- Reacción rápida.
- Evolución.

Tener especial cuidado con

- Rotación entre grupos.
- Overhead por comunicación mediante servicios.
- Monitoreo.
- Consistencia de datos.

Un minuto para hablar sobre – motivación

With larger, monolithic systems, there are fewer opportunities for people to step up and **own** *something. With microservices, on the other hand, we have multiple autonomous codebases that will have their own independent lifecycles (. . .)*

Un minuto para hablar sobre – motivación

With larger, monolithic systems, there are fewer opportunities for people to step up and **own** *something. With microservices, on the other hand, we have multiple autonomous codebases that will have their own independent lifecycles (. . .)*

(. . .) great software comes from great people (. . .)

Un minuto para hablar sobre – motivación

With larger, monolithic systems, there are fewer opportunities for people to step up and **own** *something. With microservices, on the other hand, we have multiple autonomous codebases that will have their own independent lifecycles (. . .)*

(. . .) great software comes from great people (. . .)

Sam Newman, Building Microservices – designing fine-grained systems. O'Reilly, 2015.

Un minuto para hablar sobre – pautas

Rules are for the obedience of fools and the guidance of wise men.

Sam Newman, Building Microservices – designing fine-grained systems. O'Reilly, 2015.

Volvamos a la base – pautas

Separación entre componentes.

Deploys ágiles, descentralizados, automáticos.

Interfaces livianas.

Organización del equipo por funcionalidad.

Gestión descentralizada.

Datos descentralizados.

Design for failure.

Volvamos a la base – pautas

Separación entre componentes.

Deploys ágiles, descentralizados, automáticos.

Interfaces livianas.

Organización del equipo por funcionalidad.

Gestión descentralizada.

Datos descentralizados.

Design for failure.

Volvamos a la base – pautas

Separación entre componentes.

Deploys ágiles, descentralizados, automáticos.

Interfaces livianas.

Organización del equipo por funcionalidad.

Gestión descentralizada.

Datos descentralizados.

Design for failure.

- Programación defensiva entre componentes.

País – información consolidada

```
{ "countryId":  "1",
    // countryMainData
  "name": "Argentina", "continent": "America",
  "capitalCityId": 1001,
  "capitalCityName": "Buenos Aires",
    // cityData
  "population": {  // currentPopulationData
    "total": 44272125, "females": 22603547, ...  }
  "mainCities": [ // mainCities
    { "cityId": 1003,
      "name": "Rosario", "population": 1358817, ...  }
      // cityData
   ...  ]  }
```

País – información consolidada – analizamos errores

```
{ "countryId":  "1",
    // countryMainData - on error: do not give results
  "name": "Argentina", "continent": "America",
  "capitalCityId": 1001,
  "capitalCityName": "Buenos Aires",
    // cityData - on error: (not available)
  "population": {  // currentPopulationData
    // on error: data not available
    "total": 44272125, "females": 22603547, ...  }
  "mainCities": [ // mainCities
    // on error: data not available
    { "cityId": 1003,
      "name": "Rosario", "population": 1358817, ...  }
      // cityData - on error: do not include city
   ...  ]  }
```

Un poco de programación defensiva

```
Promise.all([
  fetchCountryData(countryId, 'mainData'),
  fetchCountryDataOrNotFound(countryId, 'population')
])
.spread(function(mainCountryData, populationData) {
  if (populationData.notFound) {
    populationData = "Data not available"
  }
  mainCountryData.population = populationData
  response.status(200)
  response.json( mainCountryData )
})
.catch(function(err) {
  response.status(err.statusCode)
  response.json( { errorDescription: err.errorDescription } )
})
```

Pautas – Design for failure

De qué se trata

Programación defensiva entre componentes.

Fortalezas

Facilita desacoplamiento entre equipos.

Aplicaciones más robustas.

Debilidades/riesgos

Se agrega complejidad al desarrollo.

Pautas – Design for failure

De qué se trata

Programación defensiva entre componentes.

Fortalezas

Facilita desacoplamiento entre equipos.

Aplicaciones más robustas.

Debilidades/riesgos

Se agrega complejidad al desarrollo.

Llegando al final – un aspecto a destacar

Foco en escalar

- recursos asignados a deploy.
- equipo de trabajo.
- funcionalidad.

Para cerrar – la palabra de hoy

Evolución

Para cerrar – la palabra de hoy

Evolución

(. . .) our architects need to shift their thinking away from creating the perfect end product, and instead helping to create a framework in which the right systems can emerge, and continue to grow as we learn more.

Sam Newman, Building Microservices – designing fine-grained systems. O'Reilly, 2015.