



# Bitespeed Backend Task: Identity Reconciliation

Meet the brilliant yet eccentric Dr. Emmett Brown, better known as Doc. Hopelessly stuck in 2023, he is fixing his time machine to go back to the future and save his friend. His favourite online store FluxKart.com sells all the parts required to build this contraption. As crazy as he might get at times, Doc surely knows how to be careful. To avoid drawing attention to his grandiose project, Doc is using different email addresses and phone numbers for each purchase.

FluxKart.com is deadpan serious about their customer experience. There is nothing more important than rewarding their loyal customers and giving a personalised experience. To do this, FluxKart decides to integrate Bitespeed into their platform. Bitespeed collects contact details from shoppers for a personalised customer experience.

However, given Doc's modus operandi, Bitespeed faces a unique challenge: linking different orders made with different contact information to the same person.



## Bitespeed Needs Your Help!

Bitespeed needs a way to identify and keep track of a customer's identity across multiple purchases.

We know that orders on FluxKart.com will always have either an `email` or `phoneNumber` in the checkout event.

Bitespeed keeps track of the collected contact information in a relational database table named `Contact`.

```
{  
  id          Int  
  phoneNumber String?  
  email       String?
```

```

    linkedId          Int? // the ID of another Contact link
                        ed to this one
    linkPrecedence     "secondary"|"primary" // "primary" if
it's the first Contact in the link
    createdAt         DateTime
    updatedAt         DateTime
    deletedAt         DateTime?
}

```

One customer can have multiple **Contact** rows in the database against them. All of the rows are linked together with the oldest one being treated as "primary" and the rest as "secondary" .

**Contact** rows are linked if they have either of **email** or **phone** as common.

## For example:

If a customer placed an order with

**email=lorraine@hillvalley.edu** & **phoneNumber=123456**

and later came back to place another order with

**email=mcfly@hillvalley.edu** & **phoneNumber=123456** ,

database will have the following rows:

```

{
  id          1
  phoneNumber  "123456"
  email       "lorraine@hillvalley.edu"
  linkedId    null
  linkPrecedence "primary"
  createdAt   2023-04-01 00:00:00.374+00
  updatedAt   2023-04-01 00:00:00.374+00
  deletedAt   null
},

```

```
{
  id                23
  phoneNumber       "123456"
  email            "mcfly@hillvalley.edu"
  linkedId         1
  linkPrecedence   "secondary"
  createdAt        2023-04-20 05:30:00.11+00
  updatedAt        2023-04-20 05:30:00.11+00
  deletedAt        null
}
```

## Requirements

You are required to design a web service with an endpoint `/identify` that will receive HTTP POST requests with JSON body of the following format:

```
{
  "email"?: string,
  "phoneNumber"?: number
}
```

The web service should return an HTTP 200 response with a JSON payload containing the consolidated contact.

Your response should be in this format:

```
{
  "contact":{
    "primaryContactId": number,
    "emails": string[], // first element being email
of primary contact
    "phoneNumbers": string[], // first element being
```

```

phoneNumber of primary contact
    "secondaryContactIds": number[] // Array of all C
ontact IDs that are "secondary" to the primary contact
    }
}

```

## Extending the previous example:

Request:

```

{
  "email": "mcfly@hillvalley.edu",
  "phoneNumber": "123456"
}

```

will give the following response

```

{
  "contact": {
    "primaryContactId": 1,
    "emails": ["lorraine@hillvalley.edu", "mcfly@hillv
alley.edu"]
    "phoneNumbers": ["123456"]
    "secondaryContactIds": [23]
  }
}

```

▼ In fact, all of the following requests will return the above response (*use toggle to expand*)

```

{
  "email": null,

```

```
}  
  "phoneNumber": "123456"  
}
```

```
{  
  "email": "lorraine@hillvalley.edu",  
  "phoneNumber": null  
}
```

```
{  
  "email": "mcfly@hillvalley.edu",  
  "phoneNumber": null  
}
```

## But what happens if there are no existing contacts against an incoming request?

The service will simply create a new `Contact` row with `linkPrecedence="primary"` treating it as a new customer and return it with an empty array for `secondaryContactIds`

## When is a secondary contact created?

If an incoming request has either of `phoneNumber` or `email` common to an existing contact but contains new information, the service will create a "secondary" `Contact` row.

### Example:

#### Existing state of database:

```
{  
  id: 1  
  phoneNumber: "123456"  
  email: "lorraine@hillvalley.edu"  
}
```

```

    linkedId          null
    linkPrecedence    "primary"
    createdAt         2023-04-01 00:00:00.374+00
    updatedAt         2023-04-01 00:00:00.374+00
    deletedAt         null
  }

```

### Identify request:

```

{
  "email": "mcfly@hillvalley.edu",
  "phoneNumber": "123456"
}

```

### New state of database:

```

{
  id          1
  phoneNumber "123456"
  email       "lorraine@hillvalley.edu"
  linkedId    null
  linkPrecedence "primary"
  createdAt   2023-04-01 00:00:00.374+00
  updatedAt   2023-04-01 00:00:00.374+00
  deletedAt   null
},
{
  id          23
  phoneNumber "123456"
  email       "mcfly@hillvalley.edu"
  linkedId    1
  linkPrecedence "secondary"
  createdAt   2023-04-20 05:30:00.11+00
  updatedAt   2023-04-20 05:30:00.11+00
}

```

```
    deletedAt      null
  },
```

## Can primary contacts turn into secondary?

Yes. Let's take an example

### Existing state of database:

```
{
  id          11
  phoneNumber  "919191"
  email        "george@hillvalley.edu"
  linkedId    null
  linkPrecedence "primary"
  createdAt    2023-04-11 00:00:00.374+00
  updatedAt    2023-04-11 00:00:00.374+00
  deletedAt    null
},
{
  id          27
  phoneNumber  "717171"
  email        "biffucks@hillvalley.edu"
  linkedId    null
  linkPrecedence "primary"
  createdAt    2023-04-21 05:30:00.11+00
  updatedAt    2023-04-21 05:30:00.11+00
  deletedAt    null
}
```

### Request:

```
{
  "email": "george@hillvalley.edu",
```

```
"phoneNumber": "717171"  
}
```

### New state of database:

```
{  
  id          11  
  phoneNumber  "919191"  
  email       "george@hillvalley.edu"  
  linkedId    null  
  linkPrecedence "primary"  
  createdAt   2023-04-11 00:00:00.374+00  
  updatedAt   2023-04-11 00:00:00.374+00  
  deletedAt   null  
},  
{  
  id          27  
  phoneNumber  "717171"  
  email       "biffsucks@hillvalley.edu"  
  linkedId    11  
  linkPrecedence "secondary"  
  createdAt   2023-04-21 05:30:00.11+00  
  updatedAt   2023-04-28 06:40:00.23+00  
  deletedAt   null  
}
```

### Response:

```
{  
  "contact": {  
    "primaryContactId": 11,  
    "emails": ["george@hillvalley.edu", "biffsucks@  
hillvalley.edu"]  
    "phoneNumbers": ["919191", "717171"]  
    "secondaryContactIds": [27]  
  }  
}
```

```
}  
}
```

## What stack to use?

**Database:** Any SQL database can be used

**Backend framework:** NodeJs with typescript is preferred but any other framework can also be used.

## How to submit this task?

1. Publish the code repository to Github
2. Keep making small commits with insightful messages.
3. Expose the `/identify` endpoint
4. Host your app online and share the endpoint in the readme file. (You can use free hosting services like render.com)
5. Note: Use **JSON Body** and not **form-data** for request payloads.
6. **Submit the task here**

BiteSpeed - Frontend Task Submission



<https://forms.gle/hsQBJQ8tzbsp53D77>

## More about Bitespeed:

 [Way Of Life At BiteSpeed - Our Values & Purpose](#)