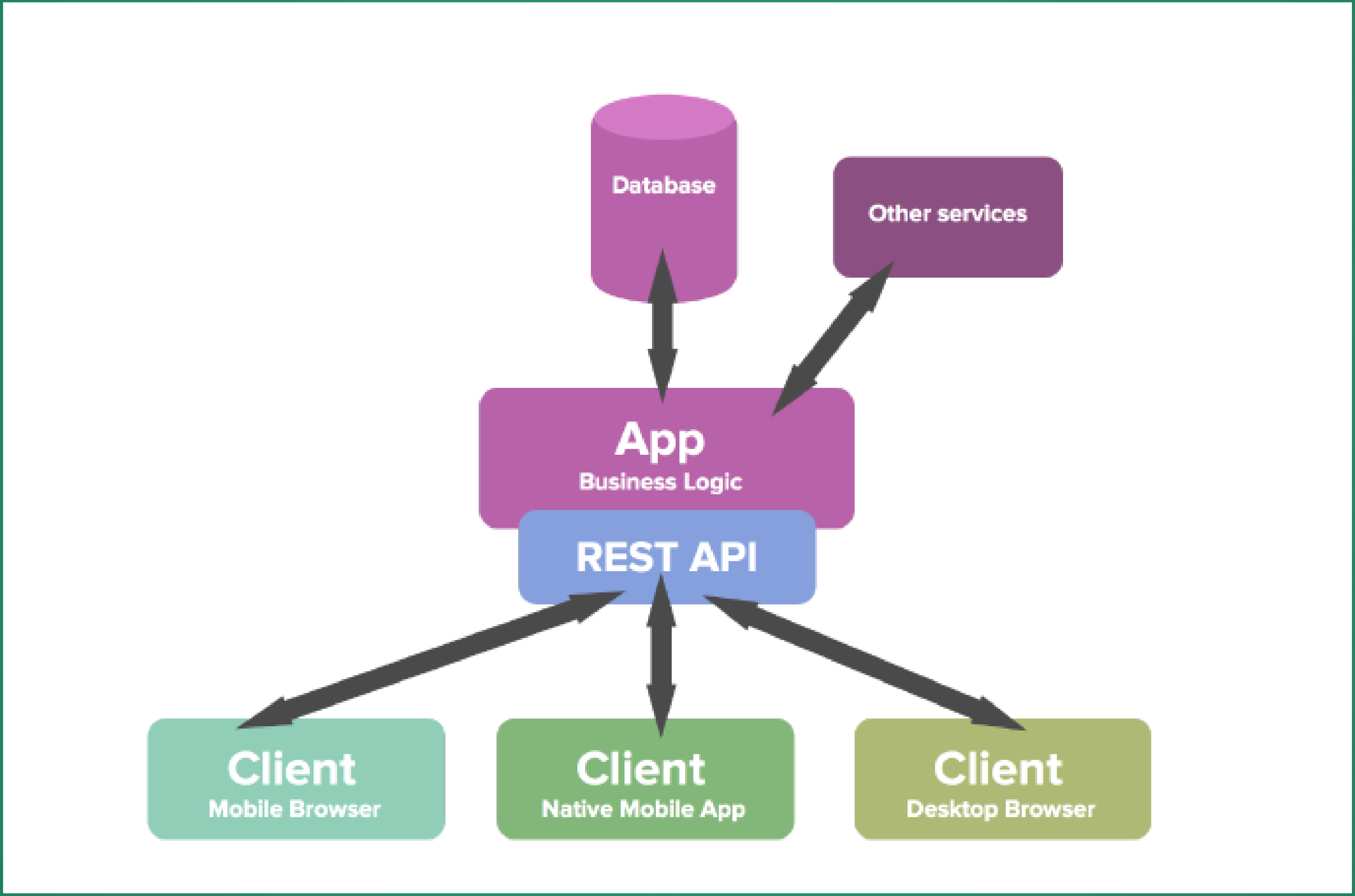
GraphQL

Graph Query Language

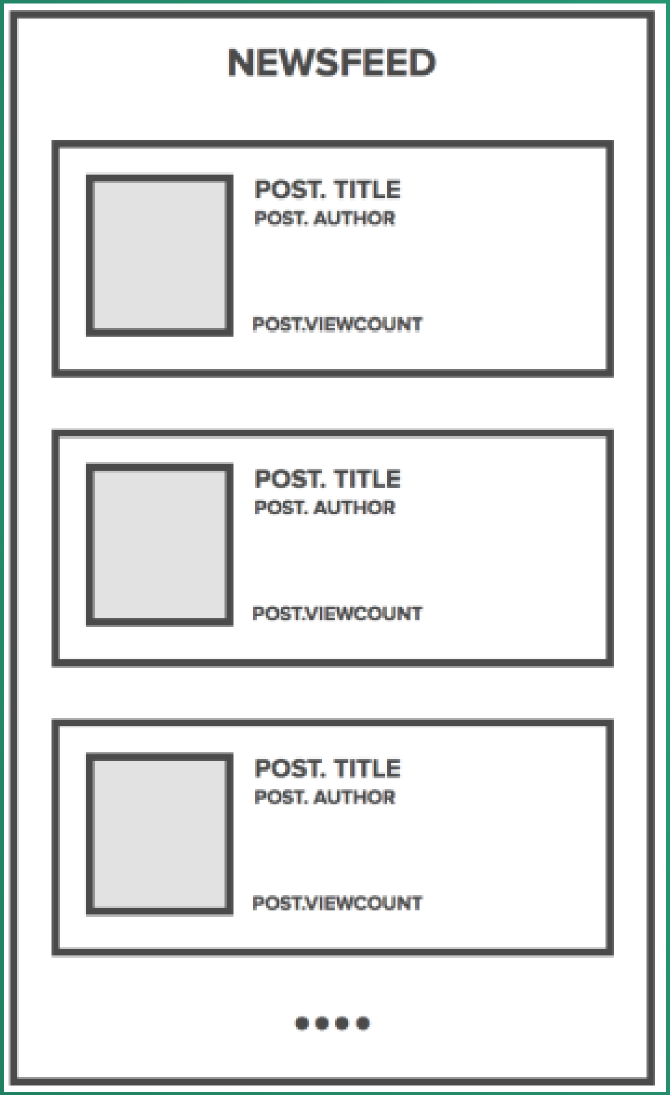
Typical REST architecture



Common issues that developers face w/ REST API

- Multiple Round Trips
- Over/Under Fetching of data
- Versioning headaches
- Number of URLs to remember
- Documention

Issue : Multiple Round Trips



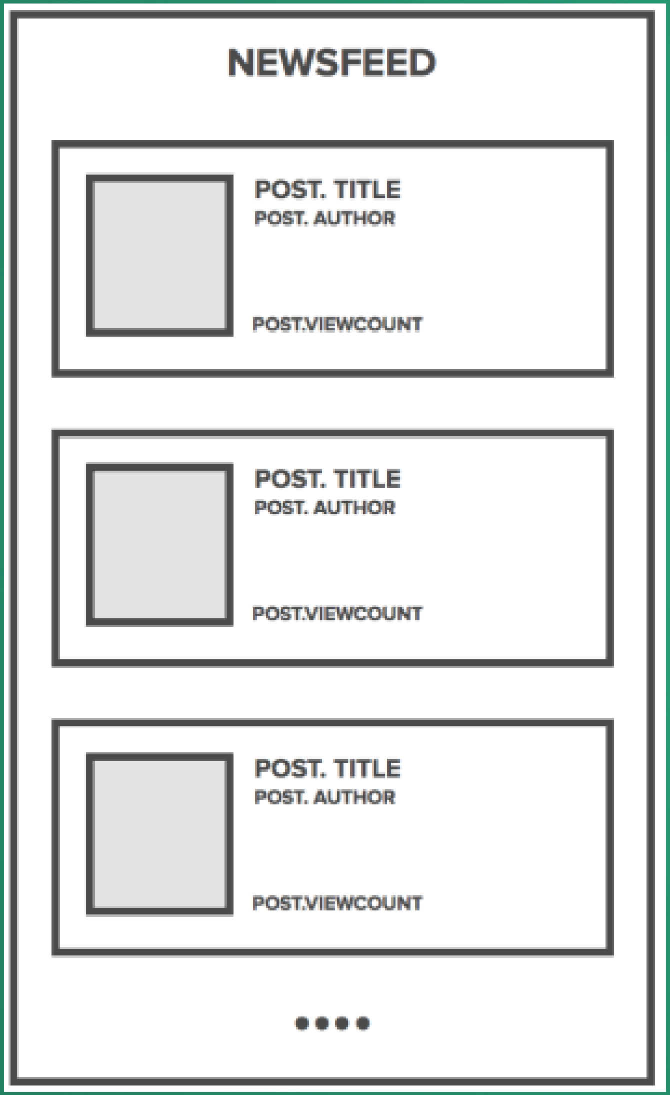
Designing REST API

```
Two resources
- Users
- Posts

POST /posts
GET /posts/1
PUT /posts/1
DELETE /posts/1
...

POST /users
GET /users/1
PUT /users/1
DELETE /users/1
```

Issue : Over Fetching



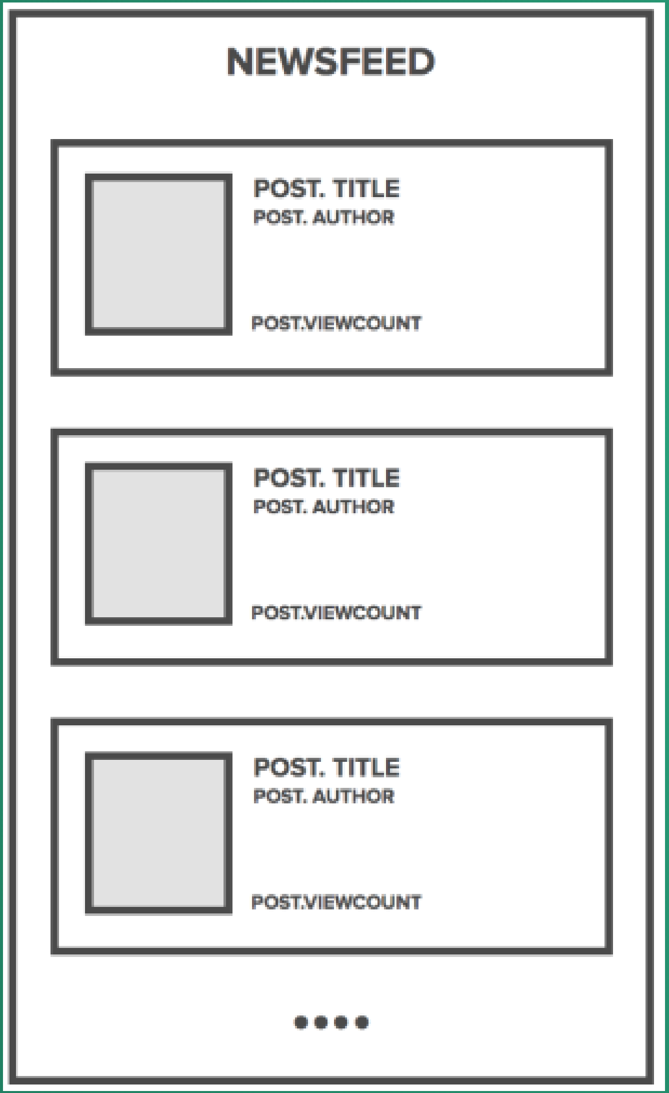
Designing REST API

```
Two resources
- Users
- Posts

POST /posts
GET /posts/1
PUT /posts/1
DELETE /posts/1
...

POST /users
GET /users/1
PUT /users/1
DELETE /users/1
```

Issue : Versioning



Designing REST API

```
Two resources
- Users
- Posts

POST /v1/posts
GET /v1/posts/1
PUT /v1/posts/1
DELETE /v1/posts/1
...

POST /v1/users
GET /v1/users/1
PUT /v1/users/1
DELETE /v1/users/1
```

What is GraphQL

GraphQL is a data query language and runtime designed and used at Facebook to request and deliver data to mobile and web apps since 2012

- Developed by Facebook
- Common interface between Client & Server for data fetching & manipulations
- Response format is controlled by Client instead of Server

Topics

- Our first GraphQL Server
- Types and Schemas
- Reading Data with Queries, Resolvers & Arguments
- Writing data with Mutations
- Connecting GraphQL to MongoDB

Our first GraphQL server

```
npm install express express-graphql graphql
```

DEMO

Our first GraphQL server

```
var express = require('express');
var graphqlHTTP = require('express-graphql');
var { buildSchema } = require('graphql');

// Construct a schema, using GraphQL schema language
var schema = buildSchema(`
  type Query {
    hello: String
  }
`);

// The root provides a resolver function for each API endpoint
var root = {
  hello: () => {
    return 'Hello world!';
  },
};

var app = express();
app.use('/graphql', graphqlHTTP({
  schema: schema,
  rootValue: root,
  graphiql: true,
}));
app.listen(4000);
```

Course Management GraphQL server

```javascript
const express = require('express');
const graphqlHTTP = require('express-graphql');
const gql = require('graphql');
const makeExecutableSchema = require('graphql-tools').makeExecutableSchema;
const cors = require('cors');

const typeDefs=require('./typeDefs')
const resolvers=require('./resolvers')

const port = process.env.PORT || 3000;
const app = express();
const schema = makeExecutableSchema({ typeDefs, resolvers });

app.use(
  '/graphql',
  cors(),
  graphqlHTTP({
    schema,
    graphiql: true
  })
);

app.listen(port);
console.log(`Server listening at localhost:${port}`);
```

typeDefs

```
let typeDefs=`
  type CourseType {
    id: ID
    name: String
    description: String
    level: String
  }
  type StudentType {
    id: ID
    firstName: String
    lastName: String
    active: Boolean
    courses: [CourseType]
  }
  input CourseInput {
    id: ID!
    name: String!
    description: String
    level: String
  }
  input StudentInput {
    id: ID!
    firstName: String!
    lastName: String!
    active: Boolean!
    coursesIds: [ID]!
  }
  type Query {
    allCourses: [CourseType]
    allStudents: [StudentType]
  }
  type Mutation {
    createCourse(name: String!, description: String, level: String): CourseType
    updateCourse(id: ID! name: String!, description: String, level: String): CourseType
    deleteCourse(id: ID!): CourseType
    createStudent(firstName: String! lastName: String!, active: Boolean!, coursesIds: [ID]!): StudentType
    updateStudent(id: ID!, firstName: String! lastName: String!, active: Boolean!, coursesIds: [ID]!): StudentType
    deleteStudent(id: ID!): StudentType
  }
`;
```

resolvers

```
var gql=require('graphql');
var db=require('./db')

const resolvers = {
  Query: {
    allCourses: () => {
      return new Promise((resolve,reject)=>{
        db.Course.find({},(err,courses)=>{
                if(err) reject(err);
                resolve(courses);
        })
      })
    },
    allStudents: () => {
      return [];
    }
  },
  Mutation: {
    createCourse: (_, { name, description, level }) => {
      return new Promise((resolve,reject)=>{
        var newCourse=new db.Course({
                name,
                description,
                level
        });
        newCourse.save((err,course)=>{
                if(err) reject(err);
                resolve(course);
        })
      })
    },
    updateCourse: (_, { id, name, description, level }) => {
      const input = { id, name, description, level };
      return input;
    },
    deleteCourse: (_, { id }) => {
      const input = { id };
      return input;
    },
    createStudent: (_, { firstName, lastName, active, coursesIds }) => {
```