Scenario

Your task is to create a simple Node.js application that exposes its API through REST service. We use the terms “client” and “server” to describe the roles in Client-server model.

Submitting your solution

Don’t hesitate to ask if you have any questions.

Please submit your solution as a ZIP attachment in a form of reply to the email where you’ve received this assignment.

Available HTTP endpoints

```
GET /address - Returns all available addresses
POST /address - Creates new address
GET /address/{id} - Returns specific address
PATCH /address/{id} - Modifies specific address
DELETE /address/{id} - Permanently removes specific address
```

Data format for all endpoints

Application expects content type `application/json` (default value if not specified explicitly) when communicating with the client, if any other content type is requested the server should respond with 415 code described in RFC 2616.

`GET /address`

Returns all available addresses as a list and responds with 200 code described in RFC 2616, example of response body:

```
[
	{
		"id": "581b5b28f3bc7b88210c4fe2",
		"country": "CZ",
		"city": "Brno",
		"street": "Husova",
		"postalcode": "60200",
		"number": 6,
		"numberAddition": "",
		"createdAt": "2016-11-03T15:22:31Z",
		"updatedAt": "2016-11-03T15:22:31Z",
		"status": null,
		"name": null,
		"email": null
	}
]
```

POST /address

Creates a new address and responds with 201 code described in RFC 2616, example of request body:

```
{
	"country": "CZ",
	"city": "Brno",
	"street": "Husova",
	"postalcode": "60200",
	"number": 6,
	"numberAddition": ""
}
```

Only attributes described above are allowed, all of them are required. `country` has to be a valid Alpha-2 code from ISO 3166-1. `city` and `street` have to be non-empty strings, `postalcode` has to be a string with length of 5 characters and can only contain digits. `number` has to be a positive integer and `numberAddition` has to be string, empty value is allowed.

If any of these attributes will not be provided or will be provided in incorrect format - 422 code described in RFC 4918 should be returned.

Example of response body:

```
{
	"id": "581b5b28f3bc7b88210c4fe2",
	"country": "CZ",
	"city": "Brno",
        "street": "Husova",
        "postalcode": "60200",
        "number": 6,
        "numberAddition": "",
        "createdAt": "2016-11-03T15:22:31Z",
        "updatedAt": "2016-11-03T15:22:31Z",
        "status": null,
        "name": null,
        "email": null,
}
```

`id` attribute will be assigned automatically by the database, both `createdAt` and `modifiedAt` will be assigned automatically on the server. Values for `status`, `name` and `email` will be set to `null`.

`GET /address/{id}`

Returns specific address as a hash and responds with 200 code described in RFC 2616, example of response body for `/address/581b5b28f3bc7b88210c4fe2`:

```
{
        "id": "581b5b28f3bc7b88210c4fe2",
        "country": "CZ",
        "city": "Brno",
        "street": "Husova",
        "postalcode": "60200",
        "number": 6,
        "numberAddition": "",
        "createdAt": "2016-11-03T15:22:31Z",
        "updatedAt": "2016-11-03T15:22:31Z",
        "status": null,
        "name": null,
        "email": null
}
```

If the address specified by `id` does not exist 404 response code described in RFC 2616 should be returned with a message describing why the request could not be completed.

PATCH /address/{id}

Updates specific address with specific attributes and responds with 200 code described in RFC 2616, example of request body for `/address/581b5b28f3bc7b88210c4fe2`:

```
{
        "status": "not at home",
        "name": "Bart Simpson",
        "email": "bart@simpson.com"
}
```

Only attributes `status`, `name` and `email` are allowed to be updated by the client. `status` can have one of these 3 values: “not at home”, “not interested” or “interested” and has to always be provided. `name` is optional and has to be a string if provided, `email` is optional and has to be in valid email format if provided. Property `updatedAt` has to be set to current date and time automatically if request succeeded.

All of these attributes can only be provided if currently saved `status` has either value `null` or “not at home”. If `status` has either value “not interested” or “interested” no further changes should be allowed and 403 response code described in RFC 2616 should be returned with a message describing why the request could not be completed.
Example of response body:

```
{
        "id": "581b5b28f3bc7b88210c4fe2",
        "country": "CZ",
        "city": "Brno",
        "street": "Husova",
        "postalcode": "60200",
        "number": 6,
        "numberAddition": "",
        "createdAt": "2016-11-03T15:22:31Z",
        "updatedAt": "2016-11-03T15:22:31Z",
        "status": null,
        "name": null,
        "email": null
}
```

If the address specified by `id` does not exist 404 response code described in RFC 2616 should be returned with message describing why the request could not be completed.

If any of request body attributes will be provided in incorrect format - 422 code described in RFC 4918 should be returned.

`DELETE /address/{id}`

Permanently removes specific address and responds with 204 response code described in RFC 2616. Empty response body should be returned if requests succeeds.

If the address specified by `id` does not exist 404 response code described in RFC 2616 should be returned with message describing why the request could not be completed.

In case request cannot be completed 409 response code described in RFC 2616 should be returned.

Testing

You can use any application you like for testing - even a web browser, here are some ideas on how to test the API you'll create:

- Postman - Chrome extension or Mac app
- Advanced REST Client - Chrome extension
- HTTPie - command line HTTP client
- curl - command line HTTP client

You can write some automated tests if you like, but it's out of the scope of basic version of this assignment.

Implementation

The only requirement we have besides using Node.js is to use MongoDB as your data store. You can create a free MongoDB instance at MongoDB Atlas or use a local one.

Bonus features

You don't need to implement the following features in order to complete the basic version of this assignment, but they can distinguish you as a more experienced developer.

1. Implement the logic of creating and updating addresses in separate module.
2. Include `Location` header with correct URL for HTTP response of `POST /address`.
3. Implement basic caching of endpoints responding to `GET` requests via combination of `Last-Modified` and `If-Modified-Since` HTTP headers.

Final instructions

We don't put a strong time limit on completing this assignment, but it should be possible for you to implement all of required features within one day (e.g. during a weekend or couple of evenings after work) even if you're not familiar with the technologies we want you to use.

It's expected that you may need to look into the documentation of the technologies mentioned above.

Keep these points in mind when working on this assignment:

1. Submitted code should be of the same quality as any other code that you would deploy to production environment. Well structured and clean code is very important to us.
2. Let us know which requirements (if any) you were not able to cover in the README file. **You may not need to implement all of the requirements depending on your prior experience and the position you are applying for**. We appreciate the progress you make in either case.
3. Make sure to set the proper content type and return valid JSON where applicable.

Good luck with your assignment!