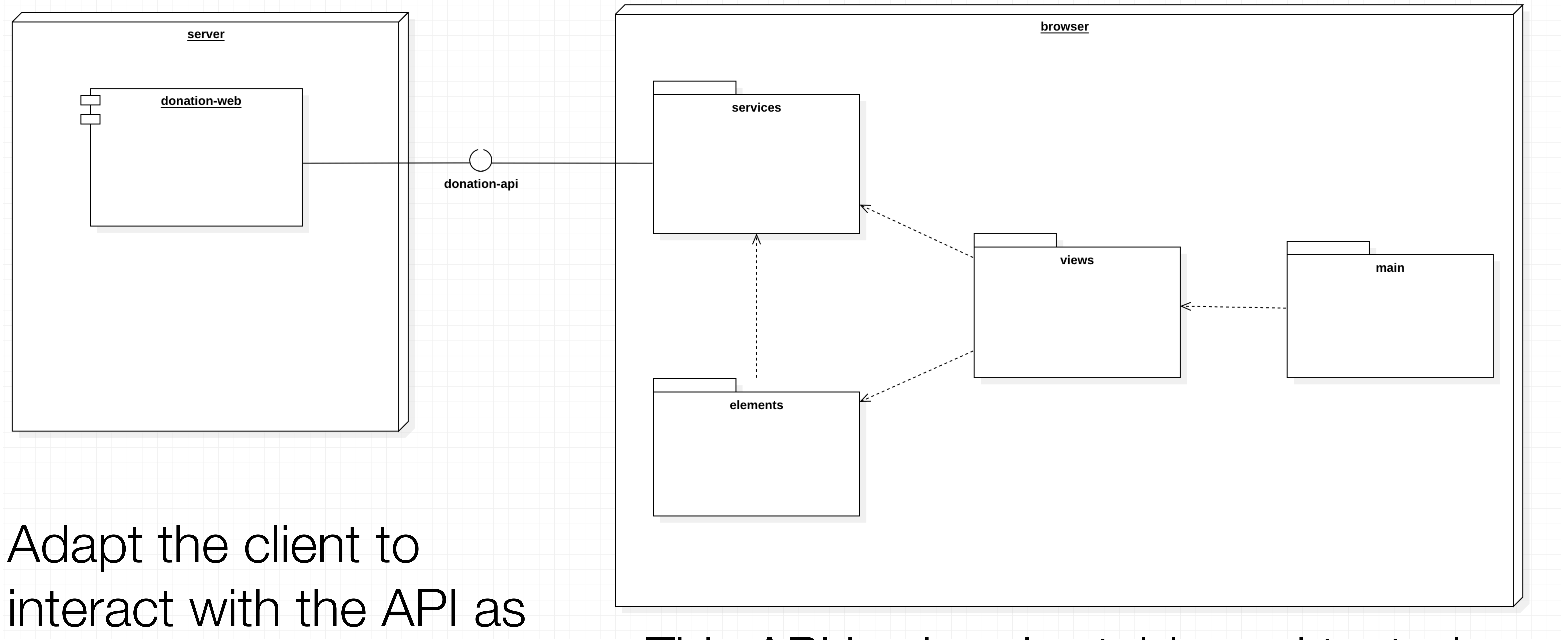
Donation-Client/Donation-web

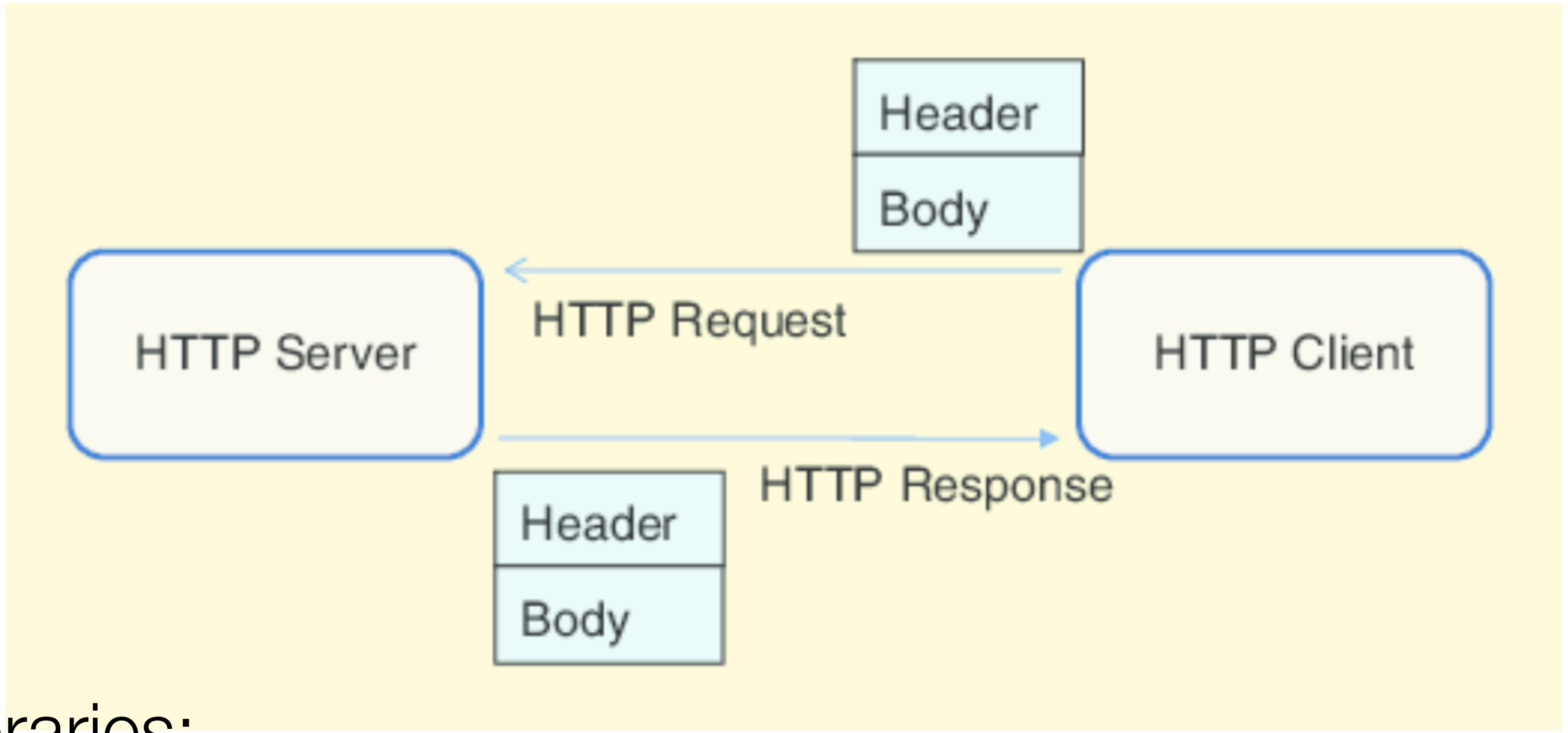
Donation-web Integration

Hook up the doantion-client application to donation-web server.

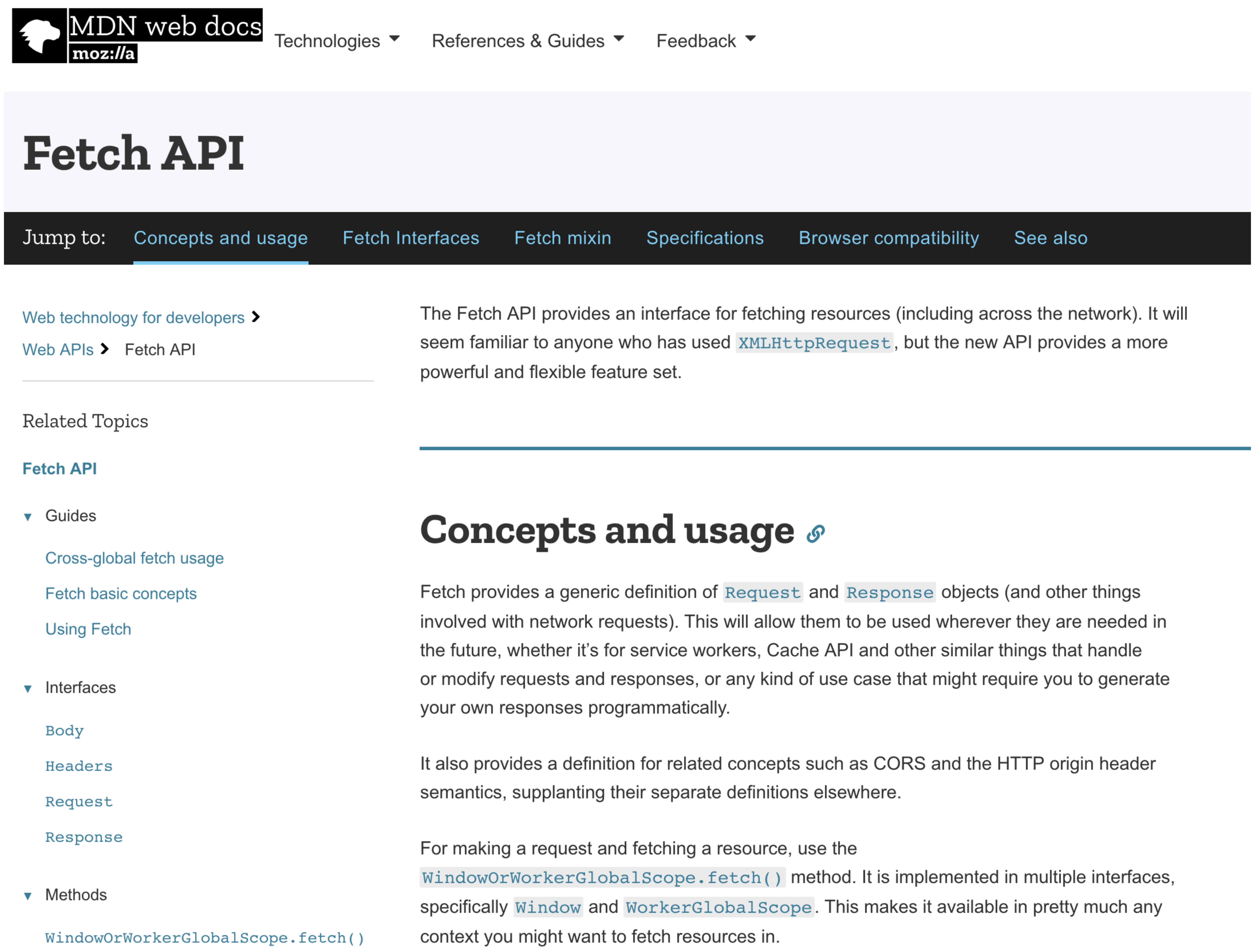


- Adapt the client to interact with the API as developed in the Hap web application
- This API is already stable and tested

Http Clients



- Aurelia comes with 2 http client libraries:
    - **aurelia-http-client** - A basic HttpClient based on XMLHttpRequest. It supports all HTTP verbs, JSONP and request cancellation.
    - **aurelia-fetch-clien**t - A more forward-looking HttpClient based on the Fetch specification. It supports all HTTP verbs and integrates with Service Workers, including Request/Response caching.



MDN web docs moz://a Technologies References & Guides Feedback

Fetch API

Jump to: Concepts and usage Fetch Interfaces Fetch mixin Specifications Browser compatibility See also

Web technology for developers › Web APIs › Fetch API

Related Topics

Fetch API

▼ Guides

- Cross-global fetch usage
- Fetch basic concepts
- Using Fetch

▼ Interfaces

- `Body`
- `Headers`
- `Request`
- `Response`

▼ Methods

- `WindowOrWorkerGlobalScope.fetch()`

The Fetch API provides an interface for fetching resources (including across the network). It will seem familiar to anyone who has used `XMLHttpRequest`, but the new API provides a more powerful and flexible feature set.

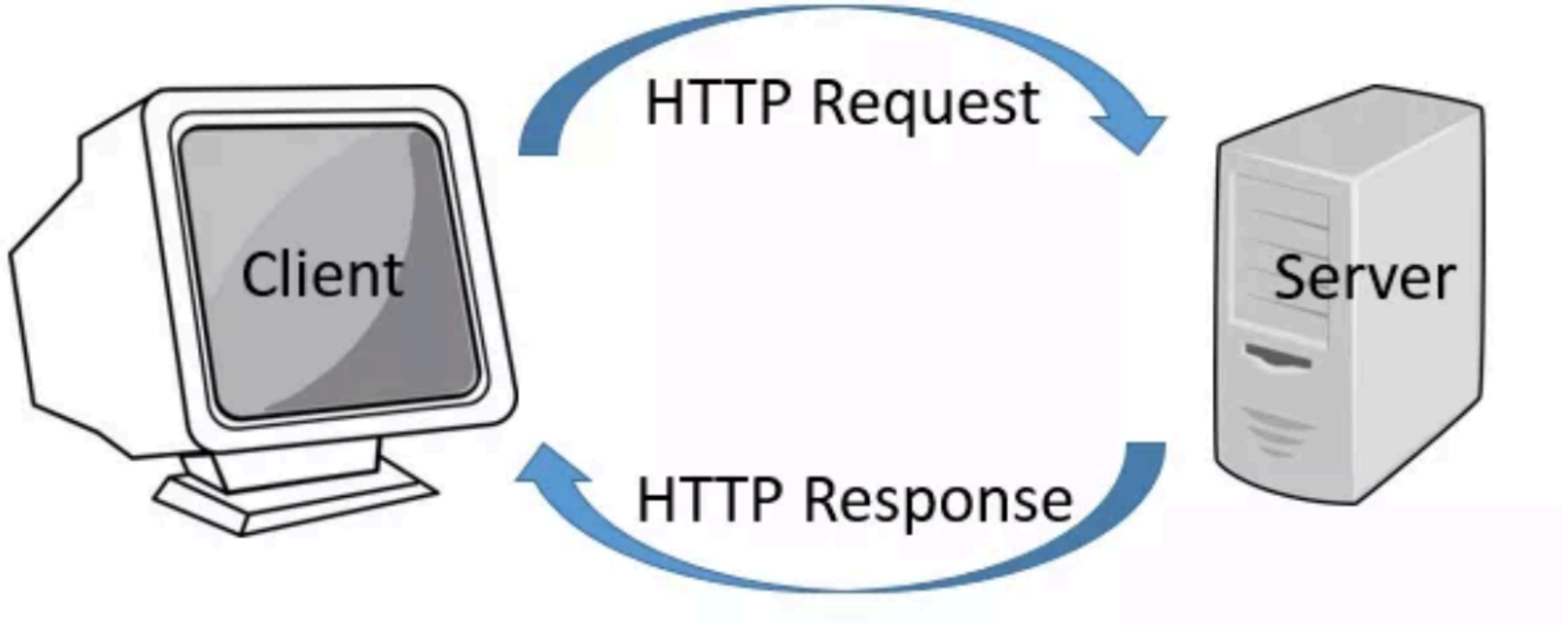
Concepts and usage

Fetch provides a generic definition of `Request` and `Response` objects (and other things involved with network requests). This will allow them to be used wherever they are needed in the future, whether it's for service workers, Cache API and other similar things that handle or modify requests and responses, or any kind of use case that might require you to generate your own responses programmatically.

It also provides a definition for related concepts such as CORS and the HTTP origin header semantics, supplanting their separate definitions elsewhere.

For making a request and fetching a resource, use the `WindowOrWorkerGlobalScope.fetch()` method. It is implemented in multiple interfaces, specifically `Window` and `WorkerGlobalScope`. This makes it available in pretty much any context you might want to fetch resources in.

aurelia-http-client



- Provides a comfortable interface to the browser's XMLHttpRequest object.

- Not included in the modules that Aurelia's bootstrapper installs, since it's completely optional and many apps may choose to use a different strategy for data retrieval.

- Must install it first…

Installing aurelia-http-client

```
yarn add aurelia-http-client
```

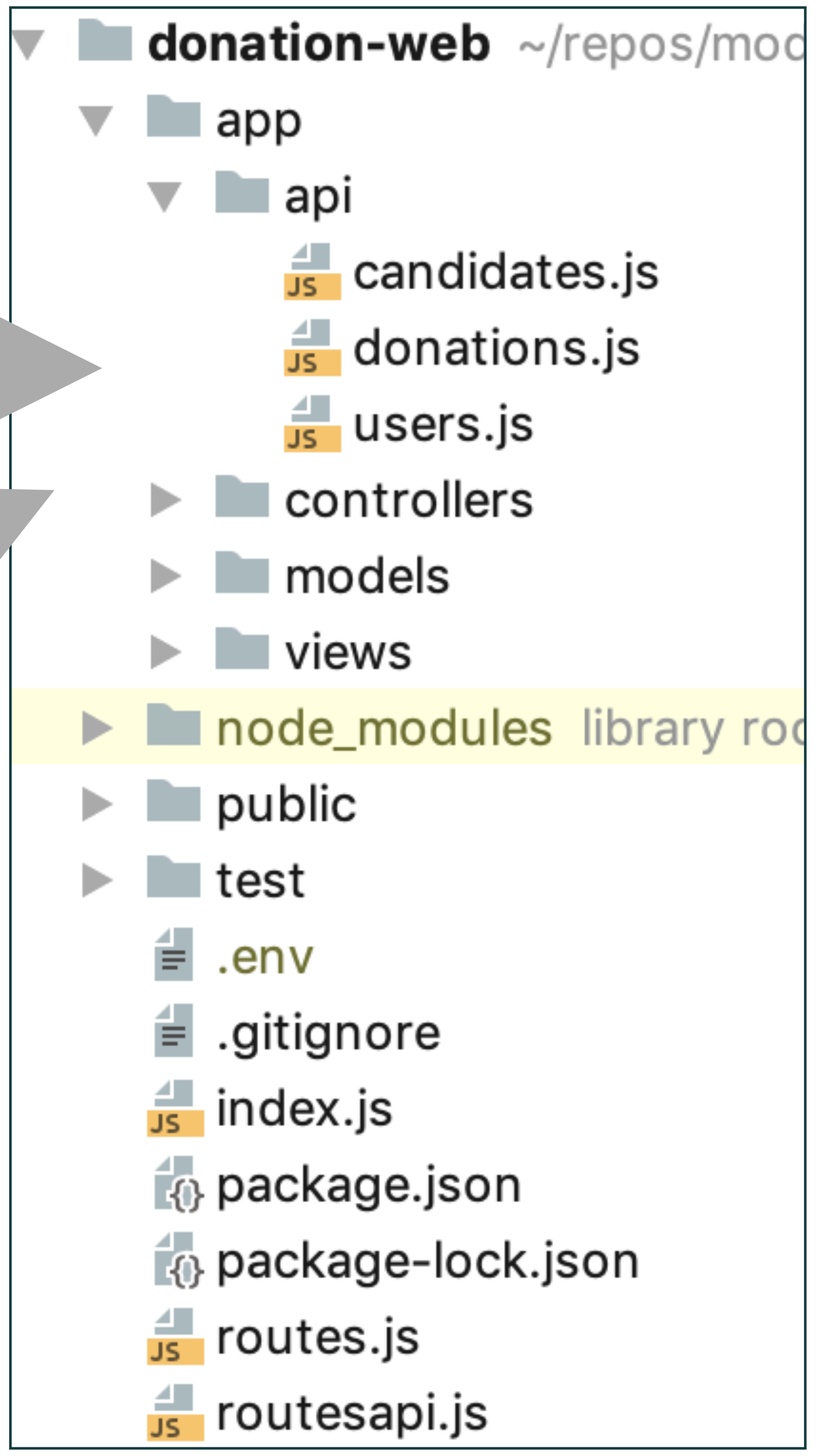
- We already installed this as we are loading the Json test data from the static folder of our client application

```
{
  "name": "donation-aurelia",
  "description": "An Aurelia client application.",
  "version": "0.1.0",
  "repository": {
    "type": "???",
    "url": "???"
  },
  "license": "MIT",
  "dependencies": {
    "aurelia-animator-css": "^1.0.4",
    "aurelia-bootstrapper": "^2.3.0",
    "aurelia-http-client": "^1.3.0",
    "aurelia-polyfills": "^1.3.0",
    "bluebird": "^3.5.2"
  },
...
}
```

Change BaseUrl to point to donation-web

```
@inject(HttpClient, EventAggregator, Aurelia, Router)
export class DonationService {
  users: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

  constructor(private httpClient: HttpClient, private ea: EventAggregator,
              private au: Aurelia,            private router: Router) {
    httpClient.configure(http => {
      http.withBaseUrl(‘http://localhost:3000');
    });
    this.getCandidates();
    this.getUsers();
  }

  async getCandidates() {
    const response = await this.httpClient.get('/api/candidates.json');
    this.candidates = await response.content;
    console.log(this.candidates);
  }

  async getUsers() {
    const response = await this.httpClient.get('/api/users.json');
    const users = await response.content;
    users.forEach(user => {
      this.users.set(user.email, user);
    });
  }
…
}
```

Donaton-web Application url

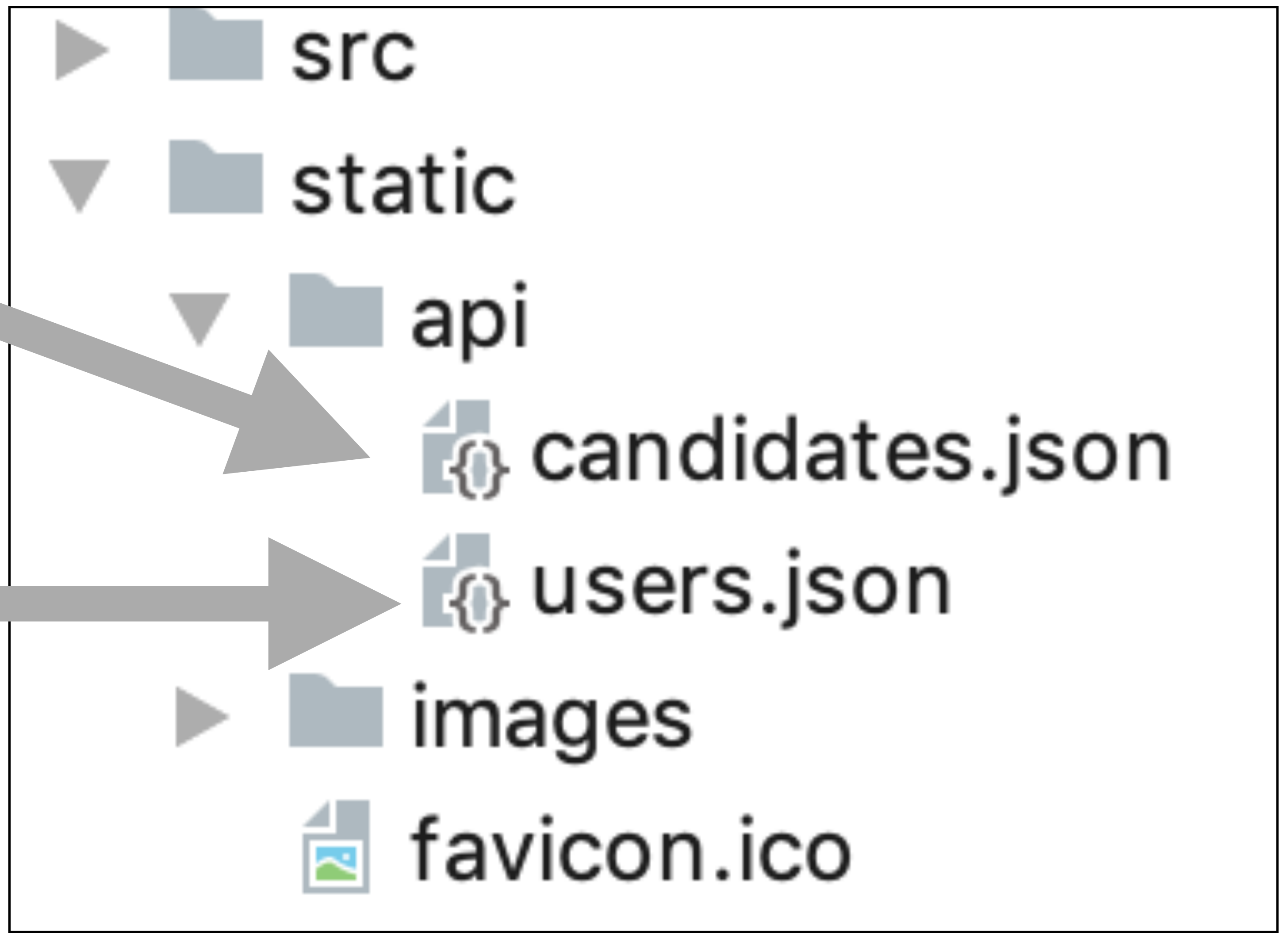


```
▼ donation-web ~/repos/moc
  ▼ app
    ▼ api
        candidates.js
        donations.js
        users.js
    ▶ controllers
    ▶ models
    ▶ views
  ▶ node_modules library roo
  ▶ public
  ▶ test
    .env
    .gitignore
    index.js
    package.json
    package-lock.json
    routes.js
    routesapi.js
```

```
@inject(HttpClient, EventAggregator, Aurelia, Router)
export class DonationService {
  users: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

  constructor(private httpClient: HttpClient, private ea: EventAggregator,
              private au: Aurelia,            private router: Router) {
    httpClient.configure(http => {
      http.withBaseUrl('http://localhost:8080');
    });
    this.getCandidates();
    this.getUsers();
  }

  async getCandidates() {
    const response = await this.httpClient.get('/api/candidates.json');
    this.candidates = await response.content;
    console.log(this.candidates);
  }

  async getUsers() {
    const response = await this.httpClient.get('/api/users.json');
    const users = await response.content;
    users.forEach(user => {
      this.users.set(user.email, user);
    });
  }

…
}
```

aurelia-http-client

Client (Aurelia)
Application url



Import library

Inject reference

Set base url
donation-web must be running and listening at this url

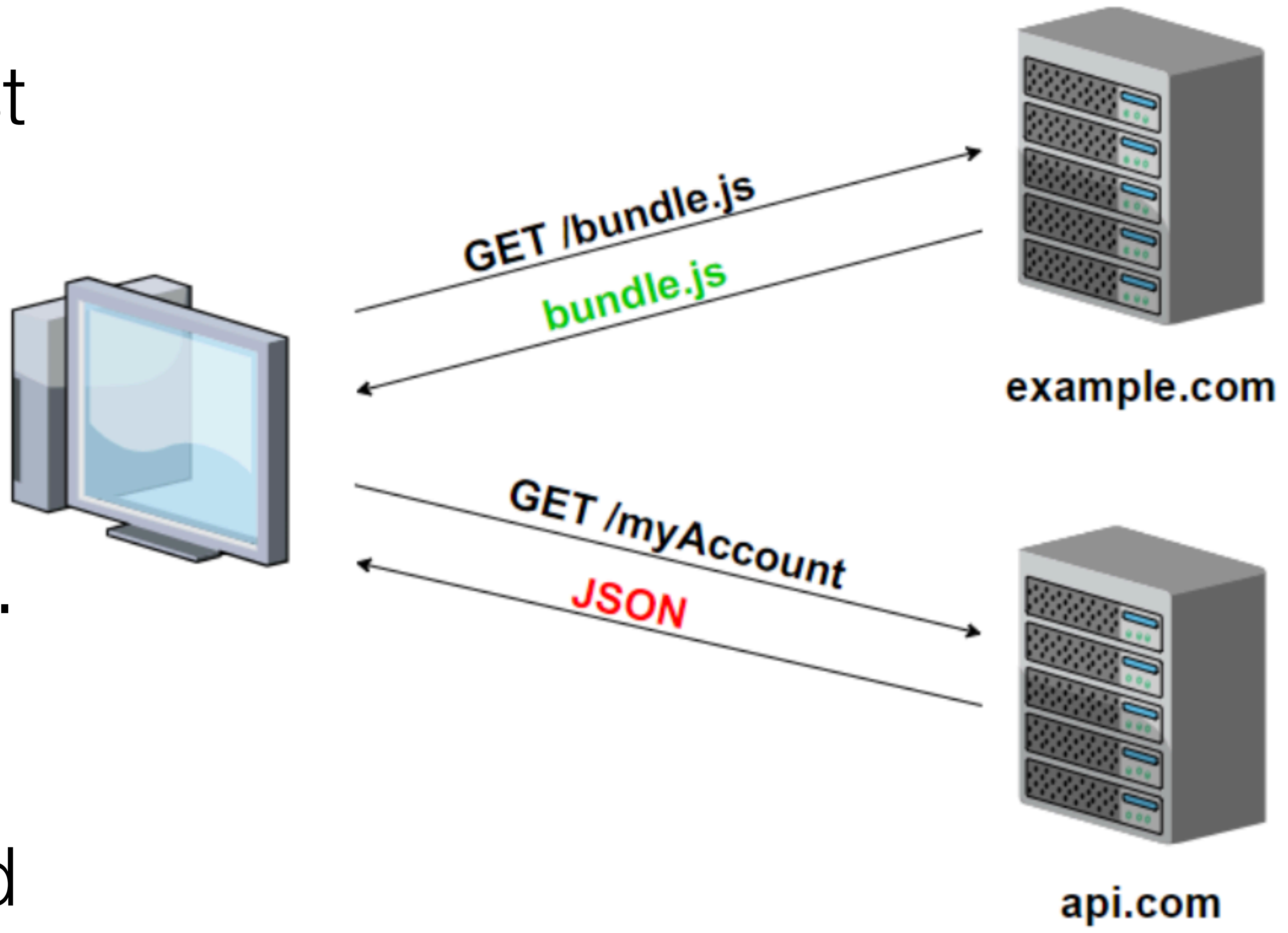
Generate http request & wait for response

Recover the json playload in javascript objects

```
import { HttpClient } from 'aurelia-http-client';


@inject(HttpClient ...)
export class DonationService {

  candidates: Candidate[] = [];

  constructor(private httpClient: HttpClient, ...) {

    httpClient.configure(http => {
      http.withBaseUrl('http://localhost:3000');
    });

  }

  async getCandidates() {


    const response = await this.httpClient.get('/api/candidates');


    this.candidates = await response.content;


  }
  ...
}
```

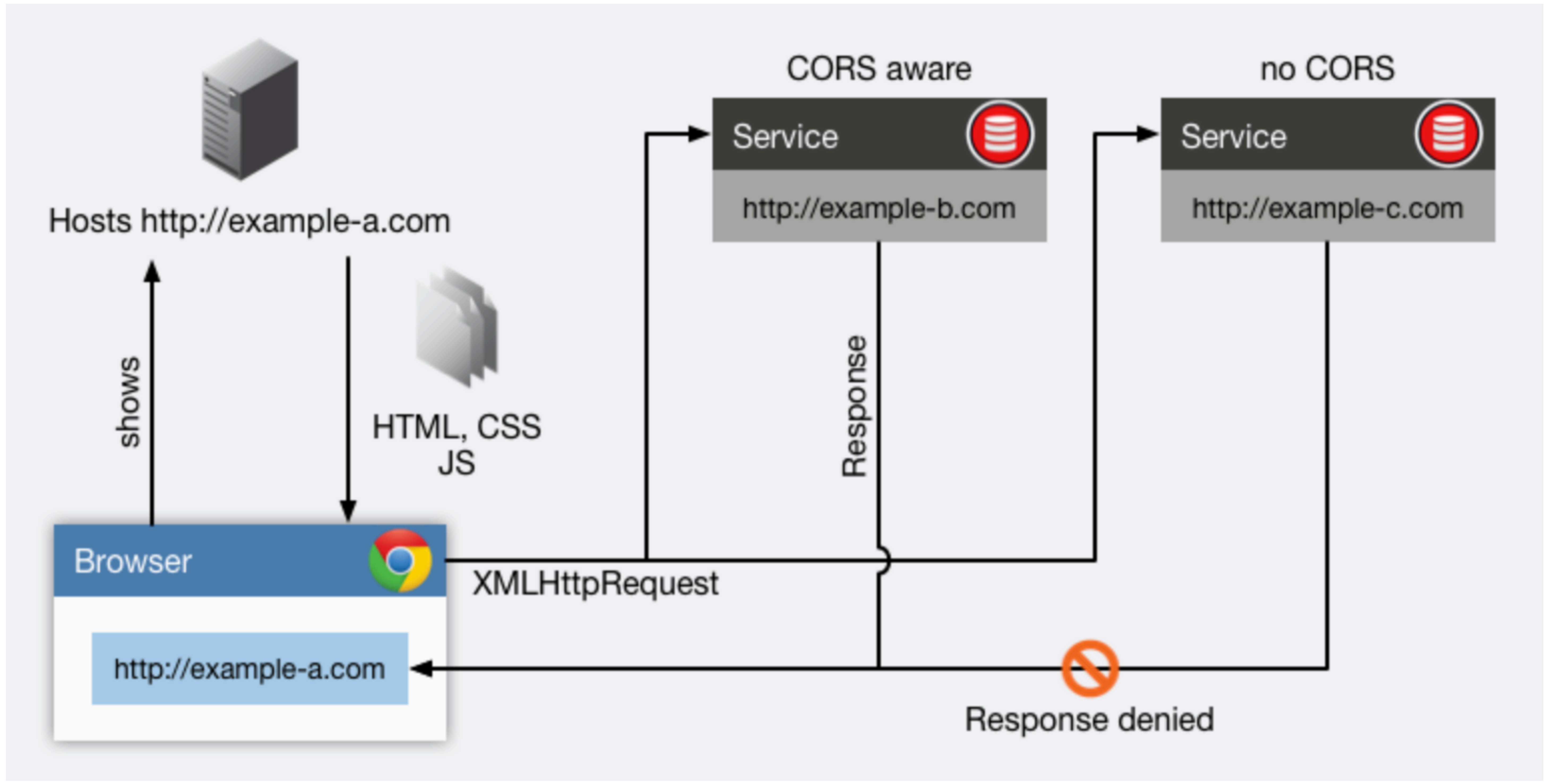
Cross Origin Requests

- A resource makes a cross-origin HTTP request when it requests a resource from a different domain than the one which the first resource itself serves.
- For example, an HTML page served from http://domain-a.com makes an <img> src request for http://domain-b.com/image.jpg.
- Many pages on the web today load resources like CSS stylesheets, images and scripts from separate domains.



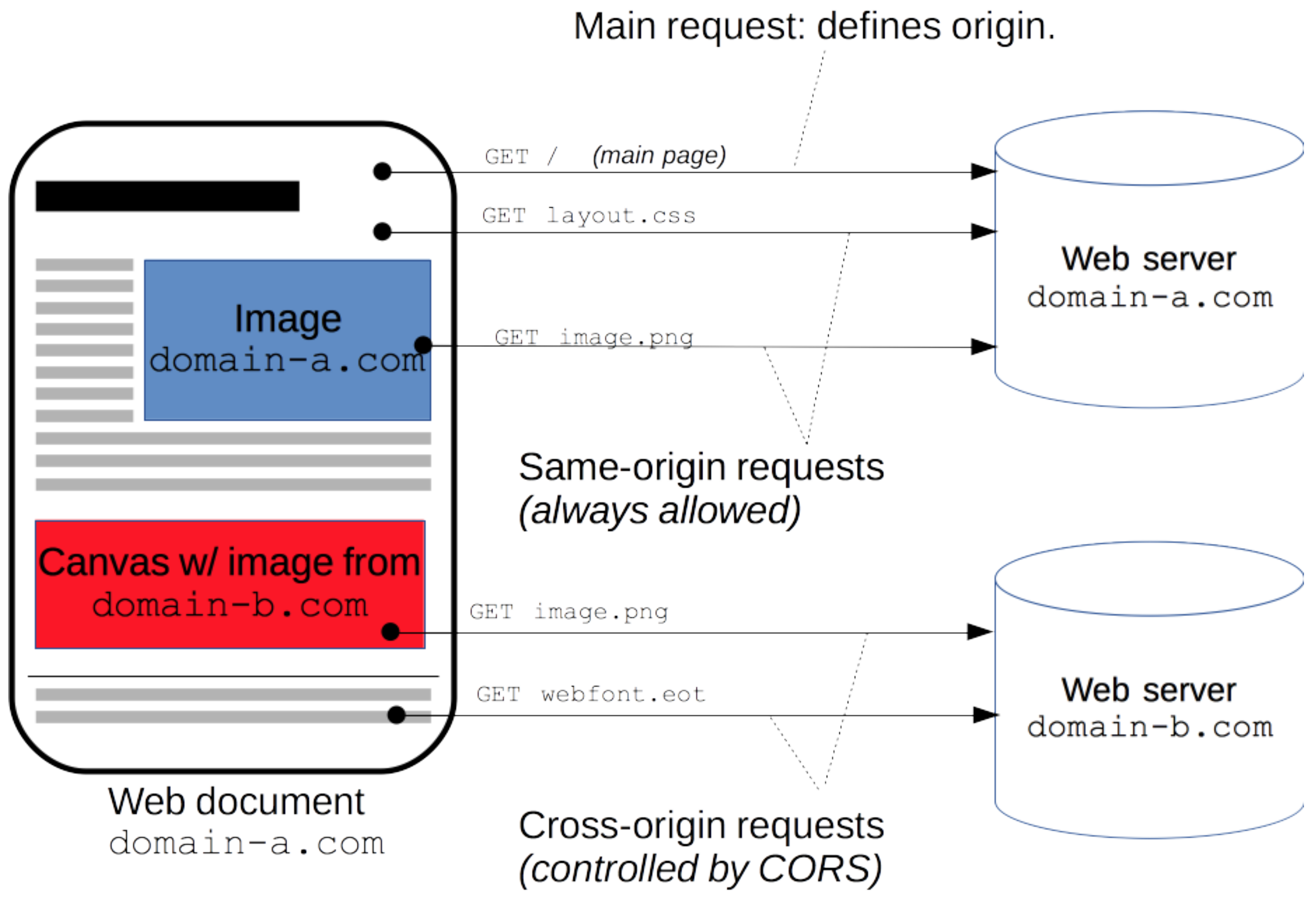
Restrictions

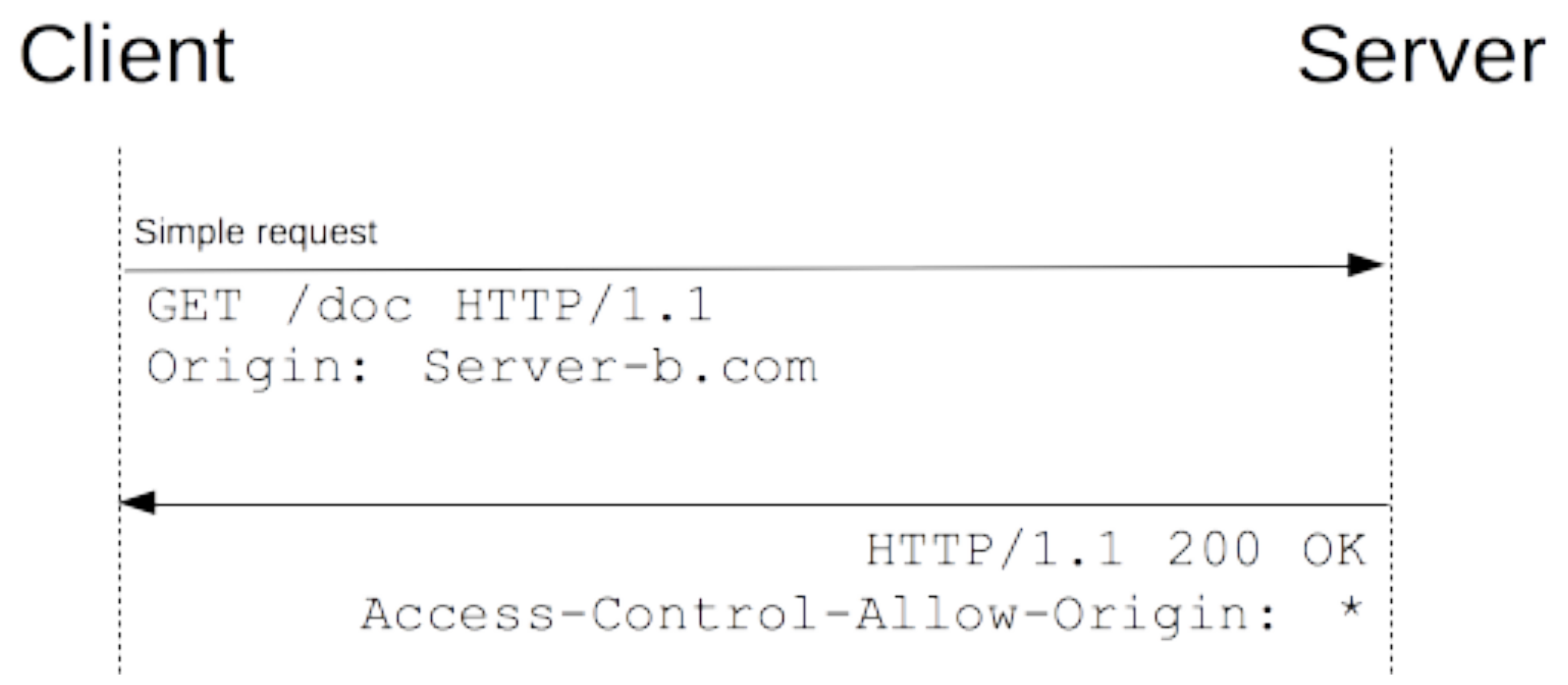
- For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts.
  - XMLHttpRequest follows the same-origin policy.
  - So, a web application using XMLHttpRequest could only make HTTP requests to its own domain.
- To improve web applications, developers asked browser vendors to allow cross-domain requests.



Cross Origin Resource Sharing (CORS)

- The Cross-Origin Resource Sharing (CORS) mechanism gives web servers cross-domain access controls, which enable secure cross-domain data transfers.





```
GET /doc HTTP/1.1
Origin: Server-b.com
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
```

https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

Cross Origin Request (COR)

- These requests to donation-web will fail due to COR restrictions
- The donation-web server will need some small modifications to permit this

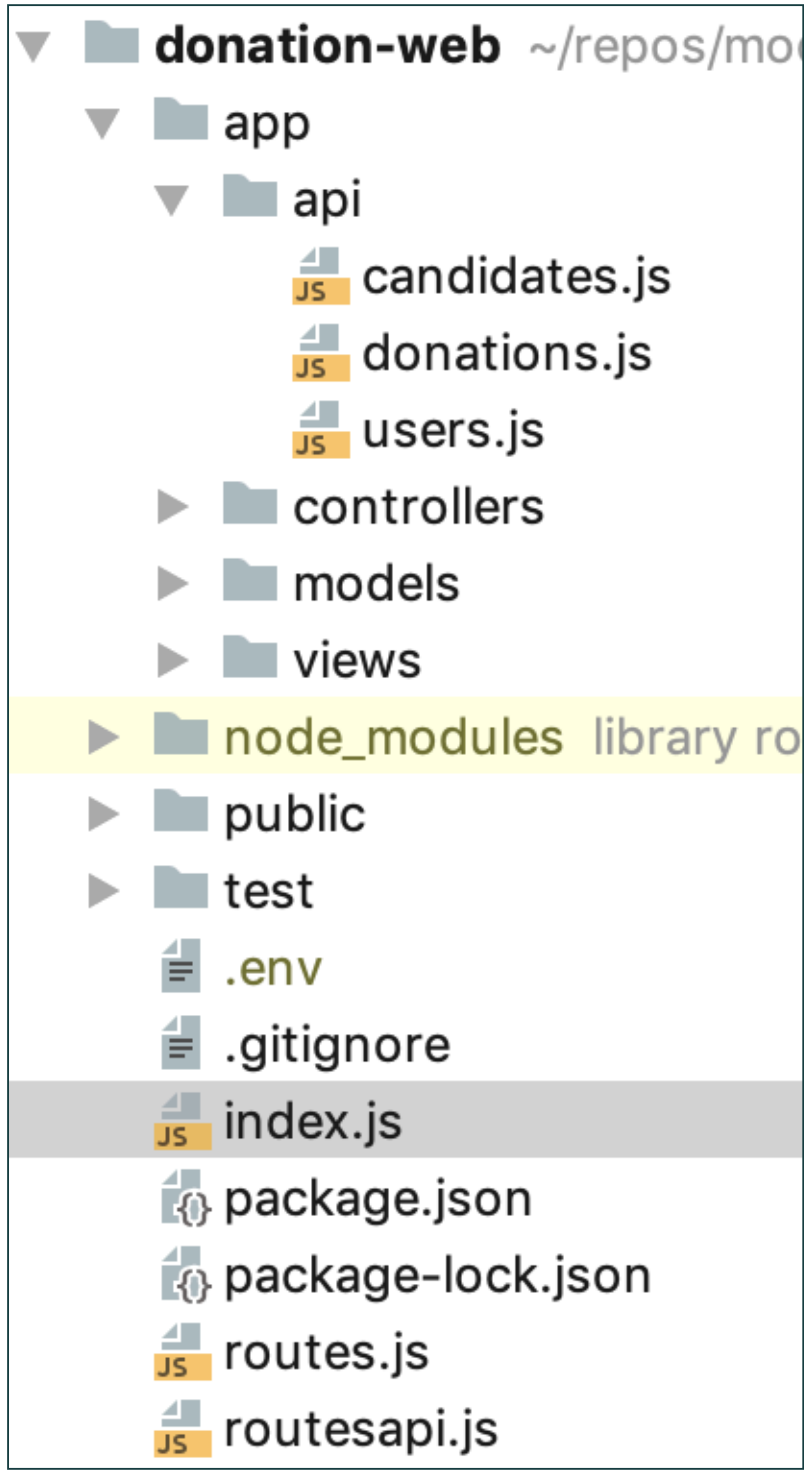
```
async getCandidates() {
  const response = await this.httpClient.get('/api/candidates.json');
  this.candidates = await response.content;
  console.log(this.candidates);
}

async getUsers() {
  const response = await this.httpClient.get('/api/users.json');
  const users = await response.content;
  users.forEach(user => {
    this.users.set(user.email, user);
  });
}
```

Update to donation-web

- Index.js modifications:

```
const server = Hapi.server({
  port: process.env.PORT || 3000
});
```

```
const server = Hapi.server({
  port: process.env.PORT || 3000,
  routes: { cors: true }
});
```



```
donation-web ~/repos/mo
  app
    api
      candidates.js
      donations.js
      users.js
    controllers
    models
    views
  node_modules library ro
  public
  test
  .env
  .gitignore
  index.js
  package.json
  package-lock.json
  routes.js
  routesapi.js
```

donation-types.ts

```
export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
}

export interface Donation {
  amount: number;
  method: string;
  candidate: Candidate;
}

export interface User {
  firstName: string;
  lastName: string;
  email: string;
  password: string;
}
```

Extend types to include IDs that will be part of donation-service responses

Distinguish between donations downloaded from server..

… and donations maintained locally with candidate references (instead of candidate ids)

```
export interface Candidate {
  firstName: string;
  lastName: string;
  office: string;
  _id : string;
}

export interface RawDonation {
  amount: number;
  method: string;
  candidate: string;
  donor: string;
}

export interface Donation {
  amount: number;
  method: string;
  candidate: Candidate;
}

export interface User {
  firstName: string;
  lastName: string;
  email: string;
  password: string;
  _id: string;
}
```

DonationService

- Can now be reworked to access donation-web to:
  - Make donations
  - Create candidates
  - Signup new users
- These features isolated from all other components

```
@inject(HttpClient, EventAggregator, Aurelia, Router)

export class DonationService {
  users: Map<string, User> = new Map();
  usersById: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

  constructor() {}

  async getCandidates() {}

  async getUsers() {}

  async getDonations() {}

  async createCandidate(firstName: string, lastName: string,
                                          office: string) {}

  async donate(amount: number, method: string, candidate: Candidate) {}

  async signup(firstName: string, lastName: string,
                  email: string, password: string) {}

  async login(email: string, password: string) {}

  logout() {}

  changeRouter(module: string) {}
}
```

DonationService: Initialisation

- Set BaseUrl to donation-web server
- Retrieve initial candidate, users and donation
- Users stored in a map, keyed by user email

```
@inject(HttpClient, EventAggregator, Aurelia, Router)
export class DonationService {
  users: Map<string, User> = new Map();
  usersById: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

constructor(private httpClient: HttpClient, private ea: EventAggregator,
            private au: Aurelia,            private router: Router) {

  httpClient.configure(http => {
    http.withBaseUrl('http://localhost:3000');
  });
  this.getCandidates();
  this.getUsers();
  this.getDonations();
}

async getCandidates() {
  const response = await this.httpClient.get('/api/candidates');
  this.candidates = await response.content;
}

async getUsers() {
  const response = await this.httpClient.get('/api/users');
  const users = await response.content;
  users.forEach(user => {
    this.users.set(user.email, user);
    this.usersById.set(user._id, user);
  });
}
```

DonationService: getDonations

- Retrieve all donations from server
- Download as RawDonations…

… save as Donations

```
export interface RawDonation {
  amount: number;
  method: string;
  candidate: string;
  donor: string;
}

export interface Donation {
  amount: number;
  method: string;
  candidate: Candidate;
}
```

```
@inject(HttpClient, EventAggregator, Aurelia, Router)
export class DonationService {

  users: Map<string, User> = new Map();
  usersById: Map<string, User> = new Map()

  candidates: Candidate[] = [];
  donations: Donation[] = [];


  async getDonations() {
    const response = await this.httpClient.get('/api/donations');
    const rawDonations: RawDonation[] = await response.content;
    rawDonations.forEach(rawDonation => {
      const donation = {
        amount: rawDonation.amount,
        method: rawDonation.method,
        candidate: this.candidates.find(candidate =>
                                rawDonation.candidate == candidate._id),
        donor: this.usersById.get(rawDonation.donor)
      };
     this.donations.push(donation);
    });
  }
```

DonationService: donate

- Create a Candidate : retain copy of created candidate in candidates array
- Create donation: retain donation in donations array

```
@inject(HttpClient, EventAggregator, Aurelia, Router)

export class DonationService {
  users: Map<string, User> = new Map();
  usersById: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

  async createCandidate(firstName: string, lastName: string, office: string) {
    const candidate = {
      firstName: firstName,
      lastName: lastName,
      office: office
    };
    const response = await this.httpClient.post('/api/candidates', candidate);
    const newCandidate = await response.content;
    this.candidates.push(newCandidate);
  }

  async donate(amount: number, method: string, candidate: Candidate) {
    const donation = {
      amount: amount,
      method: method,
      candidate: candidate
    };
    const response = await this.httpClient.post('/api/candidates/' + candidate._id + '/donations', donation);
    this.donations.push(donation);
    this.total = this.total + amount;
    this.ea.publish(new TotalUpdate(this.total));
  }
```

DonationService: login

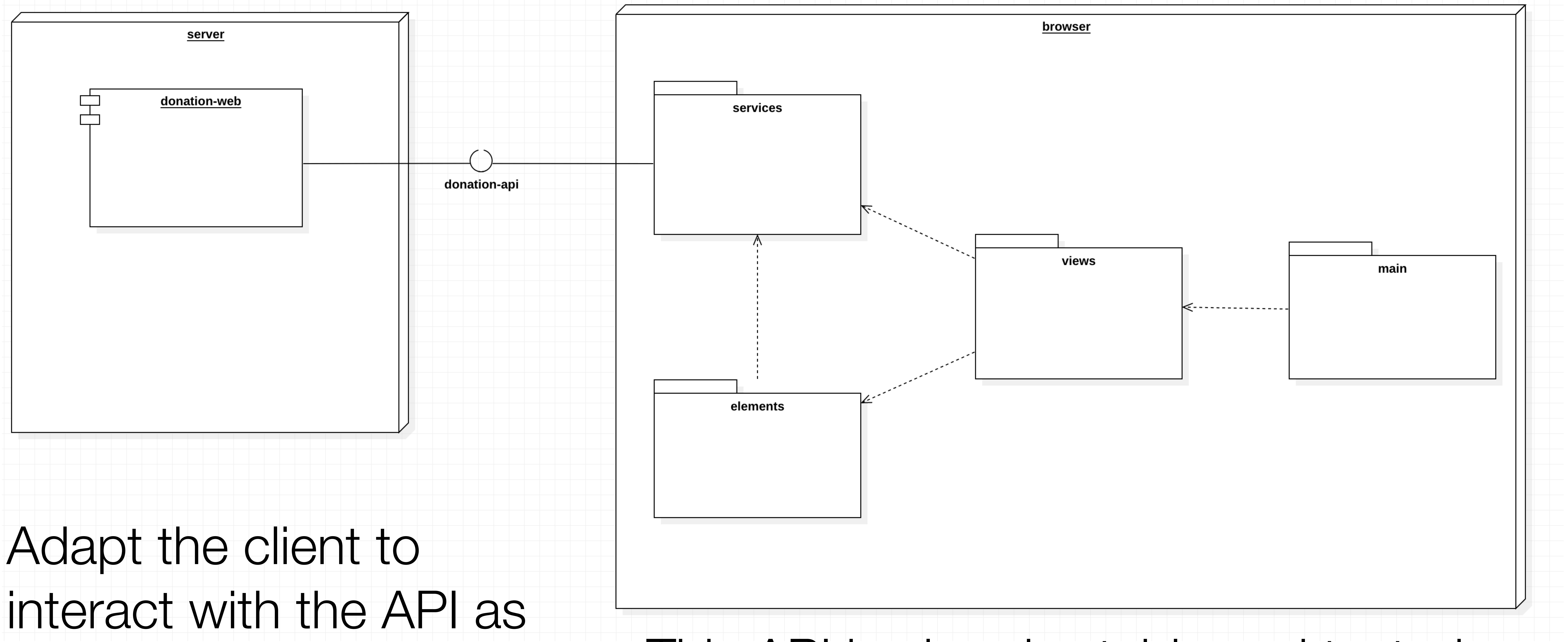
- If user match found - change router
- Change router method

```
@inject(HttpClient, EventAggregator, Aurelia, Router)

export class DonationService {
  users: Map<string, User> = new Map();
  usersById: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

  async login(email: string, password: string) {
    const user = this.users.get(email);
    if (user && user.password === password) {
      this.changeRouter(PLATFORM.moduleName('app'));
      return true;
    } else {
      return false;
    }
  }

  changeRouter(module: string) {
    this.router.navigate('/', { replace: true, trigger: false });
    this.router.reset();
    this.au.setRoot(PLATFORM.moduleName(module));
  }
```

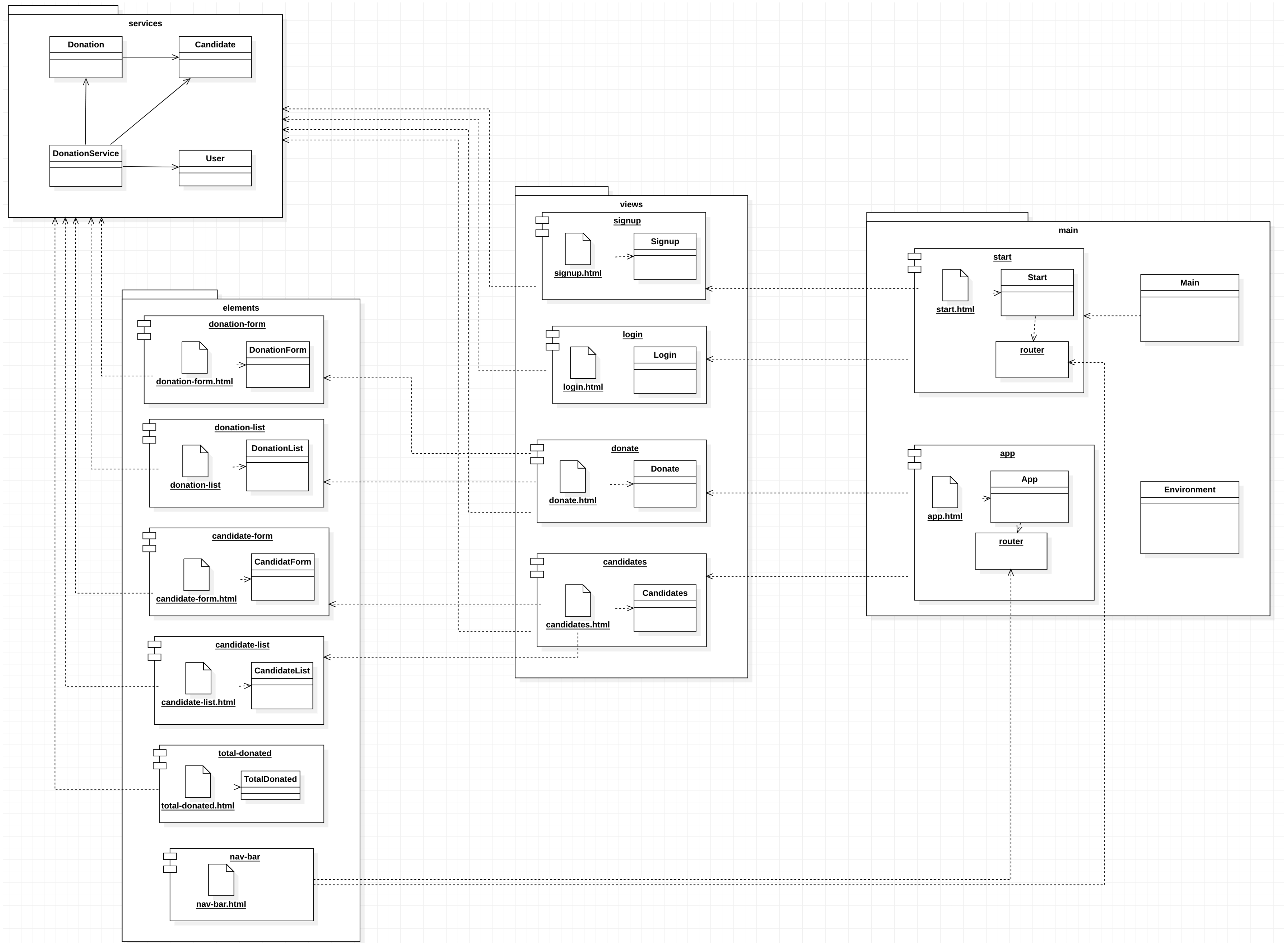
DonationService: signup

- Create new user
- Update local users map
- Change router to logged in user

```
@inject(HttpClient, EventAggregator, Aurelia, Router)

export class DonationService {
  users: Map<string, User> = new Map();
  usersById: Map<string, User> = new Map();
  candidates: Candidate[] = [];
  donations: Donation[] = [];
  paymentMethods = ['Cash', 'Paypal'];
  total = 0;

async signup(firstName: string, lastName: string, email: string, password:
  const user = {
    firstName: firstName,
    lastName: lastName,
    email: email,
    password: password
  };
  const response = await this.httpClient.post('/api/users', user);
  const newUser = await response.content;
  this.users.set(newUser.email, newUser);
  this.usersById.set(newUser._id, newUser);
  this.changeRouter(PLATFORM.moduleName('app'))
  return false;
}
```



- Adapt the client to interact with the API as developed in the Hapi web application
- This API is already stable and tested


services
Donation
Candidate
DonationService
User
views
signup
signup.html
Signup
login
login.html
Login
donate
donate.html
Donate
candidates
candidates.html
Candidates
main
start
start.html
Start
router
Main
app
app.html
App
router
Environment
elements
donation-form
donation-form.html
DonationForm
donation-list
donation-list
DonationList
candidate-form
candidate-form.html
CandidatForm
candidate-list
candidate-list.html
CandidateList
total-donated
total-donated.html
TotalDonated
nav-bar
nav-bar.html