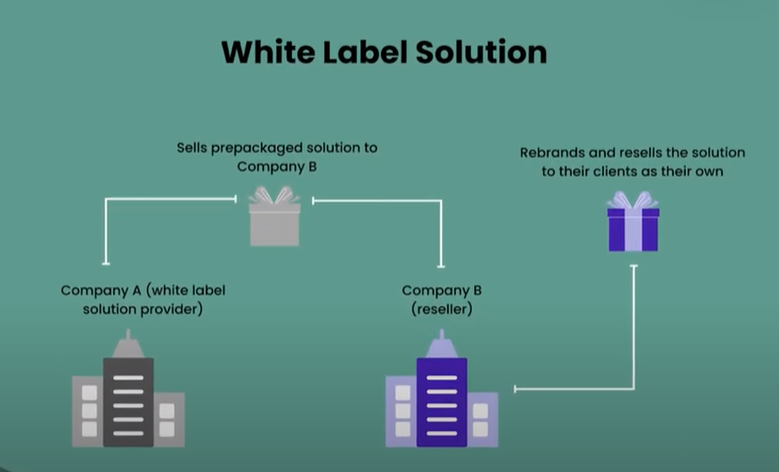
## Document terms:

1. partner-app, white-labelled app: web application (SAAS) through which Law-On-Earth(LOE)’s partner will resell the services they bought under their own brand
2. core app: web application serving as main dashboard, where partners will go and create/manage their white-labelled apps.

## Project goal

* White-labelling: allowing LOE’s partners to offers SAAS services under their own brand, without developing a platform from scratch



## Project key requirements

* Creating a white-labelled app, deployable under custom (sub-)domains that will be provided by each partner. Note: each white-labelled app will buy monthly services from the core app, and resell it to their end-users.
* We need to consider for two types of people:
  + partners that will have their own dev team, so they only want access to the API in order to integrate it to the frontend they will build themselves
  + partners that will not have a dev team, and would like to deploy our pre-built customizable frontend. This pre-built frontend should be customizable on many levels such as: design (look and feel), landing page content.
* After the project is put into production: when a partner is creating and deploying a white-labelled app, they need to be able to do it without relying on a tech team, and also without technical knowledge. From an end-user perspective, they will click on a button to have their app available.
* Sharable resources between core app and white-labelled apps.
  + Example 1: LOE, as admin, will upload a file in the core app, and that file needs to be made available to some white-labelled apps that want to resell it.
  + Example 2: LOE, as admin, may create an entry in the database on the core app, and that entry needs to be made available to some white-labelled apps that meet a given condition.
* Custom professional email for each white-labelled app; the API will use that email to send messages to the end-users.
* Other considerations (not discussed with LOE, but will likely being asked):
  + having a desktop version of the core app, and maybe also of the partner app

## Considered architectures

Kindly refer to draw.io files in [our google drive folder](https://drive.google.com/drive/folders/1o1XWCEYScYcfxgefIaFJ8l8GGJlbD9r1?usp=sharing) (kindly manage access through Rasha)

## Architectures we tried (first stage) and are using (second stage)

### First stage: custom domain names mapped toward one white-labelled frontend app

Key components:

* one frontend deployed on AWS running under a domain name A
* A white-labelled X app will have a custom domain Bx that will be mapped to A
* end users will access white-labelled app X under domain name Bx , but behind the scenes they are accessing the app deployed under domain name A.

The how:

* leveraging route53 key for domain name mapping between original domain name A, and domain names Bx

The blockers:

* Require knowledge in domain name routing, that team members don’t have
* Require intensive research. Furthermore:
  + successful outcome cannot be guaranteed
  + duration cannot be guaranteed
* Key takeaway from the above: lack of control on the project’s success

### Second stage: the white-labelled’s frontend build deployed toward a remote web host through ssh

Key components:

1. a compressed built of the white-labelled frontend
2. ssh access to remote web server

The how:

1. Having a compressed build of the white-labelled app stored on S3
2. The partner provides in in the core app dashboard the ssh credentials to access his web server
   1. For now, only login/password is supported, to make it easier to non-tech guys to have them
   2. Later, will also add file-based ssh system
3. When a partner clicks on “deploy” button on the core app dashboard, the compressed build is copied from s3, and put into the remote host then uncompressed

## Security concerns from API side: avoiding a white-labelled app accessing another one’s data

1. Each white-labelled frontend app is identified by:
   1. an ID: COMPANY-CODE, in the headers
   2. the (sub-)domain it is deployed to:
      1. HTTP requests coming from unknown domains will be denied
      2. HTTP requests coming from known domain, but without correct headers information will be denied
2. Each frontend is identified by an ID: FRONTEND--KEY, in the headers. (will help make difference between frontend built by us and frontend built by a partner own dev team)
3. Future plan: white-labelling the partner-app API
   1. How: having a custom subdomains for the partner-app API (ie: [https://[UNIQUE-CODE].apidomain.com](about:blank) )
      1. Each subdomain is dedicated to one white-labelled frontend usage.
      2. All sub-domains point toward the same API.

Key bottleneck:

1. How does a deployed frontend access/grab its credentials (COMPANY--CODE, FRONTEND--KEY), especially for the first time ?
   1. Current approach: leveraging environment variables that will be present on the remote host OS.
   2. How: the partner will copy from his dashboard on the core app the environment variable to paste on the admin dashboard provided by the remote host.

## Project resources

* API code : <https://github.com/OmdenaAI/LawOnEarth-back-end/tree/restructuring.dev.ekoue>
* API documentation (description is yet to be completed, as time is focused to deliver new endpoints):
  + Link: <https://www.apidog.com/apidoc/shared-5bab9a98-f313-440e-a311-d5f0983afa1d>.
  + Access password: azertyuiop
* UML diagrams (kindly open with ASTAH) can be found in our google drive folder: <https://drive.google.com/drive/folders/1tbs1Lr5djN353WO6d3qXO1yC2YYCqHD6?usp=sharing> (kindly refer to Rasha for access link)
* Frontend code: <https://github.com/OmdenaAI/LawOnEarth-front-end>