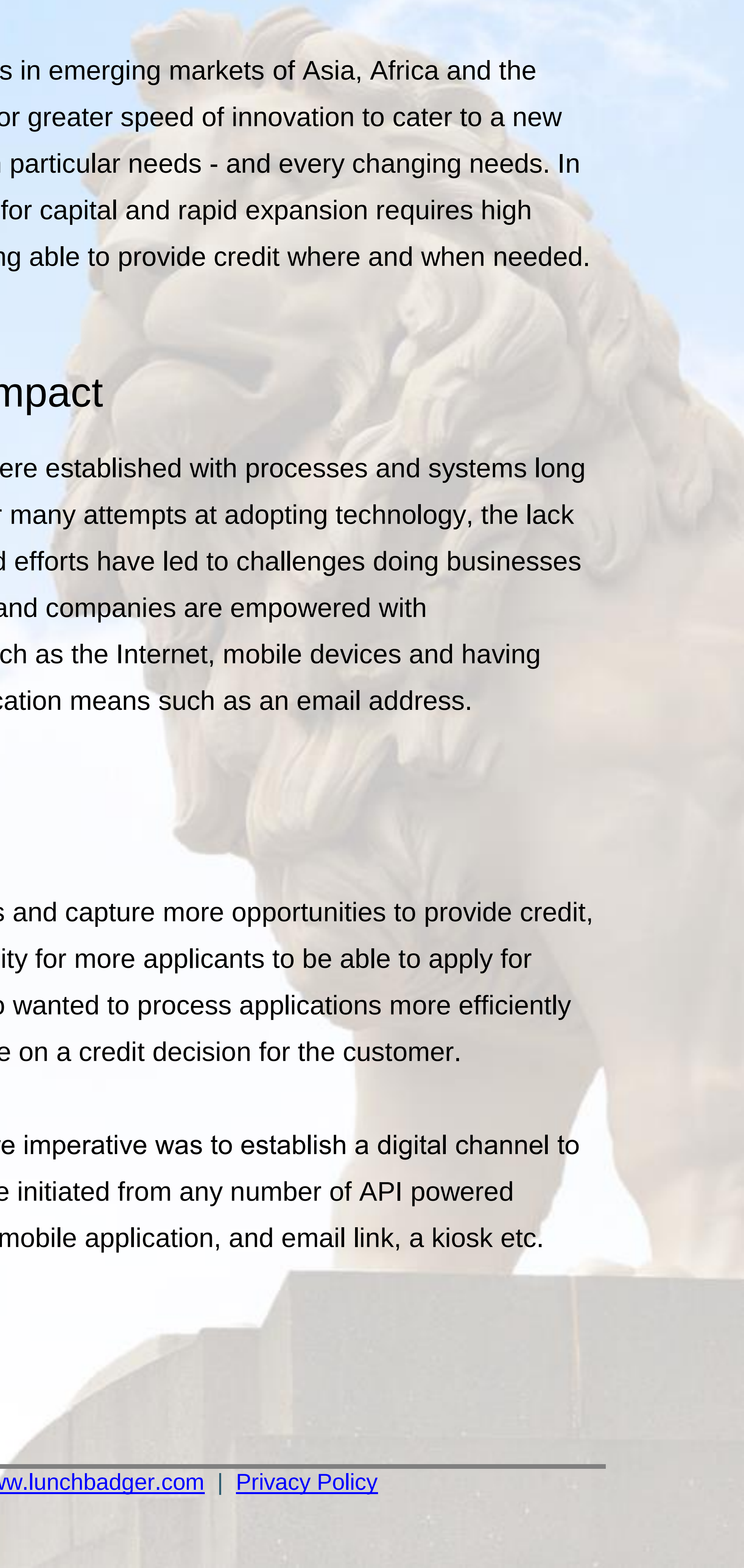
LunchBadger

# Credit Application Automation for a Multi-National Bank

A Use Case For

express serverless platform



## Background

A multinational bank operates in emerging markets of Asia, Africa and the Middle East with a demand for greater speed of innovation to cater to a new generation of customers with particular needs - and every changing needs. In emerging markets, the need for capital and rapid expansion requires high availability and fluidity of being able to provide credit where and when needed.

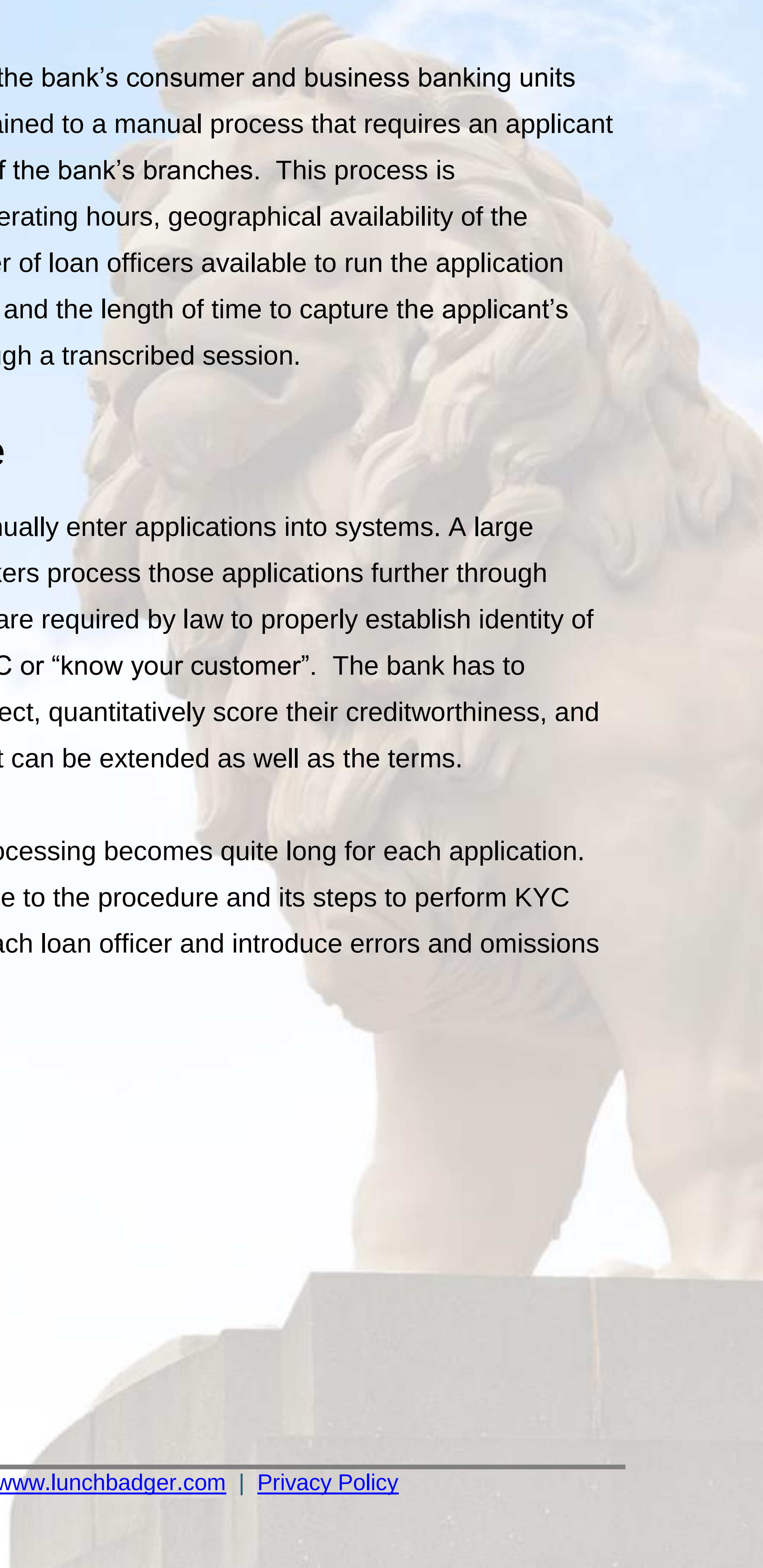
## Banking Industry Impact

The vast majority of banks were established with processes and systems long before the cloud era.  In their many attempts at adopting technology, the lack of standards and coordinated efforts have led to challenges doing businesses with the bank as individuals and companies are empowered with consumerized technology such as the Internet, mobile devices and having digital identity and communication means such as an email address.

## Core Imperative

To service emerging markets and capture more opportunities to provide credit, this bank needed the capability for more applicants to be able to apply for credit digitally. The bank also wanted to process applications more efficiently to reduce the turnaround time on a credit decision for the customer.

The multi national bank's core imperative was to establish a digital channel to allow credit applications to be initiated from any number of API powered experiences such as web, a mobile application, and email link, a kiosk etc.



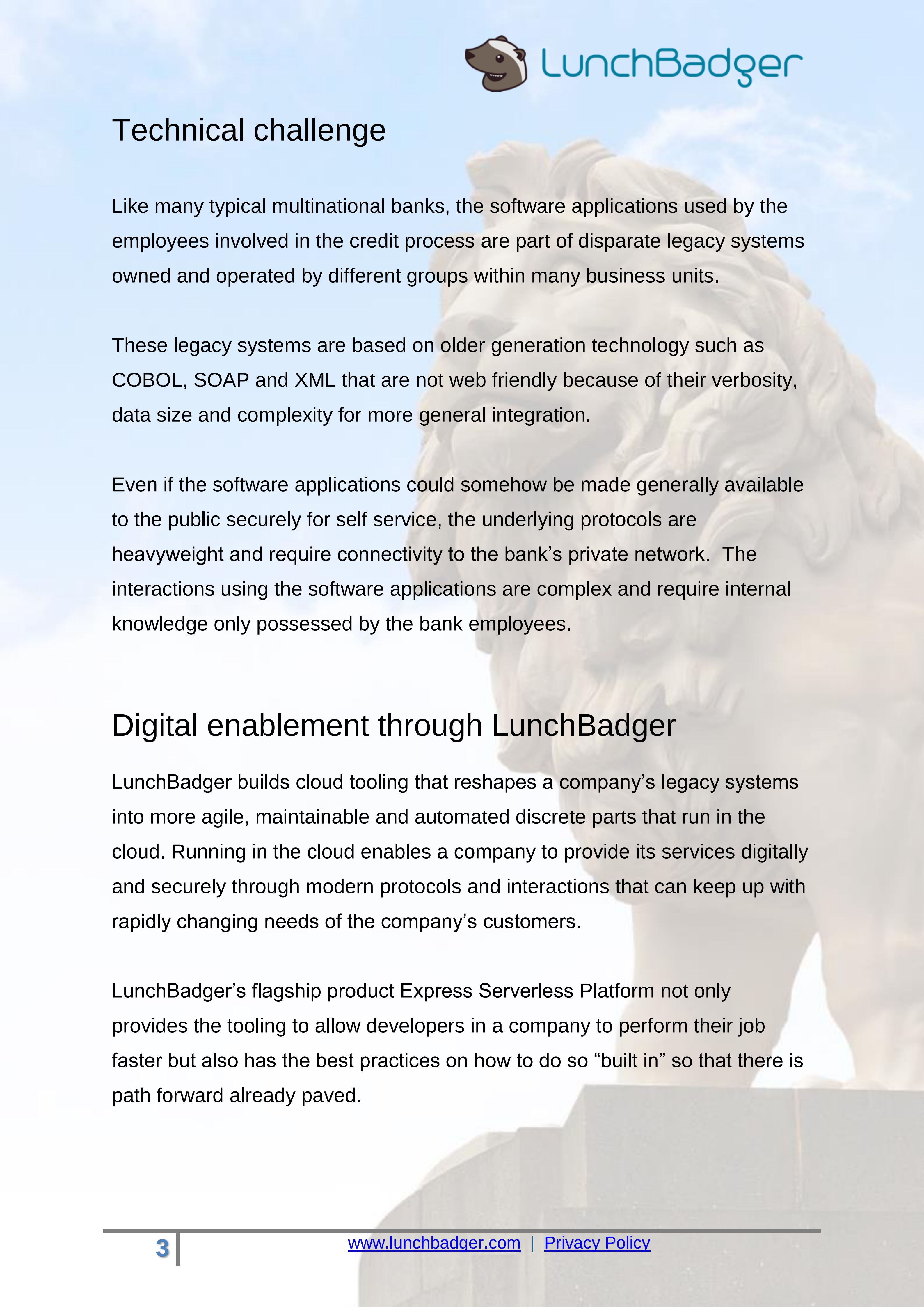
## Customer impact

Credit applications across the bank's consumer and business banking units are currently highly constrained to a manual process that requires an applicant to apply in person at one of the bank's branches.  This process is bottlenecked by branch operating hours, geographical availability of the nearest branch, the number of loan officers available to run the application process at a given branch, and the length of time to capture the applicant's disclosed information through a transcribed session.

## Manual challenge

A small set of workers manually enter applications into systems. A large number of back office workers process those applications further through operating procedures that are required by law to properly establish identity of the prospect known as KYC or "know your customer".  The bank has to properly qualify their prospect, quantitatively score their creditworthiness, and determine how much credit can be extended as well as the terms.

The manual application processing becomes quite long for each application. Furthermore, the adherence to the procedure and its steps to perform KYC can vary vastly between each loan officer and introduce errors and omissions while being conducted.



## Technical challenge

Like many typical multinational banks, the software applications used by the employees involved in the credit process are part of disparate legacy systems owned and operated by different groups within many business units.

These legacy systems are based on older generation technology such as COBOL, SOAP and XML that are not web friendly because of their verbosity, data size and complexity for more general integration.

Even if the software applications could somehow be made generally available to the public securely for self service, the underlying protocols are heavyweight and require connectivity to the bank’s private network. The interactions using the software applications are complex and require internal knowledge only possessed by the bank employees.

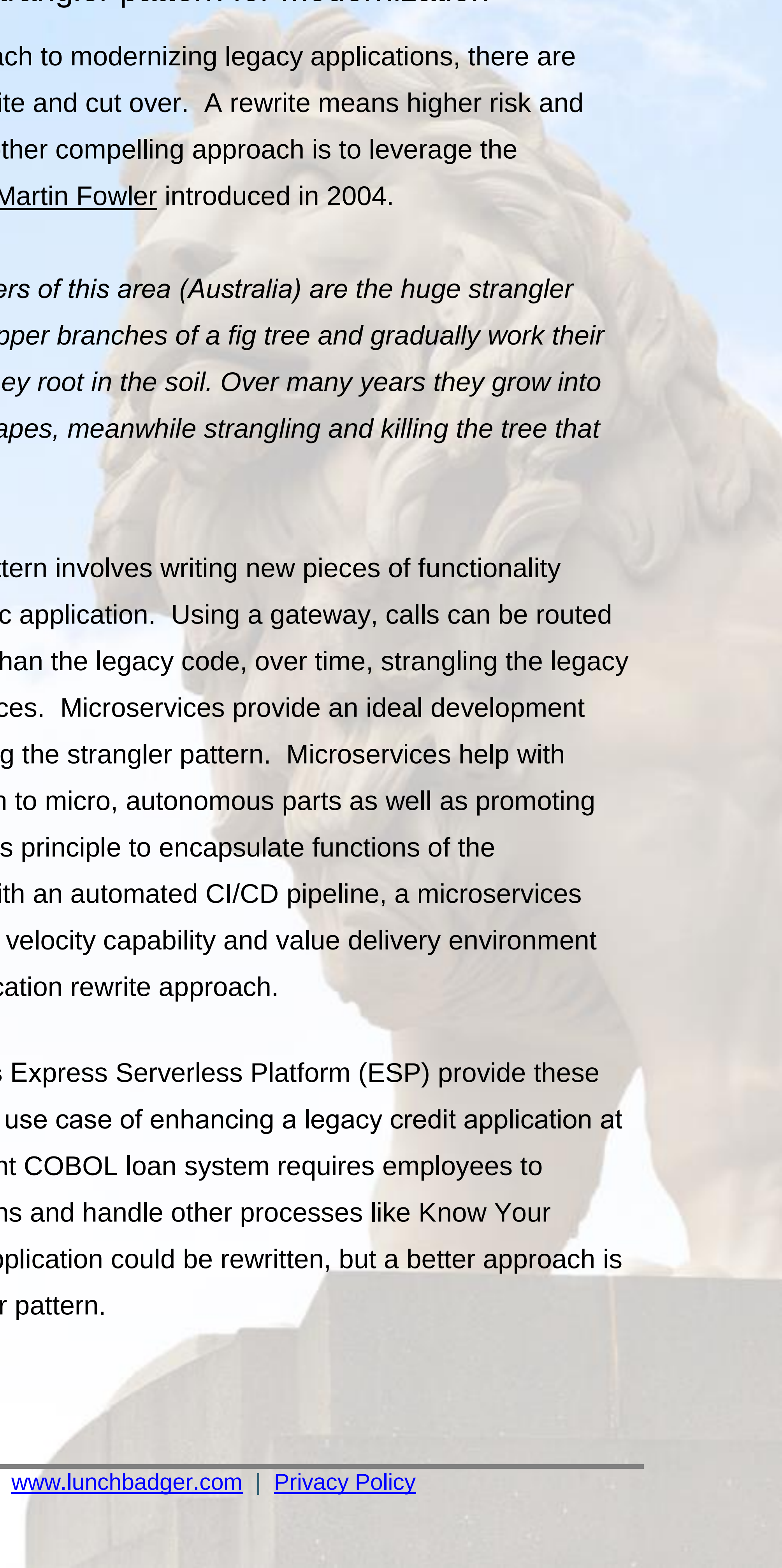
## Digital enablement through LunchBadger

LunchBadger builds cloud tooling that reshapes a company’s legacy systems into more agile, maintainable and automated discrete parts that run in the cloud. Running in the cloud enables a company to provide its services digitally and securely through modern protocols and interactions that can keep up with rapidly changing needs of the company’s customers.

LunchBadger’s flagship product Express Serverless Platform not only provides the tooling to allow developers in a company to perform their job faster but also has the best practices on how to do so “built in” so that there is path forward already paved.

Express Serverless Platform takes the buzzwords and concepts described in microservices, APIs, and serverless and make them concrete and real for practical utilization in the everyday tasks performed by developers and operations.

Express Serverless Platform runs in any private or public cloud. It's runtime takes full advantage of infrastructure resources while masking their complexity from the user.  Getting Express Serverless Platform running is a simple one step deployment to establish the bank's footprint in the cloud.



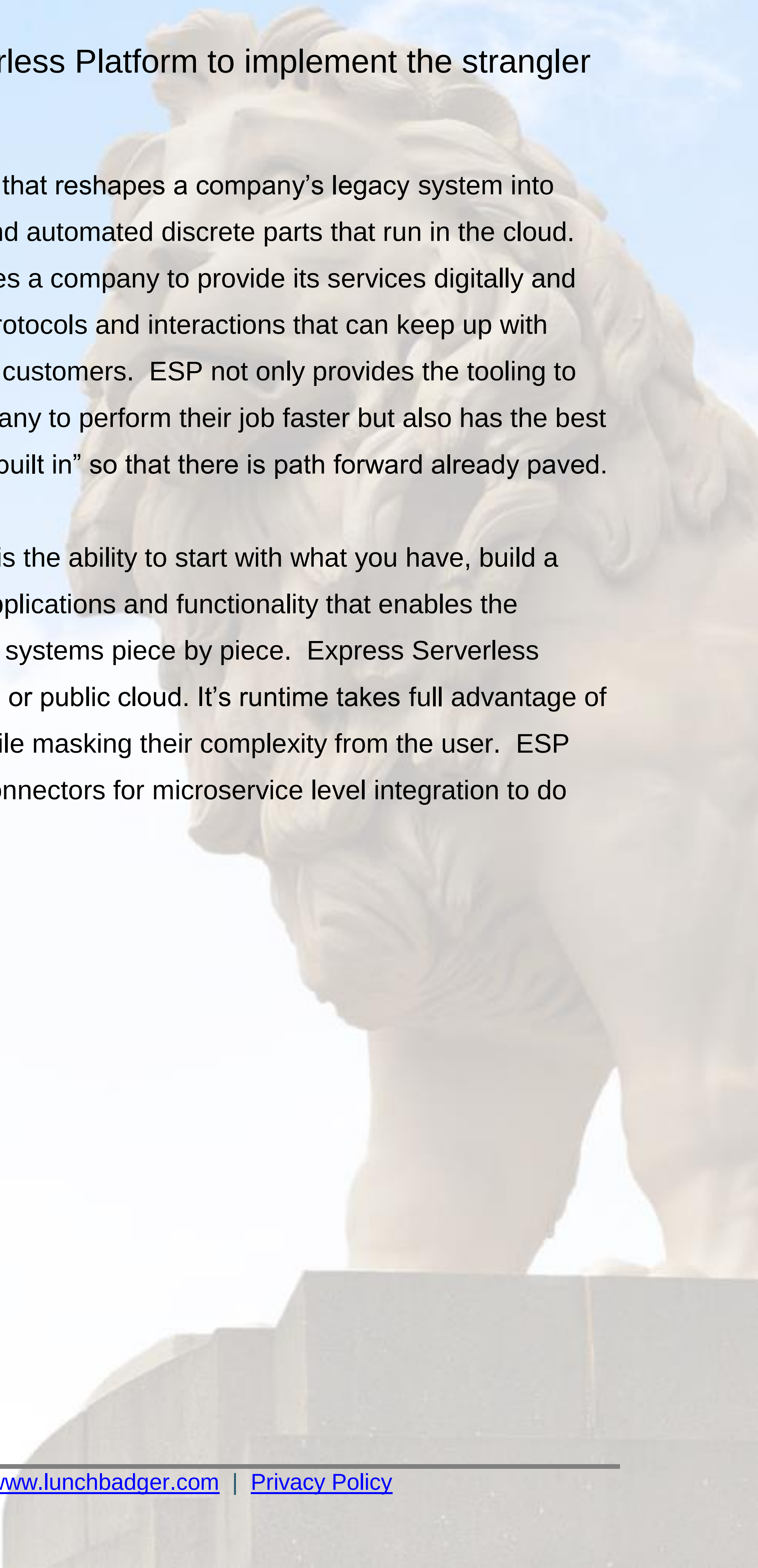
## Part 1: considering strangler pattern for modernization

When planning an approach to modernizing legacy applications, there are alternatives to a full re-write and cut over.  A rewrite means higher risk and longer time to value.  Another compelling approach is to leverage the "strangler" pattern which Martin Fowler introduced in 2004.

*"One of the natural wonders of this area (Australia) are the huge strangler vines. They seed in the upper branches of a fig tree and gradually work their way down the tree until they root in the soil. Over many years they grow into fantastic and beautiful shapes, meanwhile strangling and killing the tree that was their host."*

Applying the strangler pattern involves writing new pieces of functionality surrounding the monolithic application.  Using a gateway, calls can be routed to the new pieces rather than the legacy code, over time, strangling the legacy with newly developed pieces.  Microservices provide an ideal development paradigm for implementing the strangler pattern.  Microservices help with breaking down application to micro, autonomous parts as well as promoting the separation of concerns principle to encapsulate functions of the application.  Combined with an automated CI/CD pipeline, a microservices approach provides a high velocity capability and value delivery environment versus a monolithic application rewrite approach.

How does LunchBadger's Express Serverless Platform (ESP) provide these capabilities?  Let's take a use case of enhancing a legacy credit application at a global bank.  The current COBOL loan system requires employees to manually enter applications and handle other processes like Know Your Customer.  The legacy application could be rewritten, but a better approach is to implement the strangler pattern.

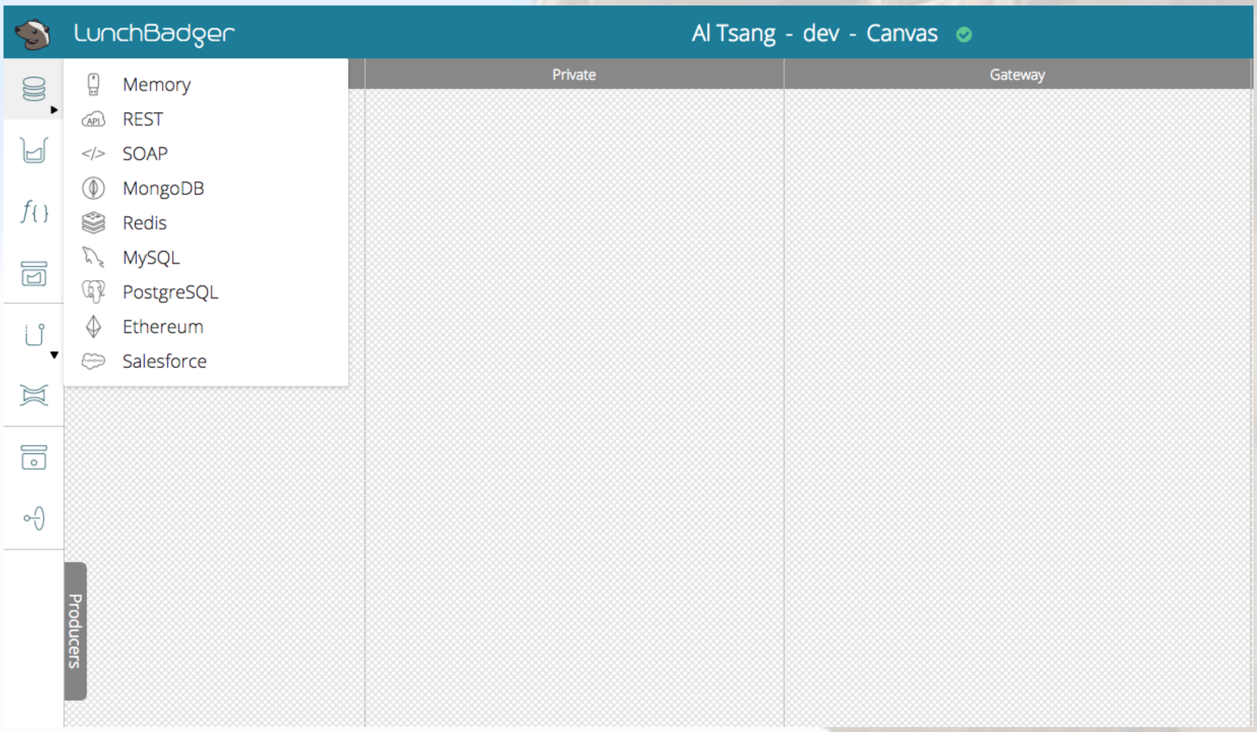


## Part 2: Express Serverless Platform to implement the strangler pattern

ESP provides cloud tooling that reshapes a company’s legacy system into more agile, maintainable and automated discrete parts that run in the cloud. Running in the cloud enables a company to provide its services digitally and securely through modern protocols and interactions that can keep up with rapidly changing needs of t customers.  ESP not only provides the tooling to allow developers in a company to perform their job faster but also has the best practices on how to do so “built in” so that there is path forward already paved.

A key differentiator of ESP is the ability to start with what you have, build a new set of modern cloud applications and functionality that enables the eventual retirement of such systems piece by piece.  Express Serverless Platform runs in any private or public cloud. It’s runtime takes full advantage of infrastructure resources while masking their complexity from the user.  ESP comes with a vast set of Connectors for microservice level integration to do this.

The Model Connectors shown below are a small subset of what is actually available. Two Model Connectors in particular allow developers to leverage existing services within enterprise legacy systems - the REST and SOAP Connectors. The bank's credit application utilizes the Customer SOAP service. This SOAP service represents all customers in the bank and contains more than 160 fields of information. With a click of a button Express Serverless Platform can easily connect to such SOAP service and provide a set of tools to interface with it.



## Part 3: Introduce a new clean microservice layer

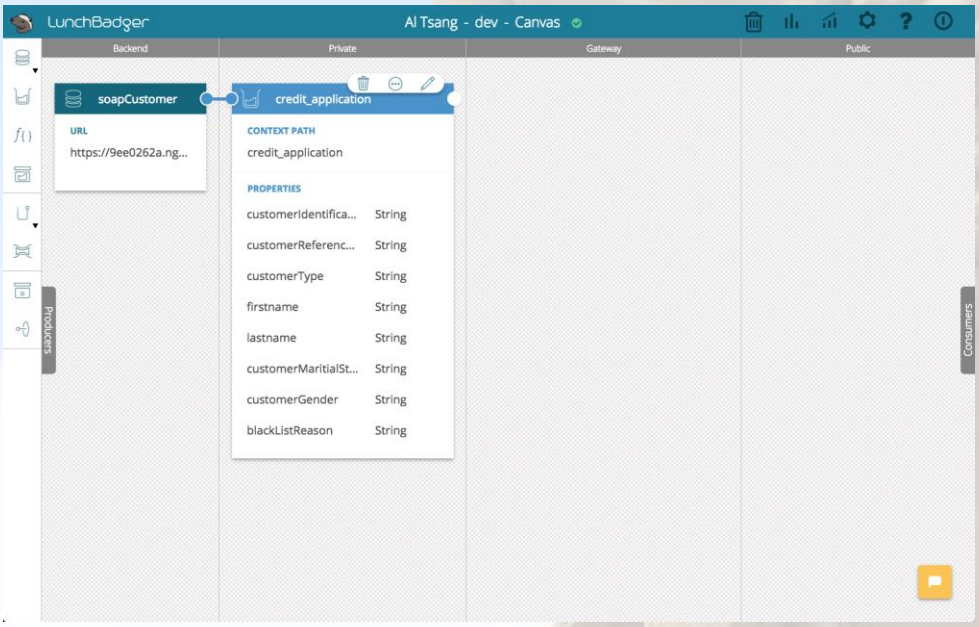
ESP has special functions known as Models.  Models are written solely in Node.js/JavaScript.  Node.js is an extremely powerful technology that is particularly well suited for not only writing function based microservices, but also doing asynchronous and synchronous integration to multiple systems.

A Model is a function that "models" a real life object and has built in behavior that is commonly required in most enterprise application use cases.

For this use case, when capturing a new customer prospect in a credit application, most of the fields within the customer SOAP service are not relevant at the time.

To introduce a new clean microservice to meet the credit application use case, a **credit_application** model can be created with only the few required fields that are necessary to expose when capturing this information externally as an API.

The **credit_application** model below is connected to the **soapCustomer** model connector that allows it to interface with the legacy customer SOAP service.  The **soapCustomer** connector can surface only the details that are required for this microservice - employment and contact details.  Separate model functions called **employment_details** and **customer_details** can be created that automatically call the SOAP service and the associated operations to get these details.



soapCustomer

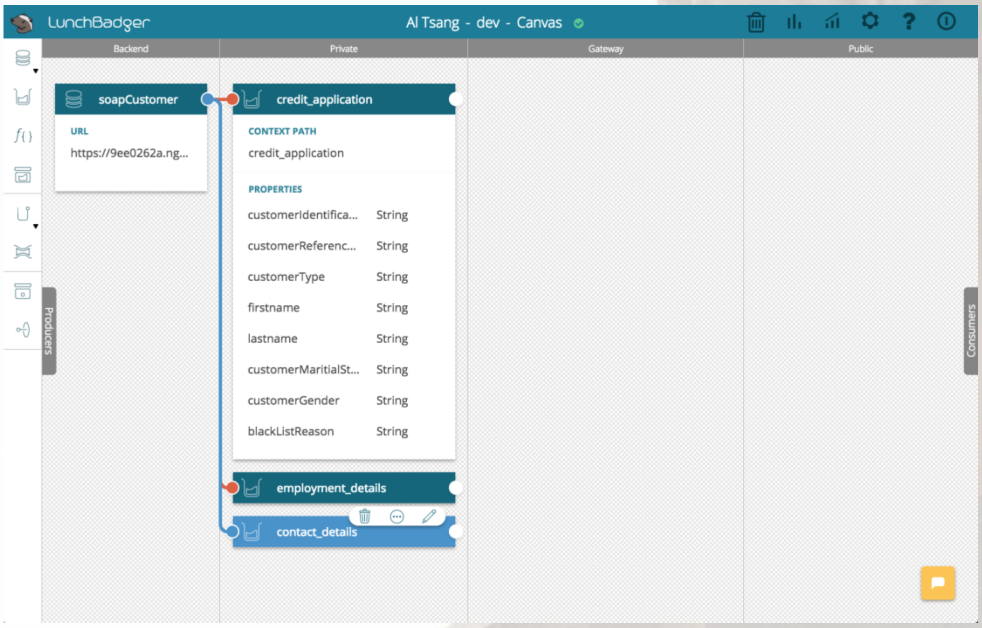
URL
https://9ee0262a.ng...

credit_application

CONTEXT PATH
credit_application

PROPERTIES

| | |
|---|---|
| customerIdentifica... | String |
| customerReferenc... | String |
| customerType | String |
| firstname | String |
| lastname | String |
| customerMaritialSt... | String |
| customerGender | String |
| blackListReason | String |



soapCustomer

URL
https://9ee0262a.ng...

credit_application

CONTEXT PATH
credit_application

PROPERTIES

| | |
|---|---|
| customerIdentifica... | String |
| customerReferenc... | String |
| customerType | String |
| firstname | String |
| lastname | String |
| customerMaritialSt... | String |
| customerGender | String |
| blackListReason | String |

employment_details

contact_details

Having these as separate models allows us to cleanly decouple the **credit_application** from the supporting details.

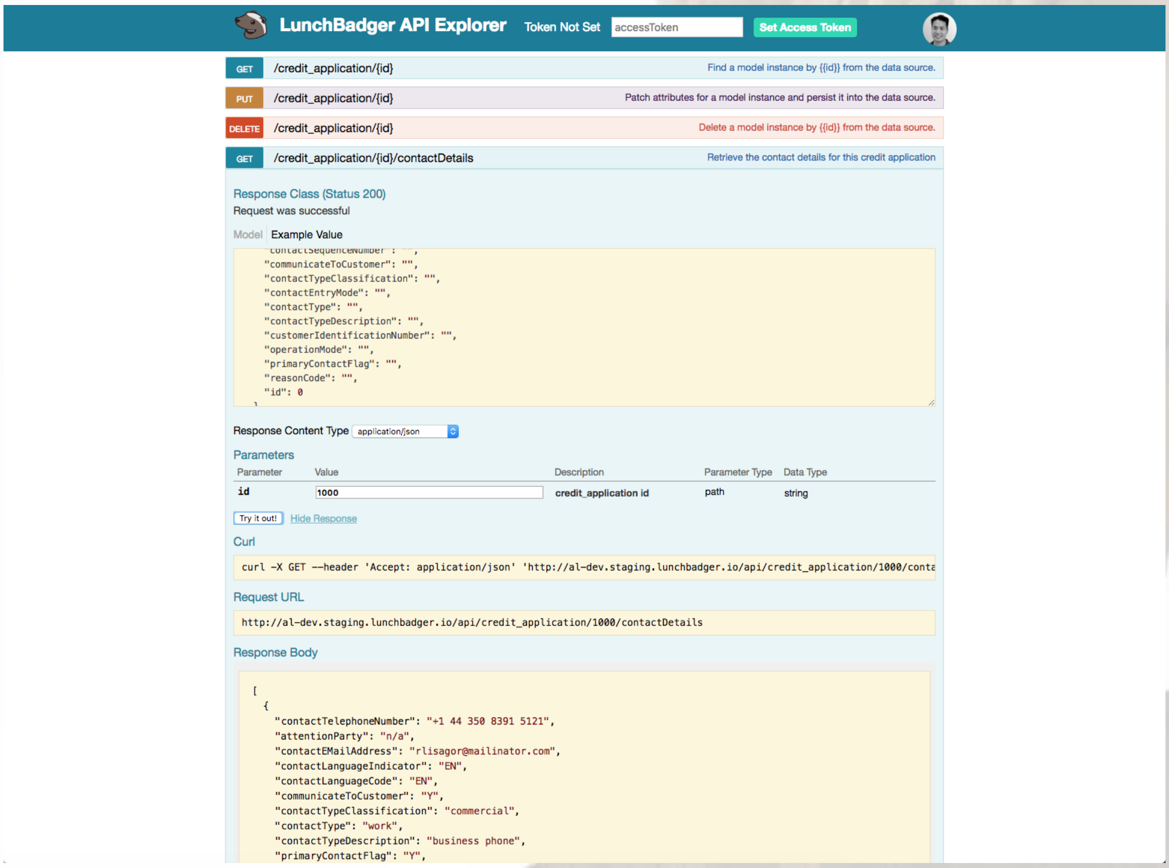
The REST API interface generated can retrieve just the credit application:

```
GET /credit_application/{id}
```

and it can also dynamically retrieve either of the associated details

```
GET /credit_application/{id}/employment_details
GET /credit_application/{id}/contact_details
```

Express Serverless Platform generates not only the REST interface but comes with an API explorer that can be embedded in any corporate portal. The REST interface that is scaffolded also has an associated OpenAPI specification generated with it.



GET /credit_application/{id} Find a model instance by {{id}} from the data source.

PUT /credit_application/{id} Patch attributes for a model instance and persist it into the data source.

DELETE /credit_application/{id} Delete a model instance by {{id}} from the data source.

GET /credit_application/{id}/contactDetails Retrieve the contact details for this credit application

Response Class (Status 200)

Request was successful

Model | Example Value

```
    "communicateToCustomer": "",
    "contactTypeClassification": "",
    "contactEntryMode": "",
    "contactType": "",
    "contactTypeDescription": "",
    "customerIdentificationNumber": "",
    "operationMode": "",
    "primaryContactFlag": "",
    "reasonCode": "",
    "id": 0
```

Response Content Type application/json

Parameters

| Parameter | Value | Description | Parameter Type | Data Type |
|---|---|---|---|---|
| id | 1000 | credit_application id | path | string |

Try it out! Hide Response

Curl

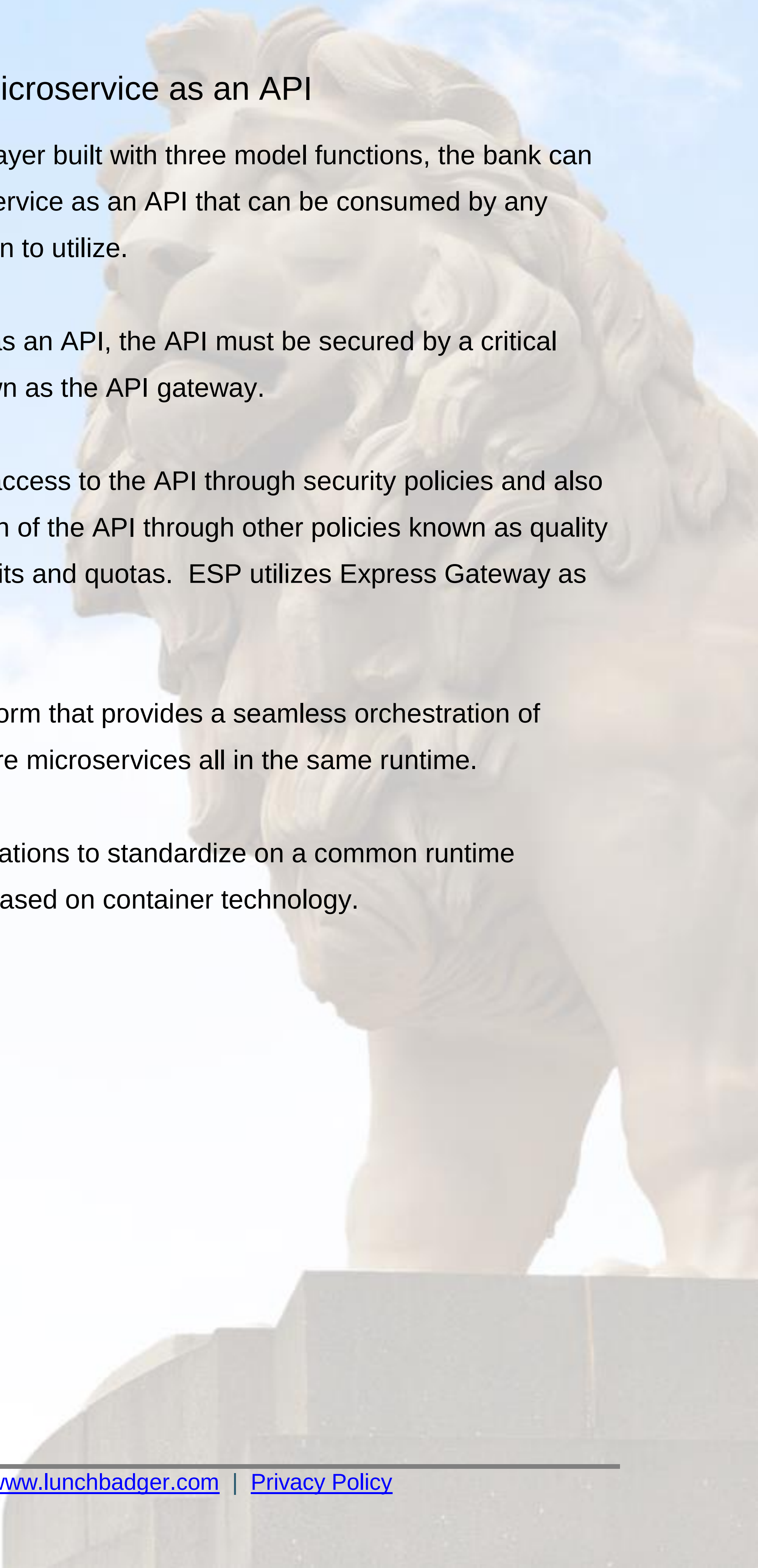
```
curl -X GET --header 'Accept: application/json' 'http://al-dev.staging.lunchbadger.io/api/credit_application/1000/conta
```

Request URL

```
http://al-dev.staging.lunchbadger.io/api/credit_application/1000/contactDetails
```

Response Body

```
[
  {
    "contactTelephoneNumber": "+1 44 350 8391 5121",
    "attentionParty": "n/a",
    "contactEMailAddress": "rlisagor@mailinator.com",
    "contactLanguageIndicator": "EN",
    "contactLanguageCode": "EN",
    "communicateToCustomer": "Y",
    "contactTypeClassification": "commercial",
    "contactType": "work",
    "contactTypeDescription": "business phone",
    "primaryContactFlag": "Y",
```



## Part 4: Expose your microservice as an API

With a clean microservice layer built with three model functions, the bank can now externalize the microservice as an API that can be consumed by any digital channel that they plan to utilize.

To expose a microservice as an API, the API must be secured by a critical piece of infrastructure known as the API gateway.

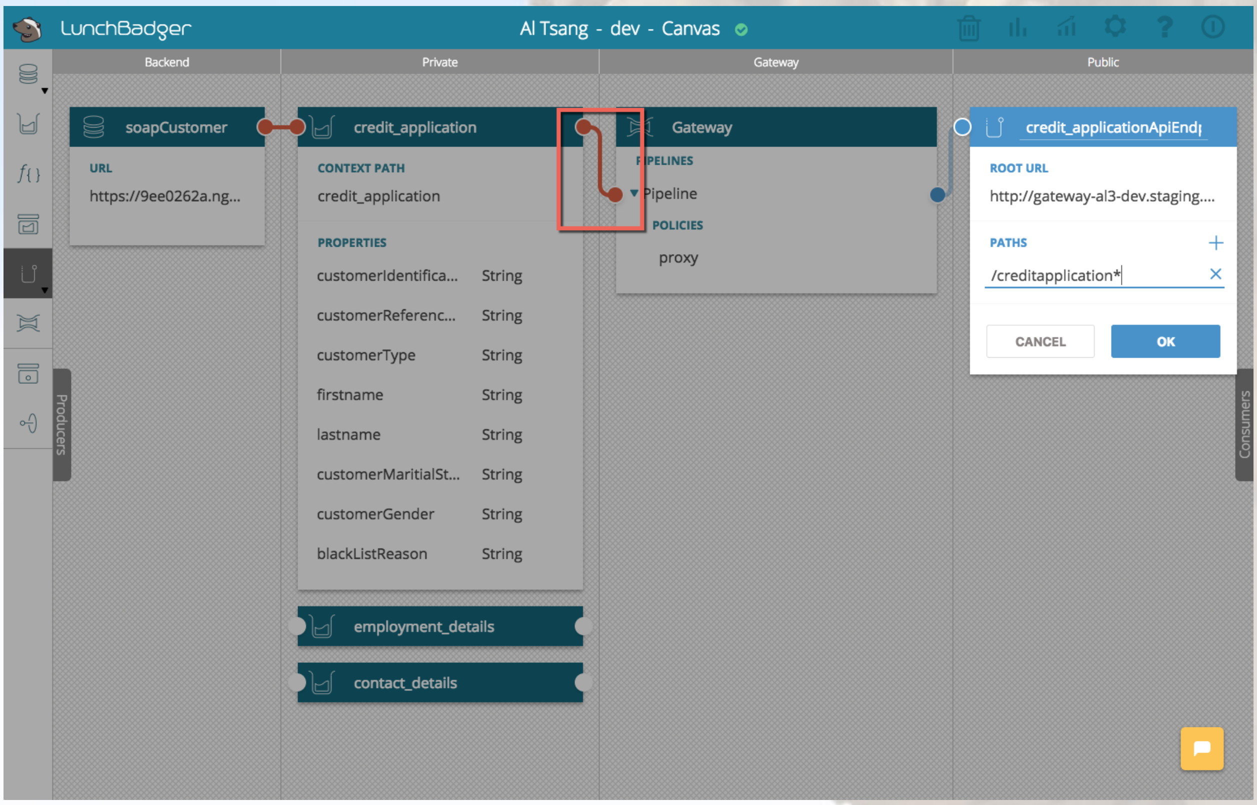
The API gateway controls access to the API through security policies and also can control the consumption of the API through other policies known as quality of services such as rate limits and quotas.  ESP utilizes Express Gateway as its built in API gateway.

ESP is the first of kind platform that provides a seamless orchestration of application and infrastructure microservices all in the same runtime.

This allows the bank's operations to standardize on a common runtime natively built for the cloud based on container technology.

In Express Serverless Platform, deploying and configuring an API gateway is a click of a button and utilizes the same Canvas to illustrate its relation to the microservice.

Once a gateway is deployed, a microservice can be exposed as an API and a set of endpoints by simply dragging a line from the model to the gateway's pipeline.



soapCustomer
URL
https://9ee0262a.ng...

credit_application
CONTEXT PATH
credit_application
PROPERTIES

|  |  |
|---|---|
| customerIdentifica... | String |
| customerReferenc... | String |
| customerType | String |
| firstname | String |
| lastname | String |
| customerMaritialSt... | String |
| customerGender | String |
| blackListReason | String |

employment_details

contact_details

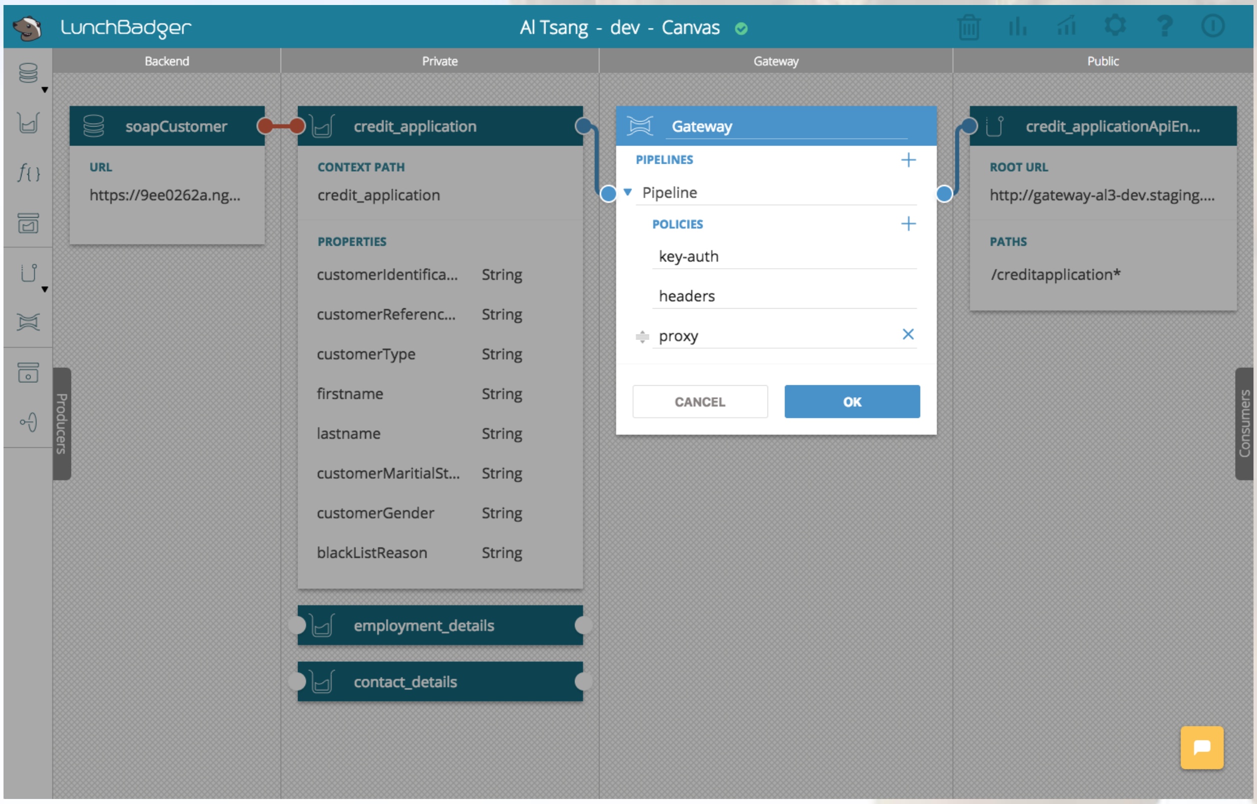
Gateway
PIPELINES
Pipeline
POLICIES
proxy

credit_applicationApiEnd
ROOT URL
http://gateway-al3-dev.staging....
PATHS
/creditapplication*

## Part 5: Secure and manage your APIs through pipelines and policies

Express Gateway is easily configured through pipelines. A pipeline is a set of policies that can be customized to control access to the API.  A single Express Gateway instance can be configured with multiple pipelines.

The bank can quickly configure multiple policies including key authorization and header manipulation policies to shape the API request to be universally consumed by the **credit_application** microservice that it proxied to.



soapCustomer
URL
https://9ee0262a.ng...

credit_application
CONTEXT PATH
credit_application
PROPERTIES

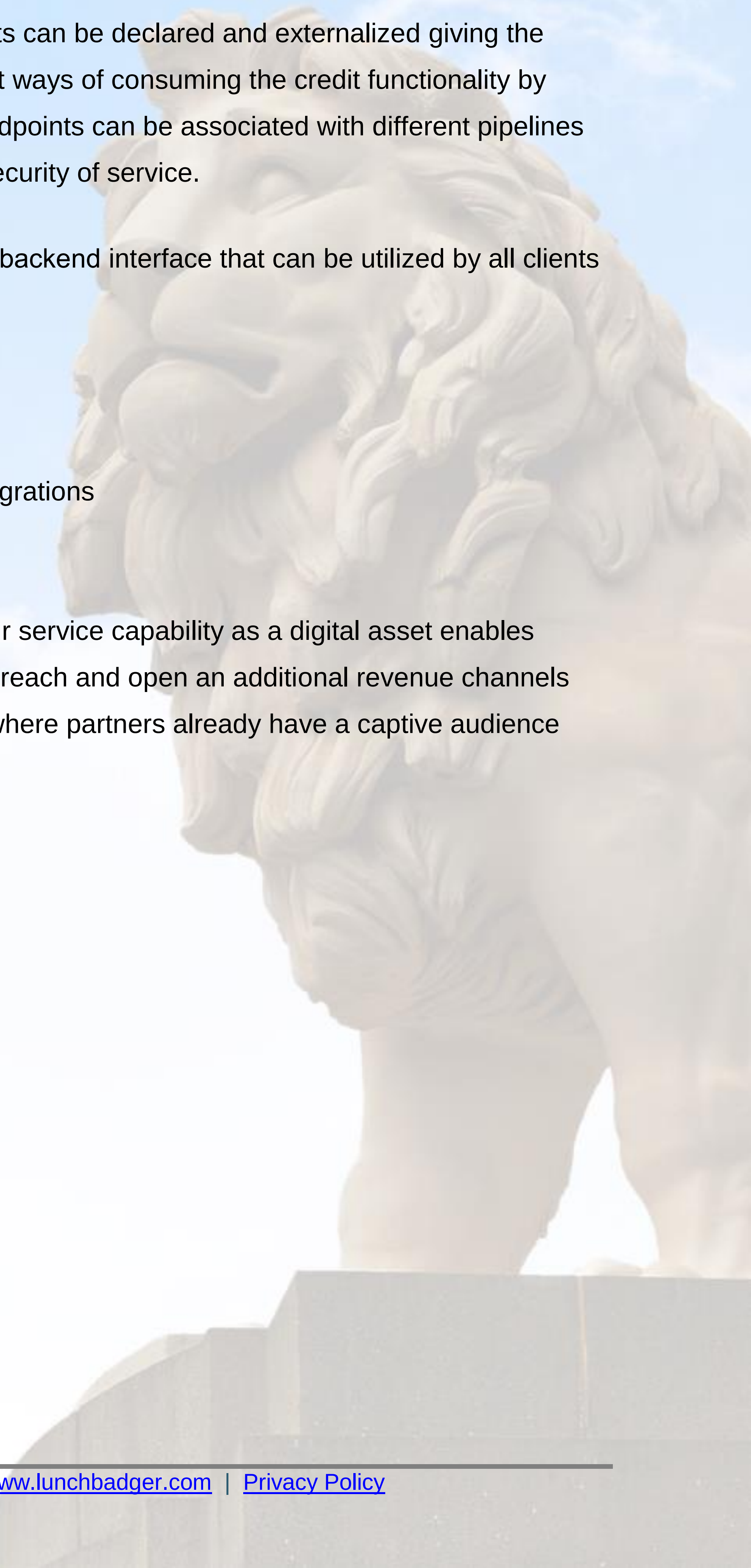
| | |
|---|---|
| customerIdentifica... | String |
| customerReferenc... | String |
| customerType | String |
| firstname | String |
| lastname | String |
| customerMaritialSt... | String |
| customerGender | String |
| blackListReason | String |

employment_details

contact_details

Gateway
PIPELINES
Pipeline
POLICIES
key-auth
headers
proxy

credit_applicationApiEn...
ROOT URL
http://gateway-al3-dev.staging....
PATHS
/creditapplication*

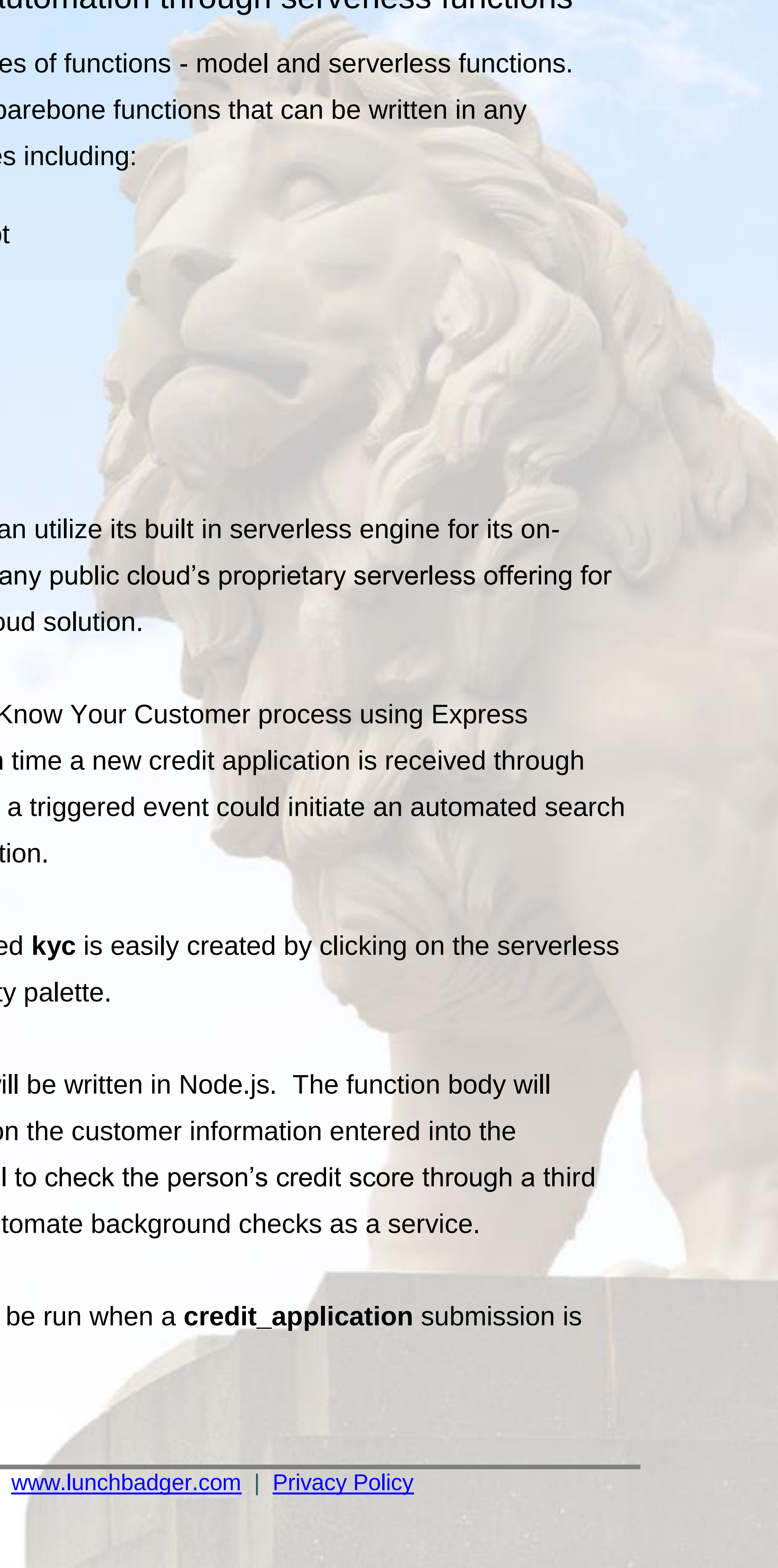


Any number of API endpoints can be declared and externalized giving the flexibility of creating different ways of consuming the credit functionality by different clients.  A set of endpoints can be associated with different pipelines to meet differing needs of security of service.

APIs are the “omnichannel” backend interface that can be utilized by all clients including:

- web applications
- mobile clients
- system to system integrations
- devices

Digitizing and extending their service capability as a digital asset enables them to extend their market reach and open an additional revenue channels within specialized markets where partners already have a captive audience and customer base.



## Part 6: On-demand automation through serverless functions

ESP has two different types of functions - model and serverless functions. Serverless functions are barebone functions that can be written in any commonly used languages including:

- Node.js / JavaScript
- Python
- Ruby
- Go
- .NET Core
- PHP
- Java

Through ESP, the bank can utilize its built in serverless engine for its on-premise deployment and any public cloud's proprietary serverless offering for a true hybrid and multi-cloud solution.

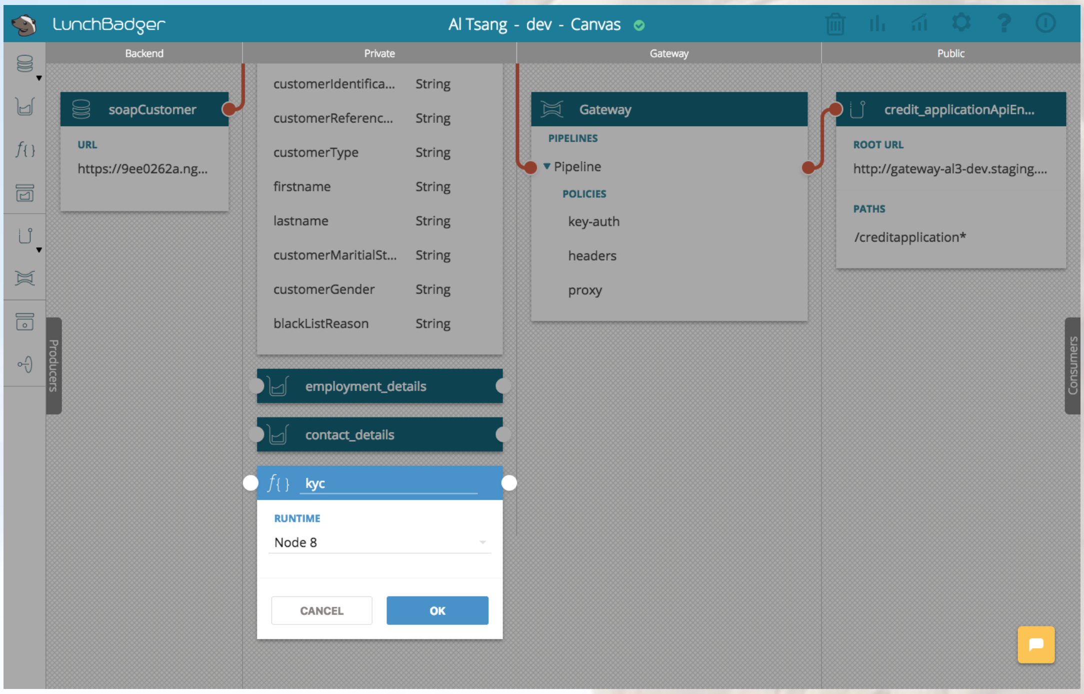
To automate the manual Know Your Customer process using Express Serverless Platform, each time a new credit application is received through the newly developed API, a triggered event could initiate an automated search through a serverless function.

A serverless function called **kyc** is easily created by clicking on the serverless function entity on the entity palette.

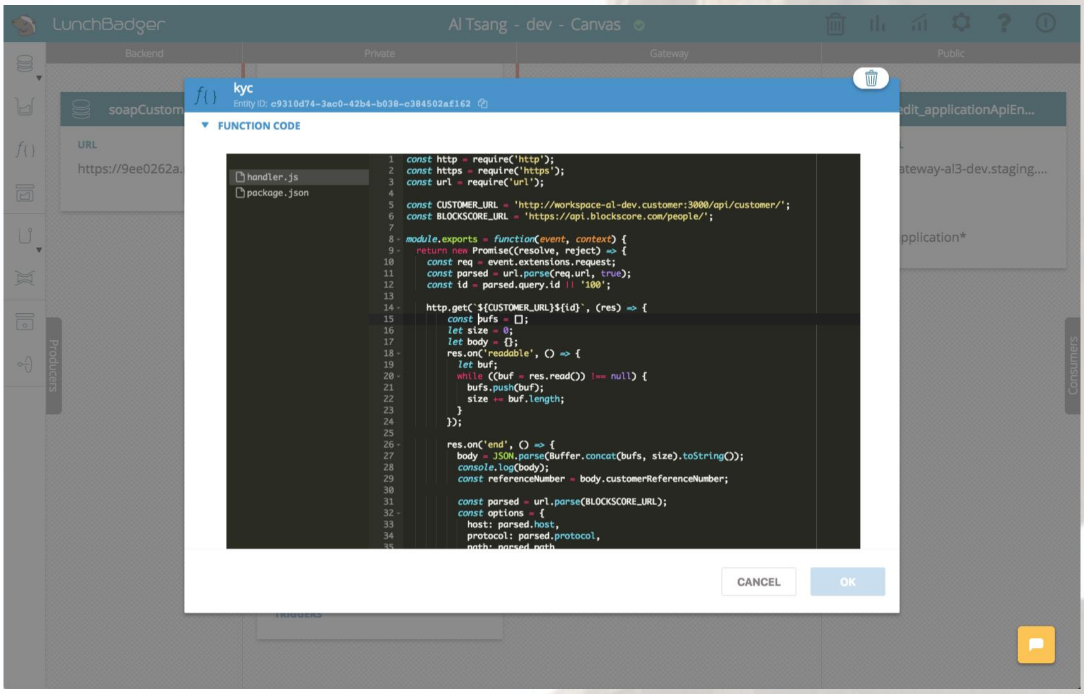
The **kyc** function below will be written in Node.js.  The function body will execute a search based on the customer information entered into the **credit_application** model to check the person's credit score through a third party provider and can automate background checks as a service.

The **kyc** function will only be run when a **credit_application** submission is successfully submitted.

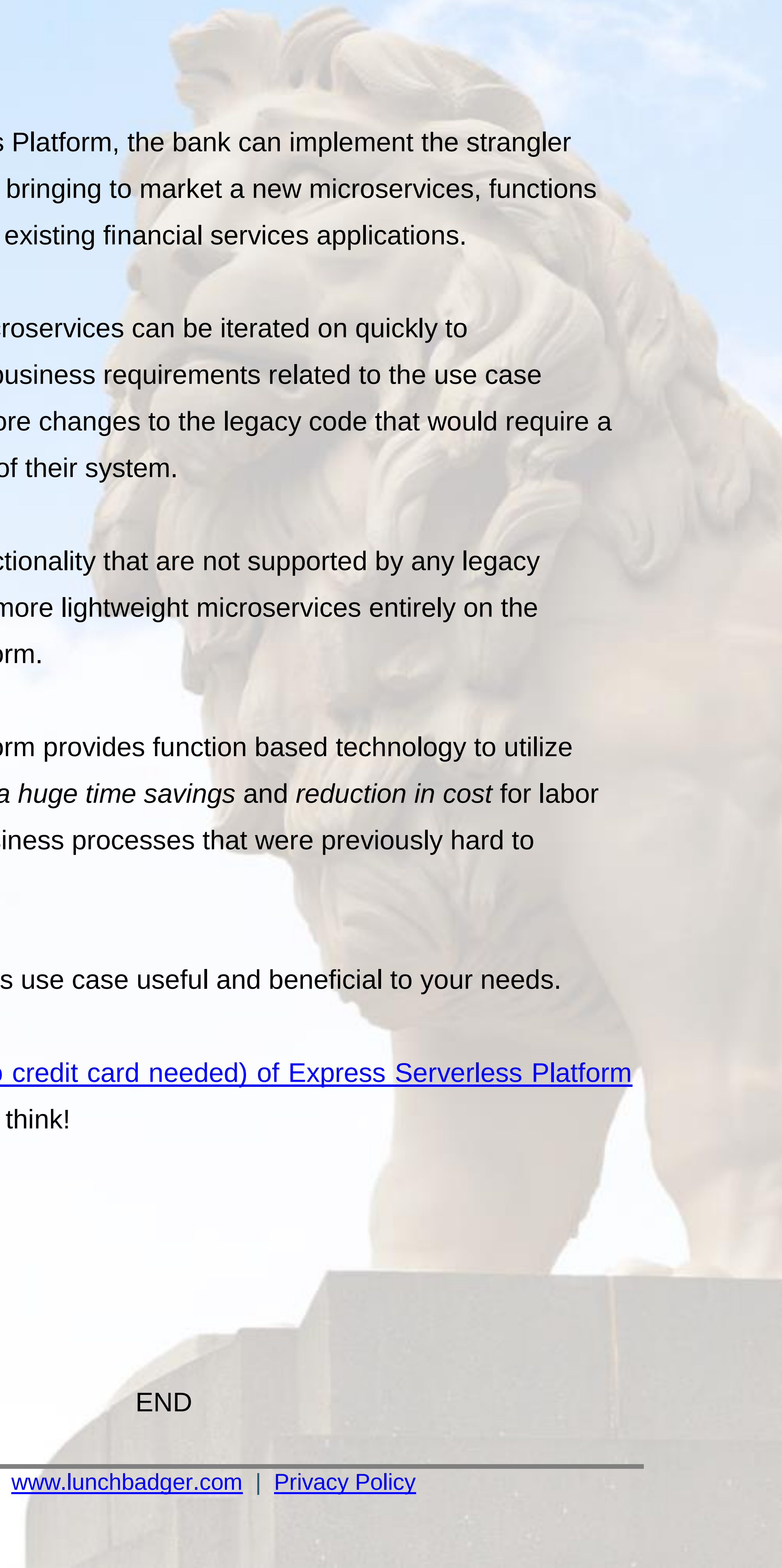
The bank has complete control on whether they would like to batch such function calls to be done in a number of applications to be processed during off hours or in real time for quicker responses depending on the client calling its credit_application API.



| | |
|---|---|
| customerIdentifica... | String |
| customerReferenc... | String |
| customerType | String |
| firstname | String |
| lastname | String |
| customerMaritialSt... | String |
| customerGender | String |
| blackListReason | String |



```
const http = require('http');
const https = require('https');
const url = require('url');

const CUSTOMER_URL = 'http://workspace-al-dev.customer:3000/api/customer/';
const BLOCKSCORE_URL = 'https://api.blockscore.com/people/';

module.exports = function(event, context) {
  return new Promise((resolve, reject) => {
    const req = event.extensions.request;
    const parsed = url.parse(req.url, true);
    const id = parsed.query.id || '100';

    http.get(`${CUSTOMER_URL}${id}`, (res) => {
        const bufs = [];
        let size = 0;
        let body = {};
        res.on('readable', () => {
          let buf;
          while ((buf = res.read()) !== null) {
            bufs.push(buf);
            size += buf.length;
          }
        });

        res.on('end', () => {
          body = JSON.parse(Buffer.concat(bufs, size).toString());
          console.log(body);
          const referenceNumber = body.customerReferenceNumber;

          const parsed = url.parse(BLOCKSCORE_URL);
          const options = {
            host: parsed.host,
            protocol: parsed.protocol,
            path: parsed.path
```



## Summary

Using Express Serverless Platform, the bank can implement the strangler pattern easily and *quickly* bringing to market a new microservices, functions and APIs leveraging their existing financial services applications.

The credit application microservices can be iterated on quickly to accommodate changing business requirements related to the use case without having to make core changes to the legacy code that would require a cumbersome revisioning of their system.

Furthermore, gaps in functionality that are not supported by any legacy system can be written in more lightweight microservices entirely on the Express Serverless Platform.

Express Serverless Platform provides function based technology to utilize other cloud services and *a huge time savings* and *reduction in cost* for labor by automating legacy business processes that were previously hard to displace.

We hope you've found this use case useful and beneficial to your needs.

Sign up for Free Trial (no credit card needed) of Express Serverless Platform and let us know what you think!

END