Business Process for SMEs utilizing E-invoicing APIs

Student teams will develop an e-invoicing application for a vendor engaged in a supply-chain relationship. ***Teams will be provided with a list of business names, and you have the choice to select any one business entity. You also have the choice to find out and work on another business not listed in the list. But firstly, you must discuss and confirm it with me.*** Each business entity uses a business API to accomplish business internal/external tasks. Your job is to find out the business APIs and play with them. The APIs shall conform to any of the three categories: E-invoice creation, E-invoice validation and E-invoice communication. For reference, some examples of APIs for each category are in the Section: Introduction to E-invoicing below.

To proceed, teams should consider the following instructions:

1. Our project will be based on a single REST API selected for each team or an external API of their own.
2. Write clients or use similar techniques to access the API.
3. Teams should identify the API endpoints, and all the information related to the API like the KHEA Version 1 table (will be provided).
4. Create a testing GUI or a web interface to test the endpoints separately.
5. Some teams will be selected to create a business process using another API and include that in GUI.
6. However, some groups will design a recommendation system for selection of best endpoint/API between 2 APIs

Points 5 and 6 will be decided later son don't need to think and worry about it now. I will pass the details later.

Within the project, there are some of the tasks that teams need to perform which includes:

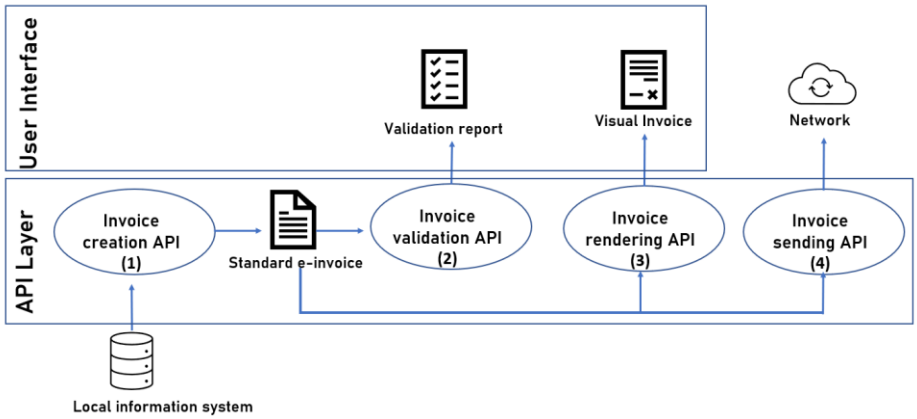
- investigate the selected business API
- find out their OpenAPI specification or API documentation
- research on how to use that API and write client code
- search out all the possible endpoints
- find out the data format (XML/JSON etc.)
- investigate the authentication methods etc.

  A sample table (KHEA Version 1) will be provided to each group to tally and list the details accordingly.

Regarding the web application, the testing GUI should be designed to ensure the business API is working properly. All the endpoints of the API should be tested and demonstrated on GUI separately such that the user can easily go to the specific endpoint to perform a task.

For a business process: Students can use the APIs assigned by the client to develop a business process or you can use other external business APIs. All required prototype scenarios will be provided by the client for each given category. Groups should utilize the business APIs to carryout prototype scenarios.

The application is illustrated below:



To design a business process, there are some variations depending on which API is used. Possible variations will be communicated with the groups.

Sample Variation

- **Invoice creation:** https://upbrainsai.com/ will allow you to convert a PDF invoice to JSON. This JSON will need to be converted to UBL XML
- **Invoice validation:** ESS Validator (for details contact Muhammad Raheel Raza muhammad_raheel.raza@unsw.edu.au). A guide to ESS Validator is in the given PDF ESS Validate API - Developer Guide 2- Google Docs.pdf
- **Invoice sending**: Service to send invoices through email

Below is a brief background study of E-invoicing and the mentioned categories.

Introduction to e-invoicing

Background

Industry 4.0 and e-invoicing

Contemporary manufacturing often involves large manufacturers dealing with a small number of risk-sharing partners who co-design and deliver key subsystems of the customized product. These partners sub-contract the supply of key systems to their suppliers which results in a vast network of collaboration activities throughout the supply chains involving companies of all sizes. The "digitization" of production and logistics provides a number of significant, new advantages to modern supply chain management in the so-called *Industry 4.0* concept. Industry 4.0 concept embraces the Internet of Things, Internet of Services and Cyber-Physical Systems to bring the new design for modern manufacturing. Design features of Industry 4.0 are Interoperability, Virtualisation, Decentralisation, Real-Time Capability, Service Orientation, and Modularity.

The rise of Industry 4.0 is driven by the need to support the rapid formation of partner collaborations, the ability to respond to fast-changing market needs and the capacity to quickly respond to new business opportunities. SMEs in collaborative networks focus on their core competencies and try to find complementary partners to compensate for the lack of competencies and cooperate to fit to tender or to run new product development

The main motivation behind this project is to support small and medium-sized enterprises so that they can fully participate in Industry 4.0 emergent collaborations. Its main focus is in the area of business document exchanges between collaborating partners and in particular e-Invoicing. E-invoicing usage forms part of a company's utilisation of electronic data interchange (or EDI), which also encompasses the automation of other key supply chain partner interactions. The most common forms of e-invoicing are divided into two categories: B2B (Business to Business) and B2G (business to government). There is another category B2C (Business to Consumers) but it is not widely used at present.

Regulations play an important role in the adoption of e-invoicing within one country or region. Such regulations mandate the use of certain standards to guarantee the security and integrity of business document exchanges and fight against tax evasion and fraud. For example, the EU Commission implemented in April 2014 new regulations relating to public procurement across Europe in the form of the e-invoicing Directive 2014/55/EU, which set out deadlines by which European government bodies must be able to receive structured electronic invoices from suppliers.

The introduction of e-invoicing legislation across Europe in particular has greatly boosted the number of companies wishing to be able to exchange structured electronic invoices. Currently, the number of invoices sent across Europe annually is estimated to exceed 40 billion. Meanwhile, the annual growth rate for e-invoicing is around 10-20%.

e-Invoicing Standards

In 2014, the European Commission declared UBL 2.1 was officially eligible for referencing in tenders from public administrations (one of the first non-European standards to be so recognized). It is expected that the adoption of UBL as a **standard message representation** will increase as it defines a royalty-free library of standard XML business documents that not only supports invoicing but also all other aspects related to the digitization of the commercial and logistical processes for domestic and international supply chains (e.g. procurement, purchasing, transport, logistics, intermodal freight management).

In addition to data and messaging standards, there are multiple alternatives for **communication methods** like email or secure File Transfer Protocol (SFTP). Peppol is a set of artifacts and technical specifications that facilitates easy data exchange across disparate government systems and their suppliers. Many EU countries have adopted Peppol as a communication method including Belgium, Croatia, Cyprus, Denmark, Greece, Ireland, Latvia, Lithuania, Luxemburg, Malta, Norway, Poland, and SloveniPEPPOL's most recent usage statistics can be accessed at PEPPOL statistics (ionite.net).

In Australia, all New South Wales state government agencies will have to use e-invoicing for goods and services valued at up to AUD 1 million from 2022. It is expected that this will be extended to SMEs in the longer term so a solution that is flexible to use is needed. Existing solutions present difficulties when adopted by SMEs:

- Cost
- Flexibility
- Vendor dependence

Therefore, SMEs need a **cost-effective solution** that is both **open** and **flexible**. This way, they can better leverage their existing assets, use components from different vendors and be in control of the evolution of their e-invoicing solutions.

Towards a SaaS e-invoicing ecosystem for Australia

What is Software-as-a-Service (SaaS)?

Software as a Service (SaaS) is a concept that pushes software away from local computers into Web services. This means users no longer buy software to install locally but instead buy the right to use the software over the Internet. SaaS providers maintain the hardware, perform upgrades, backup your data (sometimes), and otherwise perform all of the "keep the lights on" services and activities required to keep the software running. [1]

The purpose of the project is to introduce students to the concept of SaaS by considering each team as a company that performs the following:

1. The company creates a software product (referred to as a *service* as it is accessible via a REST API) and hosts it on a cloud server.
2. Customers can subscribe to the service as soon as it is operational, giving feedback to the company

3. The company makes both major and minor updates to the software, and the customers automatically get those updates as part of their subscriptions.

We will only be concerned with the technical aspects of SaaS in this project (the building and deployment of the software product) rather than other aspects like charging etc.

All software services that will be built by the teams will be around creating a SaaS e-invoicing ecosystem for Australia.

[1] Software as a Service Vs. Software as a Product - Pragmatic Institute

Ecosystem description

An **ecosystem of services** consists of SaaS services that can be composed, configured and customised according to different contexts in accordance with Service-Oriented Architecture (SOA) and Business Process Modelling (BPM) principles. Each service is providing a particular functionality in the-invoicing services ecosystem, can potentially be provided by a different vendor, is available via an API and is available on a pay-by-use basis. Companies (and in particular SMEs) can create flexible, open and multi-vendor solutions in the form of **e-invoicing workflows** that enforce business document exchange processes that are applicable in a certain context.

The categories of services in the ecosystem are listed in the following table:

SaaS Category	**Description**	**Customization Points**
Invoice Creation	Creates a standardised einvoice from an existing information system	Supports different input formats: • Convert from a text file (CSV or Excel) • Read from SQL Tables • EXtract data from PDF Invoice (image) • Ask user to fill a GUI Form
Invoice Validation	Creates a report outlining any validation errors according to Australian rules	Supports the generation of validation reports in different formats: • JSON • HTML • PDF

Invoice Sending/Receiving	Sends the electronic invoice to a number of recipients. Can fail if there is a communication error. Receives an electronic invoice from a mailbox.	Supports the sending of electronic invoices using different protocols: • Email • SFTP Supports the generation of communication reports in different formats: • JSON • HTML • PDF

Standardised electronic invoices

A rising international standard for representing e-invoices is the **UBL XML standard** which has been adopted in Australia. UBL was originally introduced in 2003, as an open, extensible, and customizable XML document for business. In September 2008, European stakeholders in public and government created the Pan-European Public Procurement On-line (PEPPOL) project. With UBL formatted XML invoices, it is possible to record business transactions from supplier to transportation, and finally to the buyer. In this context, we assume all standardised electronic invoices are in XML format. An example is shown in the diagram below:

```
<cbc:InvoiceTypeCode>380</cbc:InvoiceTypeCode> ❶
<!-- Code omitted for clarity -->
<cac:AllowanceCharge>
    <cbc:ChargeIndicator>true</cbc:ChargeIndicator>
    <cbc:AllowanceChargeReason>Insurance</cbc:AllowanceChargeReason>
    <cbc:Amount currencyID="EUR">-25</cbc:Amount>❷
    <cac:TaxCategory>
        <cbc:ID>S</cbc:ID>
        <cbc:Percent>25.0</cbc:Percent>
        <cac:TaxScheme>
            <cbc:ID>VAT</cbc:ID>
        </cac:TaxScheme>
    </cac:TaxCategory>
</cac:AllowanceCharge>
<!-- Code omitted for clarity -->
<cac:LegalMonetaryTotal> ❸
    <cbc:LineExtensionAmount currencyID="EUR">-1300</cbc:LineExtensionAmount>
    <cbc:TaxExclusiveAmount currencyID="EUR">-1325</cbc:TaxExclusiveAmount>
    <cbc:TaxInclusiveAmount currencyID="EUR">-1656.25</cbc:TaxInclusiveAmount>
    <cbc:ChargeTotalAmount currencyID="EUR">-25</cbc:ChargeTotalAmount>
    <cbc:PayableAmount currencyID="EUR">-1656.25</cbc:PayableAmount>
</cac:LegalMonetaryTotal>

<cac:InvoiceLine>
    <cbc:ID>1</cbc:ID> ❹
    <cbc:InvoicedQuantity unitCode="DAY">-7</cbc:InvoicedQuantity>
    <cbc:LineExtensionAmount currencyID="EUR">-2800</cbc:LineExtensionAmount>
    <!-- Code omitted for clarity -->
    <cac:Price>
        <cbc:PriceAmount currencyID="EUR">400</cbc:PriceAmount>❺
    </cac:Price>

<cac:InvoiceLine>
    <cbc:ID>2</cbc:ID>❻
    <cbc:InvoicedQuantity unitCode="DAY">3</cbc:InvoicedQuantity>
    <cbc:LineExtensionAmount currencyID="EUR">1500</cbc:LineExtensionAmount>
    <!-- Code omitted for clarity -->
    <cac:Price>
        <cbc:PriceAmount currencyID="EUR">500</cbc:PriceAmount>
    </cac:Price>
```

There are slight variations between the different UBL invoices according to countries. In this context, we use Australia's UBL specifications available at:

- https://softwaredevelopers.ato.gov.au/sites/default/files/resource-attachments/A_NZ_Invoice_specification_tracked_v1.0.docx
- https://github.com/A-NZ-PEPPOL/A-NZ-PEPPOL-BIS-3.0

We are only considering Commercial Invoice (type 380) so other types can be ignored.

Introducing UBL XML format

What is UBL?

Defined by OASIS Open, UBL, the Universal Business Language, defines a royalty-free library of standard XML business documents supporting the digitization of the commercial and logistical processes for domestic and international supply chains such as procurement, purchasing, transport,

logistics, intermodal freight management, and other supply chain management functions. UBL has also become foundational to a number of efforts in the transport and logistics domain, including the European Common Framework (European Commission), DTTN (Port of Hong Kong), TradeNet (Port of Singapore), Electronic Freight Management (US), and Freightgate (globally). In 2014 the European Commission declared UBL 2.1 was officially eligible for referencing in tenders from public administrations (one of the first non-European standards to be so recognized).

In this document, we are only interested in describing the UBL XML standard and in particular the representation of electronic invoices in the Australian context. Such electronic invoices have these characteristics:

- Each core element of an e-invoice is part of the EN16931 semantic data model, established as a European Standard. EN16931 includes only the essential information that an e-invoice needs to ensure legal compliance.
- The syntax of the UBL document must be compliant with the UBL schema
- Each country can "customise" the standard according to its own requirements.

More information about UBL XML can be found here https://www.xml.com/articles/2017/01/01/what-is-ubl/

Example of a UBL Invoice

The figure below is a subsection of the UBL invoice where each number on the image represents:

1. Invoice type code 380 indicates it is an invoice.
2. The charge amount is negative to correct the original invoice.
3. All document-level (LegalMonetaryTotal) amounts are negative.
4. Start of invoice line 1.
5. The price amount must always be positive and is not changed.
6. Start of invoice line 2.

```
<cbc:InvoiceTypeCode>380</cbc:InvoiceTypeCode> (1)
<!-- Code omitted for clarity -->
<cac:AllowanceCharge>
    <cbc:ChargeIndicator>true</cbc:ChargeIndicator>
    <cbc:AllowanceChargeReason>Insurance</cbc:AllowanceChargeReason>
    <cbc:Amount currencyID="EUR">-25</cbc:Amount> (2)
    <cac:TaxCategory>
        <cbc:ID>S</cbc:ID>
        <cbc:Percent>25.0</cbc:Percent>
        <cac:TaxScheme>
            <cbc:ID>VAT</cbc:ID>
        </cac:TaxScheme>
    </cac:TaxCategory>
</cac:AllowanceCharge>
<!-- Code omitted for clarity -->
<cac:LegalMonetaryTotal> (3)
    <cbc:LineExtensionAmount currencyID="EUR">-1300</cbc:LineExtensionAmount>
    <cbc:TaxExclusiveAmount currencyID="EUR">-1325</cbc:TaxExclusiveAmount>
    <cbc:TaxInclusiveAmount currencyID="EUR">-1656.25</cbc:TaxInclusiveAmount>
    <cbc:ChargeTotalAmount currencyID="EUR">-25</cbc:ChargeTotalAmount>
    <cbc:PayableAmount currencyID="EUR">-1656.25</cbc:PayableAmount>
</cac:LegalMonetaryTotal>

<cac:InvoiceLine>
    <cbc:ID>1</cbc:ID> (4)
    <cbc:InvoicedQuantity unitCode="DAY">-7</cbc:InvoicedQuantity>
    <cbc:LineExtensionAmount currencyID="EUR">-2800</cbc:LineExtensionAmount>
    <!-- Code omitted for clarity -->
    <cac:Price>
        <cbc:PriceAmount currencyID="EUR">400</cbc:PriceAmount> (5)
    </cac:Price>

<cac:InvoiceLine>
    <cbc:ID>2</cbc:ID> (6)
    <cbc:InvoicedQuantity unitCode="DAY">3</cbc:InvoicedQuantity>
    <cbc:LineExtensionAmount currencyID="EUR">1500</cbc:LineExtensionAmount>
    <!-- Code omitted for clarity -->
    <cac:Price>
        <cbc:PriceAmount currencyID="EUR">500</cbc:PriceAmount>
    </cac:Price>
```

Australia/NZ UBL Specifications

This course will use the specific requirements for Australia and New Zealand. Material related to the specs can be found in https://github.com/A-NZ-PEPPOL/A-NZ-PEPPOL-BIS-3.0.

For example, samples can be accessed from this folder https://github.com/A-NZ-PEPPOL/A-NZ-PEPPOL-BIS-3.0/tree/master/Message%20examples

Australian E-invoice Examples

Two examples are provided:

Example 1: Simple/minimal UBL XML invoice example1.xml

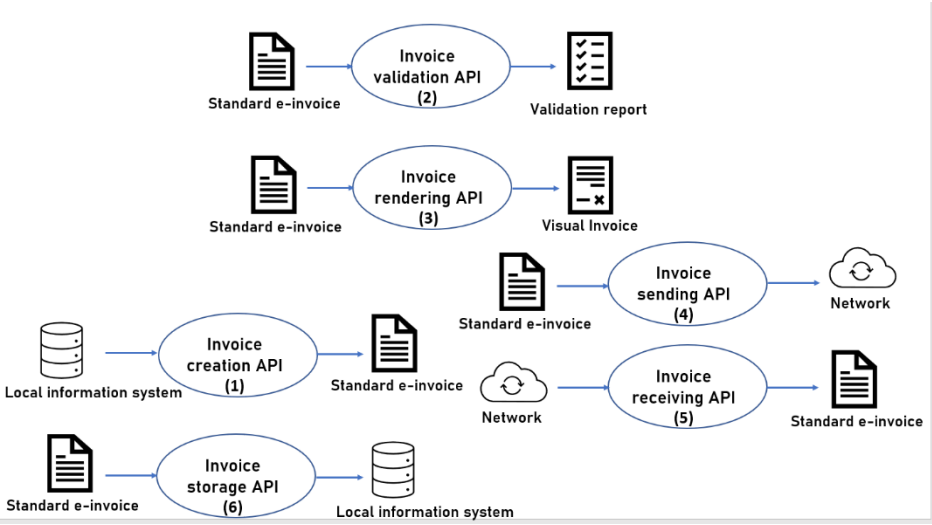
Example 2: More complete example, available from the ATO GitHub repository AU Invoice.xml

The following spreadsheet gives the correspondence between the tags and the values in both examples:

https://docs.google.com/spreadsheets/d/12tn0g5ri40iLO2vLLLKjviSDPAs81swIiEASVbN4MtE/edit?usp=sharing

Ecosystem summary

The figure below summarises all API categories in the ecosystem, however only three categories are covered in this project i.e. E-invoice creation, E-invoice validation and E-invoice sending. A brief description of API inputs, outputs, functionality are described below:



Invoice Creation

Only standardised data can be processed by other categories of services. To reduce the barrier for customers to use the system, the customer should be able to convert data from their current system to standard e-invoices in batches.

For this category, your service should be able to create standardised e-invoices from an existing information system or allow information to be imported from external sources. Although you are not expected to be able to extract information from a random source like websites or articles, services in this category should be able to process pre-formatted data. The service should extract any useful data from the source and assemble invoices as per the standard.

Input

Below are some possible approaches for creating an invoice, the implementation of your service can support some of them, or even something not on the list.

- Convert from a text file: you can assume the data of the invoice is contained in a text file (CSV or Excel)
- Read from a database: you can assume the invoice data is in SQL Tables
- Extract data from an image: you can assume you are reading invoice data from a PDF
- Ask user to fill a GUI Form

You should define the parameters required by your service. You may define optional and compulsory fields, but more compulsory fields generally mean you will handle fewer exceptions in the service but reduce the usability and generality of the service.

Output

Your output should conform to the UBL 2.1 XML format. i.e. Your output should be able to pass any correctly implemented validator mentioned in the "Invoice Validation" section of this document. Since there are many fields and constraints in the standard, a microservice should consider starting by generating minimised invoices first.

Possible APIs that can be used to implement the service

- https://upbrainsai.com/ API if to extract data from an invoice image or PDF.
- There are several APIs such as **Invoice Data Extraction** in RAPID API that can extract data from an invoice using OCR techniques.

Invoice Validation

Creates a report outlining any validation errors according to Australian rules. ATO maintains a GitHub repository with all the specifications at https://github.com/A-NZ-PEPPOL/A-NZ-PEPPOL-BIS-3.0/tree/master/Specifications

Latest Details of Validation rules can be found in Appendix A and B of the document "Specifications/A-NZ_Invoice_Extension_versionX.docx" in the ATO GitHub repository

In order to maintain an efficient system, it is essential that the invoices are validated based on the rules employed by a governing entity. The validation system aids to detect and rectify errors before a payment is initiated.

The validation microservice has four components:

- Validating Wellformedness (Checking if the input is valid and the file itself has valid contents)
- Syntax Rules (Validates EN16931 business rules) (Appendix B, BR Rules)
- PEPPOL rules (Validates Specific AUNZ business rules) (Appendix B, PEPPOL Rules)
- Schema Validation (Validates the schema of the invoice

Note: Schema rules are rules concerning what tags are allowed in the UBL XML file. For example tag cac:BillingReference is a valid tag and tag <cac:sydney> is invalid. It also specifies how the

tags are sequenced. Schema rules are contained in an XSD file. In this case, the UBL XSD file is downloadable at: https://docs.oasis-open.org/ubl/os-UBL-2.1/xsd/maindoc/UBL-Invoice-2.1.xsd (you may need to copy the path as it may not connect directly) To validate an XML file, you need to find a library that does schema validation in your programming language.

Input

The input to this microservice will be a *UBL 2.1 XML format* invoice.

Output

The output of this microservice should:

1. Be in a machine-readable format (i.e. JSON, HTML, PDF)
2. Indicate which validation rules are violated.
3. Provide additional information on why the rule was violated.
4. *Optional* Suggest quick-fix methods to validate the rules.

Possible APIs that can be used to implement the service

- ESS Validator (for details contact Muhammad Raheel Raza muhammad_raheel.raza@unsw.edu.au). A guide to ESS Validator is in the given PDF ESS Validate API - Developer Guide 2- Google Docs.pdf
- ECOSIO validator: https://ecosio.com/en/peppol-and-xml-document-validator/

Invoice Sending/Receiving

After invoices are generated, the system needs a way to distribute the invoices to corresponding recipients. This category of services should be able to deliver invoices from the internal system to some external environment. You need to design the way how invoices are delivered. Some possible ways can be email, SFTP or even SMS.

Input

Input will conform to the UBL format in the recommended format. (i.e., The input should be able to pass the validator mentioned in the "Invoice Validation" section). But you should consider and define how your service takes the input.

Output

The observable output of this service includes a report and a side effect. After the API call, your service should send the invoice and then return a communication report. The format of the report can be but is not limited to JSON, HTML, and PDF. If there is any communication error. it is preferable that your service reacts with a human-readable message. A simple synchronous API

call without delivering information to external is acceptable as a start, but not sufficient for the service. Similar is the case for Receiving scenario.

Possible APIs that can be used to implement the service

Storecove API can be used https://www.storecove.com/au/en/ (if PEPPOL network is selected).