Dmitrii Kislenko

CONTINUING DEVELOPMENT OF NETTILASKU.FI FRONTEND BASED ON THE UPDATED DESIGN

Continuing Development of nettilasku.fi Front-end Based on the Updated Design

Dmitrii Kislenko

Bachelor’s Thesis
Spring 2021
Information Technology

Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Web Development

Author(s): Dmitrii Kislenko
Title of the bachelor’s thesis: Continuing development of nettilasku.fi front-end based on the updated design
Supervisor(s): Lasse Haverinen
Term and year of completion: Spring 2021 Number of pages: 46

Epic Invoicing needed for the development of the new nettilasku project by adding new features and integrating features from the older nettilasku project. The thesis aimed at discovering the principles of the older application features realization together with the issues in the current project and finding methods for implementing enhanced solutions applying required tools in order to improve user experience in the building product. The work was commissioned by Epic Invoicing.

The work was going repeatedly in the following way: the management gives a task of feature development to the author, the author investigates the issue and finds methods of its implementation, the author builds the solution.

All the features that were implemented were accepted by the management. Three of them were put into production and two of them were put to the open test stage.

Keywords: React, TypeScript, Material UI, Web-Development

PREFACE

The work was done remotely in Oulu for Epic Invoicing during the coronapandemic in March-May 2021. It would be quite difficult to finish it in time without the help of the senior full-stack developer Ville Juppi who was giving good technical advice about the architecture of the project and programming principles, company representative Jarkko Kähkönen who was supervising the project by giving tasks for the development team and checking the quality of provided solutions and tutoring teacher Veijo Väisänen who was carefully leading me through the process of this thesis writing. I want to thank them all for their support.

Oulu, May 15, 2021
Dmitrii Kislenko

TABLE OF CONTENTS

ABSTRACT 3
PREFACE 4
TABLE OF CONTENTS 5
VOCABULARY 7
1 INTRODUCTION 8
2 DESCRIPTION OF THE CURRENT STATE 9
2.1 Employer company 9
2.2 Competence requirements and the author task 9
2.3 Project concepts 9
3 PURPOSE AND OBJECTIVES 11
4 DESCRIPTION OF WORK TASKS AND LEARNING AS DIARY ENTRIES 12
4.1 Creation of the Dashboard page 12
4.1.1 Existing solutions investigation 12
4.1.2 Building a solution 14
4.1.3 Conclusion 16
4.2 “Accounting accounts” page integration from the older nettilasku application 17
4.2.1 Implementation 17
4.2.2 Conclusion 22
4.3 “Intercom” pop-up integration 22
4.3.1 Investigation of the existing libraries 23
4.3.2 Implementation 24
4.3.3 Conclusion 26
4.4 “Purchase invoice” section implementation 26
4.4.1 Creating section architecture 27
4.4.2 Creating table and form components 30
4.4.3 Conclusion 34
4.5 Invoice folder feature integration 35
4.5.1 How “invoice folder” feature works in the older application version 35

4.5.2 Nettilasku architecture 36
4.5.3 Implementation 37
4.5.4 Conclusion 42
5 CONCLUSION 43
REFERENCES 44

VOCABULARY

CEO	Chief executive officer
DOM	Document Object Model, a way of representing a structured document using objects in React JS
MUI	Material User Interface
JS	JavaScript, a programming language that is used in React framework
JSON	JavaScript Object Notation, it is a format for storing and exchanging human-readable information. The file contains only text and uses the .json extension.
JWT	JSON Web Token, one of the safest ways to transfer information between two participants based on JSON object. For its creation, it is needed to define a header with general information on the token, payload, and signature.
React JS	JavaScript library used for creating Web Application

1 INTRODUCTION

Nettilasku is a project of Epic Invoicing that aims at providing online bookkeeping software. The most recent solution that was published is Nettilasku.fi web application. The platform offers a wide variety of functionalities used for accounting, invoicing, and financial management. The application structure consists of different types of forms and list-views that display and allow to edit the information about the financial operations and invoices of the user.

The current version of the Nettilasku platform was published in 2012 and many of the design and technical solutions are already out of date. Based on that fact in July 2020 the company started building a new web application due to the latest programming trends to improve user experience and provide better service. In February 2021 the first production version of the application was published.

The current state of the project does not contain all the functionality that is applied to the older version. The thesis targets to integrate into the application as many of the existing features of the older Nettilasku platform as possible, debug existing, and add new functionality based on the updated design patterns and required tools.

2 DESCRIPTION OF THE CURRENT STATE

2.1 Employer company

As it was mentioned earlier the Nettilasku project was started by Epic Invoicing OY which is a Finnish software company located in Tampere. The income of the company for 2019 amounted to 287 thousand euros according to finder.fi [1]. The number of employees that are constantly involved in the Nettilasku project is 3: one full-stack developer, one front-end developer, and one back-end developer. The main stakeholder is Jarkko Kähkonen who is the CEO of Epic Invoicing. The project aims at updating the user interface of the old web application in favor of obtaining great benefits with its help, such as attracting new customers, expanding the functionality of the old application, in line with new trends in web development.

2.2 Competence requirements and the task of the author

Competence requirements in the project consisted of applying skills in web front-end development using a required set of tools, the ability to work in a small front-end team while developing the back-end in parallel, and the ability to quickly assimilate new information.

The general task of the author in this thesis was to integrate existing functional front-end solutions from the old nettilasku.fi web application into a new one. Before starting the thesis, the author had worked for 9 months on the development of this application which made him a competent junior web developer. During that period concepts that were required for the project were deeply studied. The professional qualifications of the author can be regarded as an experienced front-end developer who matches the development needs.

2.3 Project concepts

The chosen project relies on the following concepts:

- Building the application using React JS [2] in cooperation with TypeScript [3]. React is a JavaScript library for building web interfaces. It is the core of the whole project which combines all the required tools. The main technical advantage of this tool is the possibility of DOM manipulation via state variables. This concept enables interaction with a web application without reloading the page every time something is changed on the page which provides a better user experience. Additionally, React JS has broad developers' community that helps in resolving most of the development issues and the Node Package Manager tool that provides ready-made solutions for web applications. TypeScript is a tool that adds static type definitions and code validation to the application. It improves the readability and maintainability of the program and simplifies documentation writing.
- Using Material UI as a pattern for creating updated views. Material Design is a design system created by Google to help in building high-quality digital for the web [4]. This concept was used for styling all the application views. The main advantage over other similar tools is the possibility of usage Material-UI NPM package [5] for building interfaces in React application. It includes many ready-made styled components, views, and solutions that simplify the process of designing new pages and front-end development in general.
- Using "Formik" [6] for working with forms. "Formik" is an NPM package for HTML forms handling. It contains a set of pre-defined functions and methods that helps in validating, error handling, and submitting forms.
- Using "react-query" [7] for interacting with the back-end. This is an NPM package that provides a set of hooks for sending and requesting information from the server. It also has features for server-side table pagination, query cache, query invalidation, etc.

3 PURPOSE AND OBJECTIVES

The purpose of the thesis was to continue the development of nettilasku.fi front-end based on the updated Material Design pattern and to provide as many working features as possible for the final version of the product. That implies integrating features from the older nettilasku.fi web application into the project by and creating new features that were requested by the stakeholder using concepts described in the second chapter.

The thesis aims at discovering the principles of the older application features realization together with the issues of the current project and finding methods for implementing enhanced solutions applying required tools in order to improve user experience in the building product.

The learning objectives of the thesis were to deeply study concepts of building and improving React applications using Typescript and Material UI.

The diary reporting plan was written on a topic basis and contains a description of the most important features that were implemented in 8 weeks of working on the project. Implementing these took 215 hours in total. The rest of the time took debugging, testing, and small features creating which are not included here. There was no initial plan of development for this period of time because new tasks were coming gradually. This thesis covers the following feature implementation:

- “Dashboard” page creation
- “Intercom” pop-up integration from the older nettilasku application
- “Invoice folder” feature integration from the older nettilasku application
- “Accounting accounts” page integration from the older nettilasku application
- “Purchase” section integration from the older nettilasku application

4 DESCRIPTION OF WORK TASKS AND LEARNING AS DIARY ENTRIES

4.1 Creation of the Dashboard page

The stakeholders considered it necessary to create a dashboard that can display the following data:

- Statistical numbers: turnover from the beginning of the accounting period, turnover for the current month, the sum of overdue sales invoices, and the sum of overdue purchase invoices.
- Cash flow line chart that contains the income of the user, outcome, and total values based on the timeline.
- Overdue purchase invoices and Bank accounts tables.
- Messages box list view/table (between employee/bookkeeper/customer/customer support, etc.)

The task was to investigate existing MUI dashboard solutions and implement a dashboard page in the nettilasku updated version that displays the required information.

4.1.1 Existing solutions investigation

Due to the given requirements, the desired solution is supposed to have an interface with a line chart, a way of representing a one-number type of data (for turnover and amount of overdue sales invoices values), and tables for showing the list type of data (messages and overdue purchase invoices). Additionally, accessible free source code would be a great advantage because it would simplify the development process. The main aim of the investigation is to find design or code solutions that could be applied to the nettilasku dashboard.

4.1.1.1 Material Kit Pro v4

Material Kit Pro v4 by Devias [8] is a good example that matches the request of the stakeholders. It is a professional React dashboard that comes with ready-to-

use Material-UI components. The template has a graph and numbers that are represented with cards and related to them pie-charts. The example also includes a message section and the latest transactions card where statistical changes are highlighted with green or red colors depending on the progress. The dashboard is presented in figure 1.

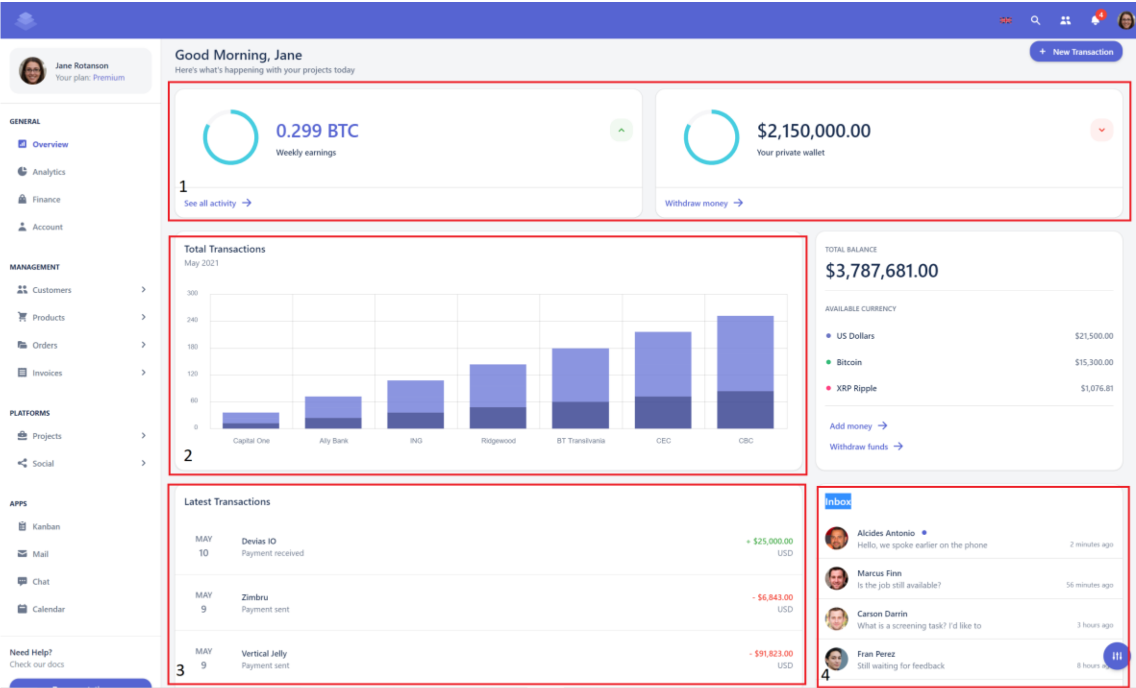

Figure 1. Material Kit Pro v4 dashboard: One-number cards (1), Barchart (2), Table with statistical data (3), Messages section (4)

The main structure of this solution looks modern and user-friendly which correlates to the nettilasku dashboard needs. The visual investigation revealed that the dashboard structure, table, one-number data, and messages can be used as a design example of data representation. The disadvantage of this interface is the lack of freely available source code.

4.1.1.2 Material UI native dashboard template

Native MUI template is a free dashboard template provided by MUI developers [9]. The example is done using native material UI components and “recharts” npm package [10]. One-number data is presented as a card with a number and

additional information on it. The dashboard also includes a table and a line chart. The general structure is quite similar to the Material Kit Pro v4 template. The example is presented in figure 2.

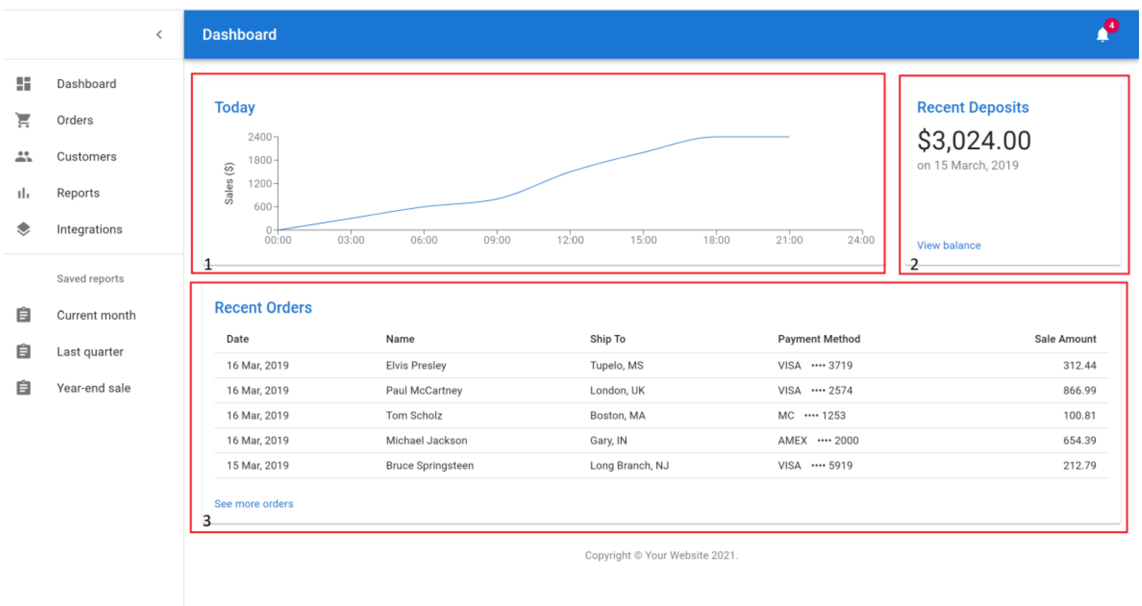

Figure 2. MUI native dashboard example: Material Kit Pro v4 dashboard: Line chart (1), One-number card (2), Table with statistical data (3)

This dashboard looks more ordinary in terms of design decisions but it has free available source code. Additionally, all components and the graph are made with libraries that were used in the nettilasku project before(“material-ui” and “recharts”). Investigation of the code revealed that this solution could provide the line chart and general material UI components.

4.1.2 Building a solution

After analyzing and comparing the investigated samples the final solution was built by combining parts of the researched dashboards.

The solution is shown in figure 3.

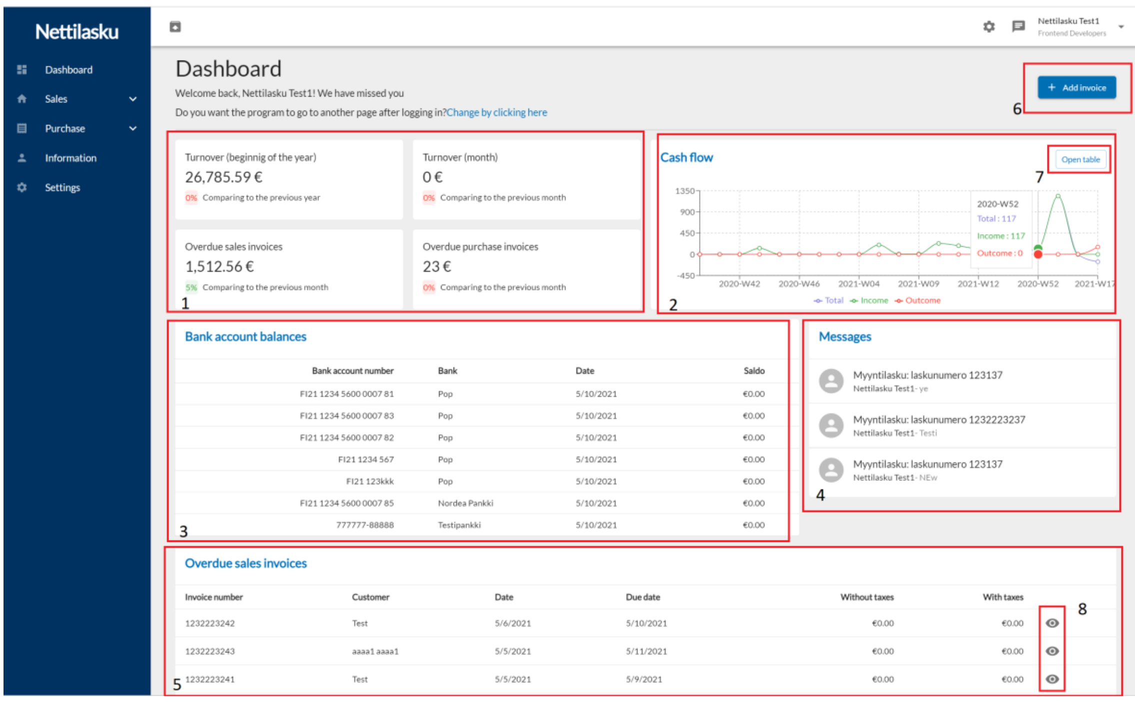

Figure 3. Solution dashboard: Cards section (1), Cash flow (2), Bank accounts table (3), Messages section (4), Overdue sales invoices table (5), Add invoice button (6), Cash flow action button (7), Overdue sales invoices view buttons (8)

The main structure is based on the Material Kit Pro V4 [8] example where the components are positioned asymmetrically about the major axes using MUI Grid elements. It allows to easily change the positions or add new dashboard parts.

Statistical numbers (turnover sums and overdue invoices totals) are presented as a set of cards that are located at the top left corner of the dashboard (figure 3, 1). This section is a combination of Material Kit Pro V4 [8] cards (figure 1, 1), the Transaction table (figure 1, 3), and the idea of putting all 4 required number-type items in one place. Every card follows the examples' card concept and displays the main number in the center, the corresponding label over it, and additional statistical information under it. Additional statistical data is highlighted with green or red colors as it was done in the Transaction table (figure 1, 3).

Cash flow (figure 3, 2), Messages section (figure 3, 4), Overdue purchase invoices (figure 3, 5), and Bank accounts (figure 3, 3) tables were done based on the native MUI template due to its simplicity and code availability. CashFlow as it was mentioned earlier used “recharts” for the line chart displaying and this code solution was improved due to the requirement of displaying 3 lines: total, income, and outcome. Tables and Messages are made using Table and List MUI components taken from the MUI website [5].

The colors, border radiuses, paper components, etc. were stylish using the MUI Theme global variable [11] since it contains all the corresponding to the current design patterns elements which ensured styles match.

Additionally, action buttons were added to improve the user experience. They transfer the user to Create invoice page(Add invoice button (figure 3, 6)), Invoices page (Cash flow action button (figure 3, 7)), and View Invoice page (Overdue sales invoices view buttons (figure 3, 8)).

The data for the dashboard was fetched from nettilasku REST API.

4.1.3 Conclusion

The aim of creating a dashboard that displays the specified set of data was achieved. The process of building the feature took about 35 hours during the first and the second thesis weeks.

The investigation required researching Native Material UI and Material Kit Pro v4 dashboard examples and selecting parts of the interfaces that are matching the initial needs set by the management team.

The implementation process involved designing a custom dashboard interface based on the researched examples of design patterns. The most difficult part was at creating statistical one-number data cards because the management team required putting four pieces of them but the investigated solutions had a maximum of two.

The implemented solution is easy to maintain and reuse because the dashboard parts are based on MUI elements and positioned by using the Grid MUI component that allows adding, deleting, or moving structural pieces.

The created dashboard matched initial needs and was accepted by the management. Additionally, it extended the functionality by adding action buttons that improved the user experience.

The feature was tested manually by the management team and merged to the production product version.

4.2 “Accounting accounts” page integration from the older nettilasku application

The accounting account page (figure 4) is a server-side paginated list view of accounting accounts with a search field and the possibility of editing/adding items. The task was to implement a view with the same options as the current project.

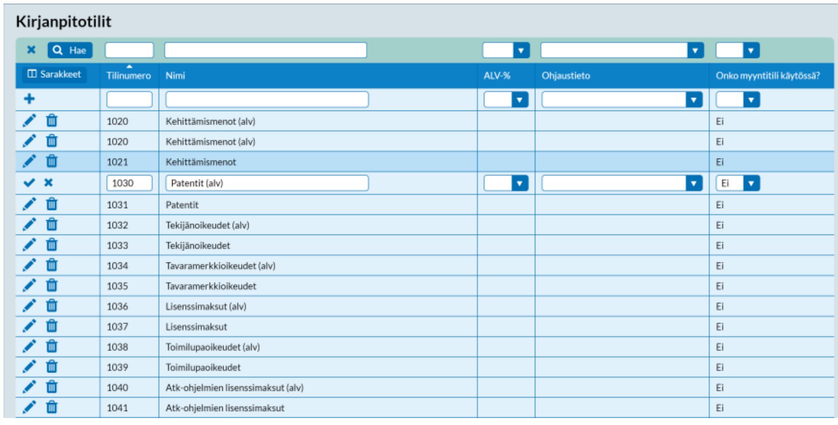

Figure 4. Older "Accounting account" page

4.2.1 Implementation

The implementation was done in two steps:

- Creating a server-side paginated table for showing and searching accounting accounts.
- Implementing a form for editing and creating accounting account items.

4.2.1.1 Table creation

The table could be implemented in two ways: by implementing a table from the scratch or by applying the SmartTable component. SmartTable is an improved version of the “mui-datatables” [12] Table that is based on the MUI Table and was created in the project before. It provides features for searching, server-side pagination, row actions addition, and many others. Due to the set of helpers that the second solution provides, it was selected for creating a list view.

The table creation required passing the required table control properties to SmartTable such as data, columns, and “serverSidePagination”. To add a search bar SearchField utility of SmartTable was used. It gets variables for controlling table filtering and returns an input field that is capable of searching. To achieve the effect of server-side pagination the data for the table was fetched using the useGetTableData hook. It is based on the useQuery “react-query” [7] hook where the “keepPreviousData” option is set to true which allows caching the previously fetched data. The code is shown in figure 5.

```
<SmartTable
  serverSidePagination
  columns={columns}
  data={tableData && tableData.data}
  toolbarLoad={[
    {
      name: 'search',
      custom: (
        <SearchField
          searchField={'name'}
          setPage={setPage}
          filters={filters}
          setFilters={setFilters}
          placeholder={intl.formatMessage({
            id: 'actions.search',
            defaultMessage: 'Search',
          })}
        />
      ),
    },
  ]}
/>
```

Figure 5. SmartTable code realisation

The front-end table view is shown in figure 6.

Accounting

Accounting accounts

+ Add accounting account

Search

Account number	Name	VAT	Control information	Is sales account active	Actions
123	Test	vat10		False	
123	Dmitriii	vat14		False	
1962	Nettilaskun saldo	vat24	nettilasku_saldo	True	
1020	Kehittämismenot (alv)			False	
1020	Kehittämismenot (alv)			False	
3005	12%	vat12		True	
29363	EU-palvelu kännetyn arvonlisäveron velka		oston_eu_palvelu_alv	False	
29362	EU-tavara kännetyn arvonlisäveron velka		oston_eu_tavara_alv	False	
29361	Rakennusalan kännetyn arvonlisäveron velka		oston_rak_kaant_alv	False	
2992	Myynnin arvonlisäverovelka 10%	vat10		True	

Rows per page: 10 0-10 of 1470

Figure 6. “Accounting accounts” implemented table

4.2.1.2 Form creation

For the form creation was decided to use SmartPopup and SmartEditForm components since SmartTable does not provide the opportunity for editing the table rows as it was done in the previous nettilasku version (figure 4).

SmartPopup and SmartEditForm were built in the nettilasku project earlier. SmartPopup is an improved MUI Dialog component that allows showing certain information blocks via a pop-up box. SmarEditForm is an updated Formik form that can automatically create the form out of the back-end response.

To build the form view SmartEditForm was passed as a child to the SmartPopup component together with the necessary fetched data and the SmartPopup toggle was connected to the onClick listener of “Create” and “Edit” accounting account actions. The form submission was made via the createOrUpdateAccountingAccount function. It checks if the id presents in the values and based on that sends the information to the corresponding endpoint. It is not presented in figure 7 because it is just a simple submit function with one additional “if” statement. The code of the form is presented in figure 7.

```
<SmartPopup
  open={isOpenPopup}
  setOpen={setIsOpenPopup}
  title={
    id
      ? intl.formatMessage(
          ...common.actions.edit_something(
            intl.formatMessage(
              ...common.models.accounting_account(1, 'edit')
            )
          )
        )
      : intl.formatMessage(
          ...common.actions.add_something(
            intl.formatMessage(
              ...common.models.accounting_account(1, 'add')
            )
          )
        )
  }
  hideActions
>
  <SmartEditForm
    columns={accountingAccountsColumns}
    onSubmit={async (values: any) => {
      if (!id) {
        await createOrUpdateAccountingAccount(values, {});
      }
    }}
    initialValues={!id ? {} : accountingAccountData.data}
  />
</SmartPopup>
```

Figure 7. Code solution for the form

The front-end view of the form can be seen in figure 8.

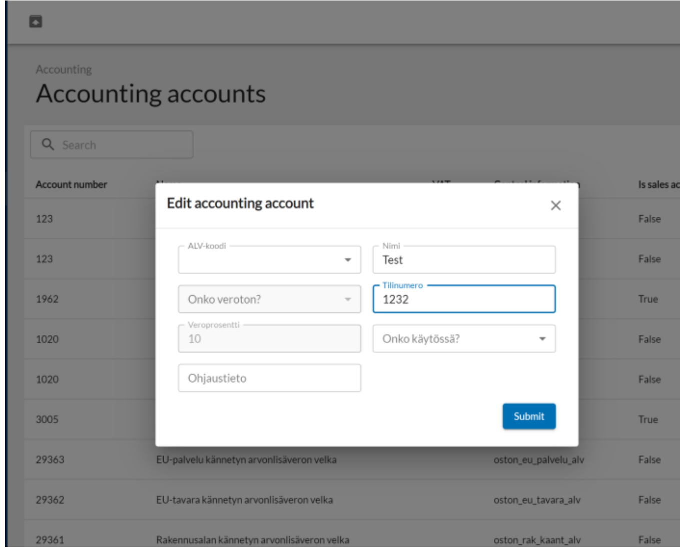

Figure 8. Front-end solution for the form

To submit the form a user needs to fill the needed fields and press the “Submit” button.

4.2.2 Conclusion

Manual testing of the feature by interacting with the implemented elements (adding and editing items, changing tables pages, searching using the search field) concluded that the solution is implemented correctly.

The feature integration involved the application feature research in the older application and looking for the most suitable elements that can be used in the project to recreate the same functionality.

The implementation process took about 15 hours during the second week of the thesis and involved applied already used principles of nettilasku page creation to new front-end elements. It required an investigation of SmartPopup, SmartEditForm, and SmartTable elements and building a solution using them. Using these components instead of the usual MUI also resolved a maintainability issue since they are implemented specifically for building interfaces with tables and forms.

The solution matched initial needs, was accepted by the management, and merged to the public test branch.

4.3 “Intercom” pop-up integration

“Intercom” is a tool provided by Intercom company [13] which connects users and support team on websites. It usually appears in the right bottom corner of the page and may contain a chat with a company representative, help menu, or bot. The example is shown in figure 9.

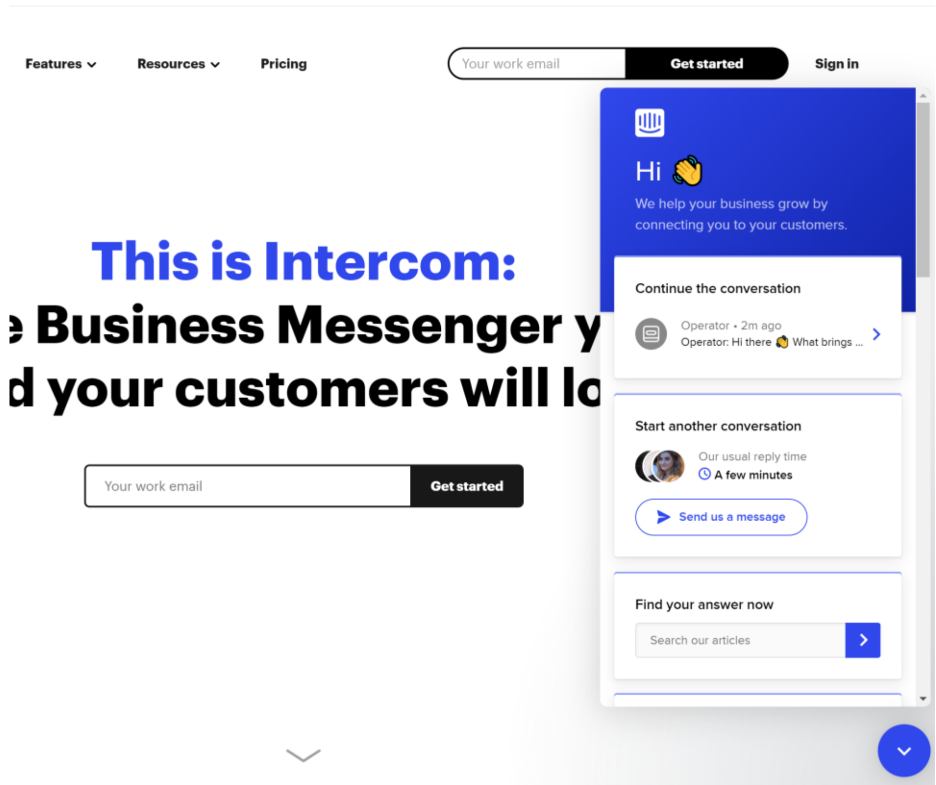

Figure 9. Intercom example

The goal of the author was to investigate existing front-end libraries since there is no official library by Intercom [13] and implement an intercom application in the way that it appears when the user is logged into the nettilasku application.

4.3.1 Investigation of the existing libraries

The project uses React for building front-end interfaces and due to that the research was done via node-package manager [14] – an open-source tool for sharing JS packaged modules of code.

The investigation by searching via NPM showed that there are 3 matching libraries: react-intercom, intercom-react, and react-use-intercom. The first two of them have not been maintained for more than 2 years due to the "last published" parameter and it was decided to use react-use-intercom which had the latest release this month.

4.3.2 Implementation

The chosen library [15] works based on the React context [16] principle that needs to wrap the part of the application where its functionality can be accessed in "ContextProvider". React context allows using variables and functions from any ancestors of a wrapper component via special functions called "hooks" instead of passing everything using props from the parent element to children which significantly components maintainability. Since there is no way of getting intercom data by an unauthorized user because of the back-end specification the AuthenticatedApp component was wrapped into IntercomProvider. Next fetched from the back-end "appId" was passed to this element due to the docs requirements [15]. The code is shown in figure 10.

```
<IntercomProvider appId={intercomData.data.app_id}>
  <AuthenticatedApp
    unreadMessagesCount={unreadMessagesCount}
    locale={locale}
    setLocale={setLocale}
    user={user}
    company={company}
  />
</IntercomProvider>
```

Figure 10. Intercom provider wrapper

Now Intercom functions are available in the AuthenticatedApp component via the "useIntercom" hook.

The next thing was to make the pop-up active when the user is logged in. Due to [17] the only way to ignite the function only once after the initial component render is to put it into useEffect React hook with an empty array as a second parameter. In our case, it should be done in the AuthenticatedApp component since we want the user to see it after logging in. The code of activating the Intercom is presented in figure 11.

```
const { boot } = useIntercom();
const bootWithProps = () =>
  boot({
    name: intercomData.data.name,
    customAttributes: intercomData.data,
  });
useEffect(() => {
  bootWithProps();
}, []);
```

Figure 11. Intercom ignition mechanism

In figure 11 “bootWithProps” function passes predicatively fetched data to the intercom “boot” function which ensures the correctness of user data when the application starts. That also makes information available for the support team when the user contacting it via the application.

The front-end part of the solution is presented in figure 12.

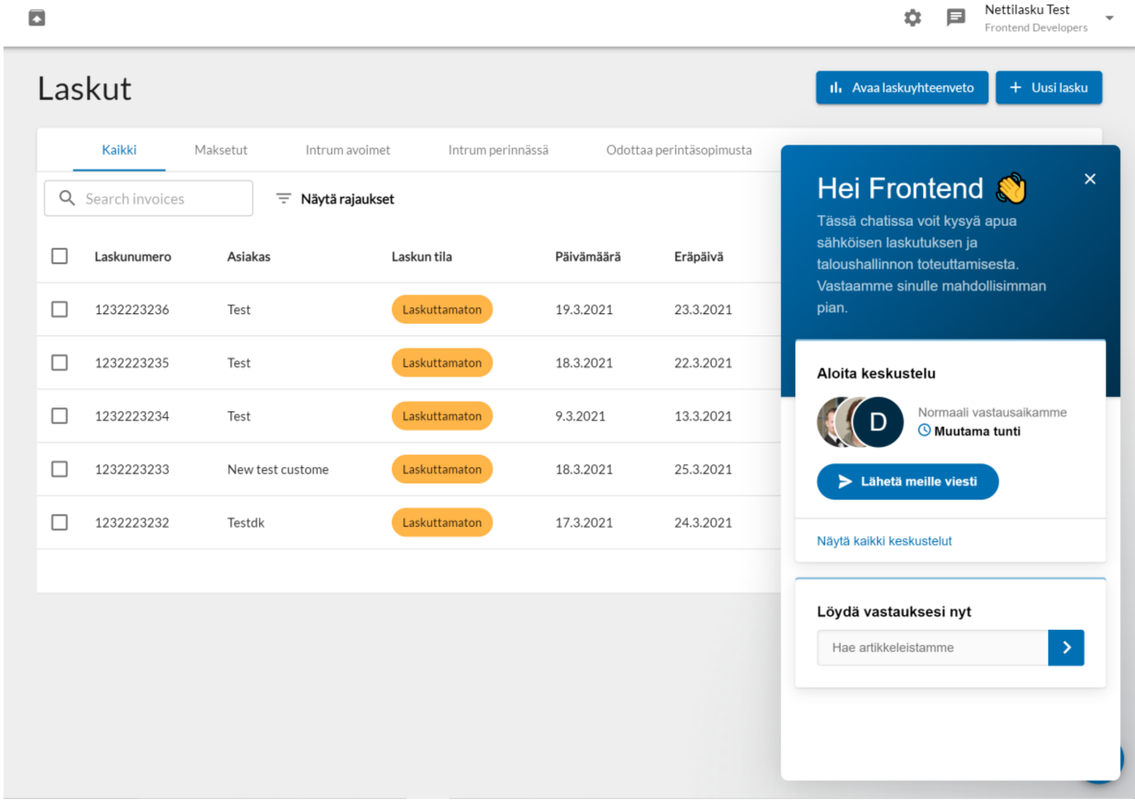

Figure 12. Intercom pop-up on the authenticated page

Figure 12 shows the page after logging in. As can be seen, Intercom uses the right company name which ensures the parameters passed to it in figure 11 are correct.

4.3.3 Conclusion

Testing the feature by logging out and logging in showed that the pop-up appears after the “AuthenticatedApp” component is rendered and shows the right information. It concludes that the feature was implemented correctly.

The feature creation required investigating the existing intercom-related libraries, principles of React Context, and React Hooks. Due to the researched data, a working Intercom solution was implemented that also allows accessing the functionality of the pop-up all over the authorized part of the application which makes it extendable in terms of new Intercom features addition.

The process of building the feature took about 15 hours during the second thesis weeks.

The solution was approved and merged to the production version.

4.4 “Purchase invoice” section implementation

The purchase section is a part of nettilasku that is responsible for handling receipts and invoices received from the outside. The user should be able to see the received items in the list views and edit/create items using the form. The goal was to integrate described functionality into this project.

Implementation was done in 2 steps:

- Creating section architecture
- Creating table and form

Every step is described due to the main solutions that were applied there.

4.4.1 Creating section architecture

The architecture implementation included basic components, routing, and form submission hook creation.

4.4.1.1 Basic components creation

The older “Purchase” section consists of 2 pages: “PurchaseInvoices” and “Receipts”. Both include two main views: a table and a form. Visual interface comparison (figure 13 and figure 14) revealed that each of them is based on the same components but display different data. That required creating “ViewPurchaseInvoice“ (for showing the table) and “Purchase” (for showing the form) universal components.

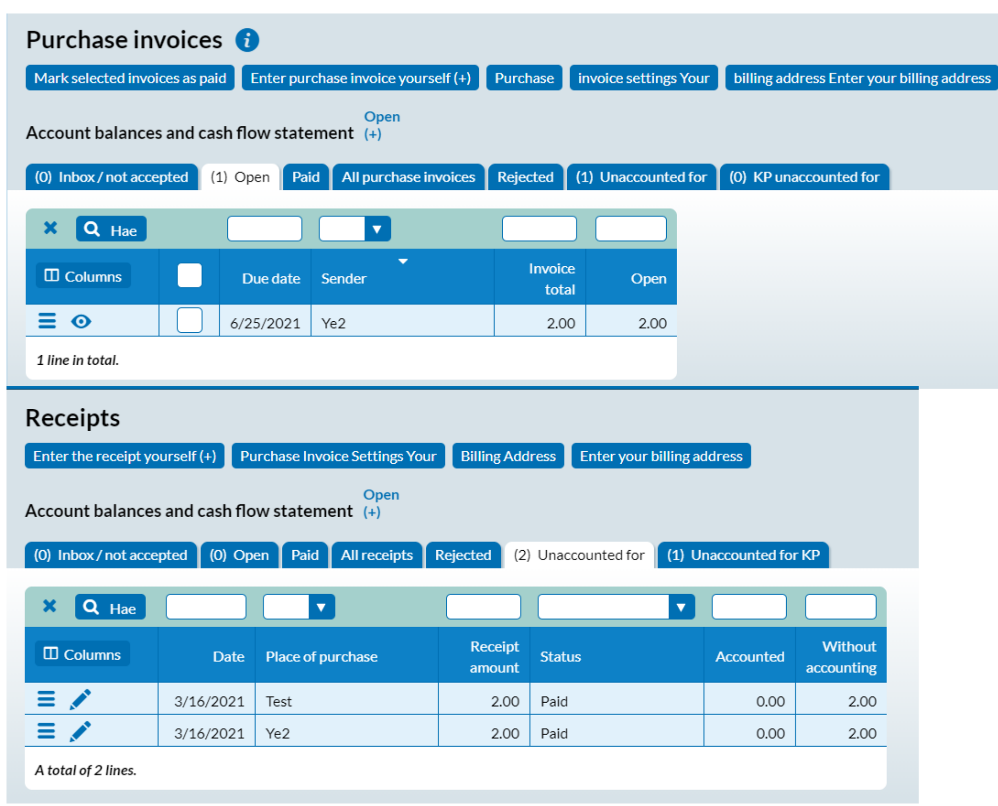

Figure 13. “Purchase invoices”(at the top) and “receipts” tables(at the bottom)

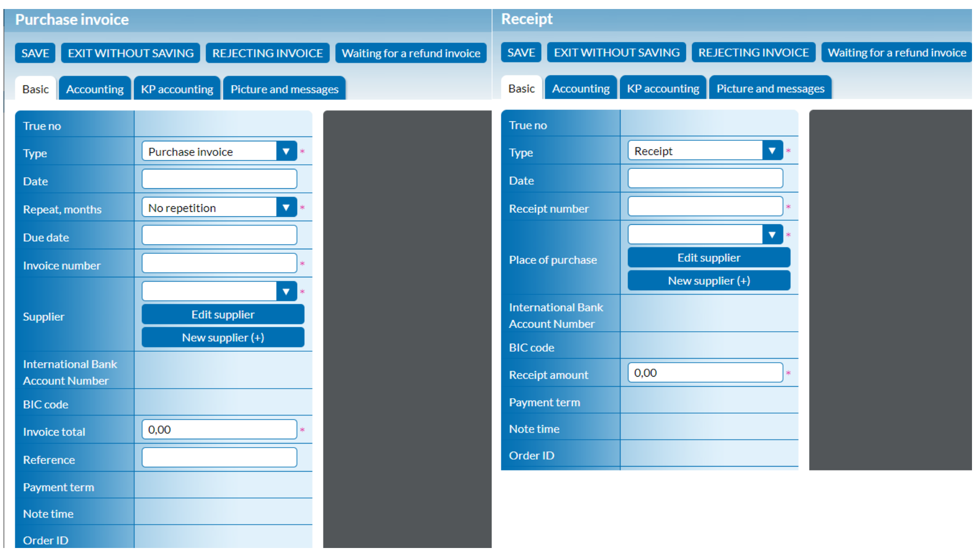

Figure 14. “Purchase invoice” (at the left) and “receipt” (at the right) forms

The components were made as empty React pages that fetch different data based on the URL. To receive correct props the components use the getPurchaseViewType (figure 15).

```
export const getPurchaseViewType = (): {
  endpoint: EndpointsConfig;
  viewType: 'receipt' | 'purchase' | 'other';
} => {
  if (window.location.href.indexOf('purchase_invoices') !== -1) {
    return {
      endpoint: endpoints.purchase_invoice,
      viewType: 'purchase',
    };
  }
  if (window.location.href.indexOf('receipt') !== -1) {
    return { endpoint: endpoints.purchase_receipt, viewType: 'receipt' };
  }
  return { endpoint: endpoints.purchase_invoice, viewType: 'other' };
};
```

Figure 15. getPurchaseViewType fucntion

The function is responsible for differentiating between “purchase” and “receipt” pages. It checks if the current URL matches “receipt” or “purchase” keywords and based on that returns 2 variables: “endpoint” (for fetching data from the

“Purchase” and “PurchaseInvoices” components) and “viewType” (for correct header displaying).

The basic components creation resolved the issue of getting the correct information for the future front-end elements that will be built onto “PurchaseInvoices” and “Purchase” components.

4.4.1.2 Creation of the routing system

To dynamically render the components routing system based on React Router was created. React Router is the standard routing library in React. It keeps the application interface in sync with the URL on the browser. To render a view Router components (figure 16) were passed to the Switch(not seen in figure 16 because Switch wraps all Router elements of the application) wrapper element with specified “path” parameters. When a user accesses a certain path the corresponding view is returned.

```
<Route
  exact
  path="/purchase/purchase_invoices/:tab/:id"
  component={ViewPurchaseInvoice}
/>
<Route
  exact
  path="/purchase/receipts/:tab/:id"
  component={ViewPurchaseInvoice}
/>
<Route exact path="/purchase/receipts" component={Purchase} />
<Route
  exact
  path="/purchase/purchase_invoices"
  component={Purchase}
/>
```

Figure 16. Routes for "Purchase" and "ViewPurchaseInvoice" components

The routing system resolved the issue of accessing created universal views in the front-end.

4.4.1.3 Form submission hook creation

To handle the form submission “usePostPurchase” (figure 17) hook was implemented. It resolved the issue of making different POST requests when creating/editing new invoices or receipts. Based on “id” and “type” variables that are passed with other values via the “data” object it posts the information to the corresponding endpoint and returns it with useMutation [18] react-query hook as it is shown in figure 17. The useMutation hook is a utility used to send data to the server. It contains functions and variables that facilitate the process of interacting with the back-end in the react application.

```
export const usePostPurchase = (options = {}) =>
  useMutation((data: any) => {
    if (data.type === 'purchase') {
      return API.post(
        encodeURI(`/purchase_invoice${data.id ? `/${data.id}` : ''}`),
        data
      ).then((res) => {
        queryClient.invalidateQueries('purchase_invoice');
        return res.data;
      });
    } else {
      return API.post(
        encodeURI(`/purchase_receipt${data.id ? `/${data.id}` : ''}`),
        data
      ).then((res) => {
        return res.data;
      });
    }
  }, options);
```

Figure 17. usePostPurchase hook for the form submission

The provided architecture represents a universal component-based system that avoids code repetition and creates a core for adding front-end elements on top of it.

4.4.2 Creating table and form components

To implement the interfaces for “Purchase” and “ViewPurchaseInvoice” Material UI, mui-datatables [12] and Formik [6] NPM packages were used.

The table was done using the “SmartTable” component which was described earlier in the thesis(Chapter 4.2.1.1). It is an improved version of the native MUI Table which features (selectable rows, server-side rendering, and table actions) were used for the interface creation. The only thing that was not provided by the component but which was necessary to implement is tabs for filtering list items. The missing element was implemented by using MUI Tabs components after investigation on its documentation [19]. The solution is presented in figure 18.

```
const SmartTabs: React.FC<InvoicesTableTabsProps> = ({
  tabValue,
  setTabValue,
  tabsData,
}: InvoicesTableTabsProps) => {
  const handleChange = (
    event: React.ChangeEvent<Record<string, unknown>>,
    newValue: string
  ) => {
    setTabValue(newValue);
  };
  return (
    <Tabs
      value={tabValue}
      onChange={handleChange}
      variant={'scrollable'}
      scrollButtons={'auto'}
    >
      {tabsData.map((el: any, key: number) => {
        return <Tab key={key} label={el.title} value={el.name} />;
      })}
    </Tabs>
  );
};
```

Figure 18. "SmartTabs" component for displaying SmartTable tabs

The function(figure 18) receives ”tabValue” and “setTabValue” states for controlling the active tab and “tabsData” array that includes object with title and name properties. The component returns the MUI element which maps through “tabsData” and automatically builds the required tabs.

The front-end final table solution is shown in figure 19.

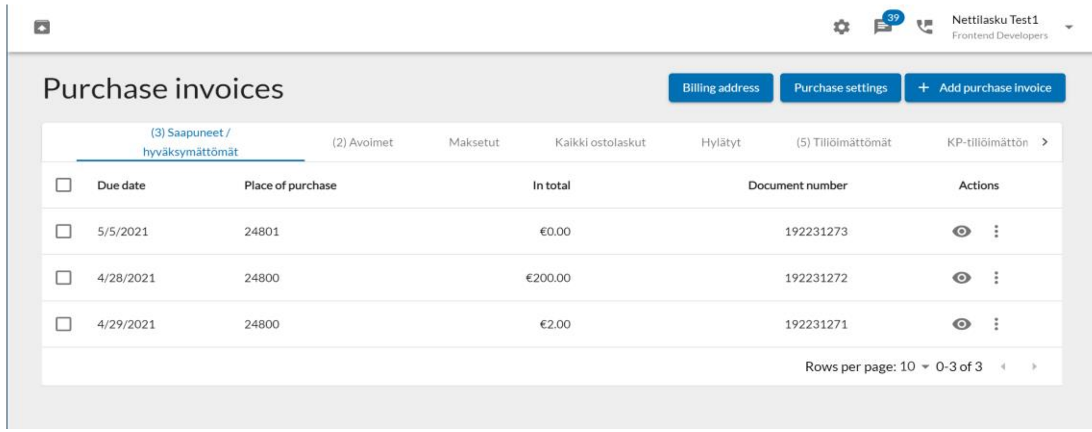

Figure 19. Purchase invoices solution table

To access the form user needs to click the “Eye” button placed at the end of each table row. All other actions are not supported and made up due to back-end issues.

The ”Purchase” form was done by combining MUI Tabs [19], Formik [6], and SmartInputField components. SmartInputField is a functional component that was implemented earlier for the nettilasku application. It receives a set of fetched from the back-end properties together with Formik variables and returns the corresponding styled input field.

The solution works in the way that Formik passes its properties for form handling (“values”, “onChange”, etc) to the Tabs component which fetches additional data and passed the received form and back-end data to SmartInputFields. The example of an input creation for the form is presented in figure 20.

```
<SmartInputField
  value={values.purchase_invoice_supplier_refid}
  name="purchase_invoice_supplier_refid"
  onChange={onChange}
  columns={columns}
  error={errors.purchase_invoice_supplier_refid}
  label="Invoice supplier refid"
/>
```

Figure 20. "Invoice supplier refid" automatically formed input field

“Invoice supplier refid” example reflects the way how the fields are created via SmartInputField. The Formik “values”, “onChange” and “errors” states are used for building the mechanisms of the input (changing values, showing error), and fetched from the back-end “columns” value in cooperation with “name” is used for defining the properties of the inputs (type of the field, min, max, etc.).

The front-end form implementation is presented in figure 21.

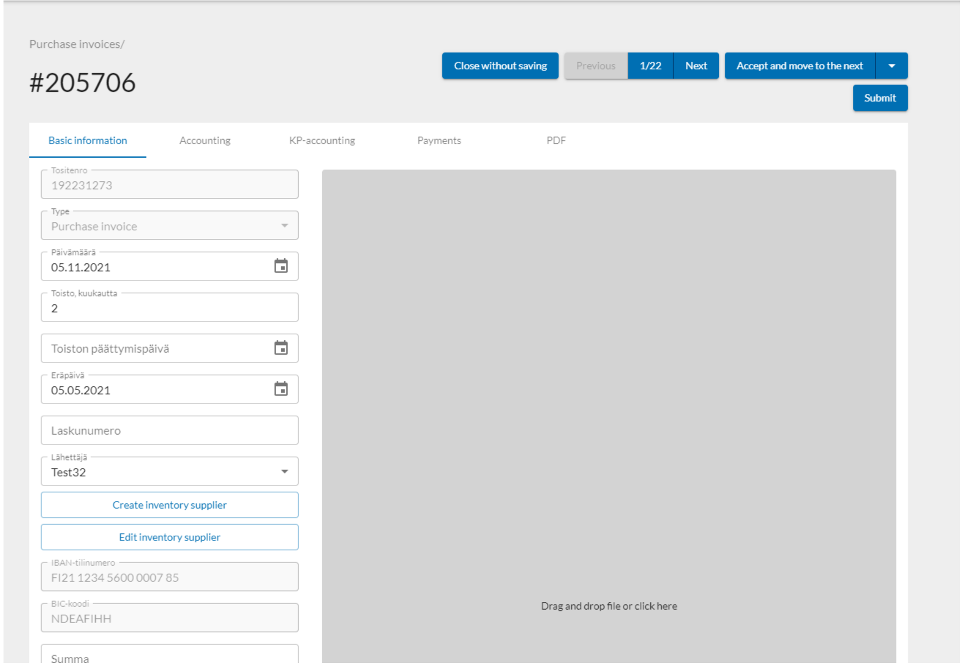

Figure 21. Invoice form solution view

“Accounting”, “KP-accounting”, “Payments, and “PDF” tabs are not presented in figure 21 because they are built using the same principles and behave similarly.

For submitting the form user needs either to click the “Enter” or “Submit” button at the top right corner of the view. It triggers passing the values to the “usePostPurchase” hook described earlier in this chapter which is responsible for the form submission.

4.4.3 Conclusion

The feature was tested manually: the architecture was checked by accessing the page using URL and seeing what components returned. The table was tested by changing the pages and accessing the form via actions. The form was tested by interacting with different fields and submitting the form in different states. Tabs were checked by switching between them. Additionally, after the manual tests made by the author, the management verified the functionality in the test branch. The testing concluded a lack of critical bugs and ensured that the main features work as expected.

The feature integration involved investigation on the principle of creating components architecture via React Router, universal components, hooks, URL interaction methods, and decoupling programming term for creating the solution architecture and research on SmartInputField, Formik, Tabs, and SmartTable for creating the front-end.

The implementation required creating a custom solution that allows reusing the same components for many views. The most challenging thing was creating the features architecture since it required building universal components for 2 pages that had not been practiced in the project before.

The process of building the feature took about 90 hours during the sixth, seventh, and eighth thesis weeks. It was approved by management and merged to the production version.

4.5 Invoice folder feature integration

In the older version of the nettilasku web application invoice folder was one of the key features that allowed to use of limited program functionality to unauthorized users. My goal was to integrate this feature into the updated nettilasku version.

4.5.1 How “invoice folder” feature works in the older application version

From the user perspective, the “invoice folder” feature works in the following way:

1. Nettilasku user creates an invoice or selects multiple invoices and chooses a recipient
2. The user sends the invoice using the “Send invoice” option
3. If the recipient is not a nettilasku user he receives an email with the link to the invoice.
4. When the recipient opens the link he is transferred to the limited nettilasku version(invoice folder) where he can see the invoice(s) he received and send a message to the sender without logging in.

The “invoice folder” provides a “safe” area that includes only a set of certain features and does not allow the user to be able to do anything with data except for reading it.

From the technical aspect, the link that is sent to the recipient contains URL parameters that are used to fetch the JWT token that is used for and getting the information from the back-end. This token is empowered with constrained rights and the back-end will not provide data that is not related to this particular user.

4.5.2 Nettilasku architecture

For building such a low-level application feature it is necessary to understand the architecture of the current project.

The application is based on the App component which renders either the Authorization part (Login/Register/Forgot password pages) or Authorized part (all other views). To decide what part needs to be displayed GET request to /user/self API endpoint is made. Every API request made from the front-end to the back-end contains a JWT token in its header. JWT is hashed information that is provided by the server and usually contains information about the session (when it started and when it should be destroyed) and the user. If it is absent or expired the server returns error which means that the user is unauthorized. In this case, the application renders the Authorization part and the user is requested to provide credentials for being able to access the full functionality of the application.

In the project, the JWT token is fetched and stored in the localStorage [20] after the user entered the right credentials.

The application architecture described above is presented in figure 23.

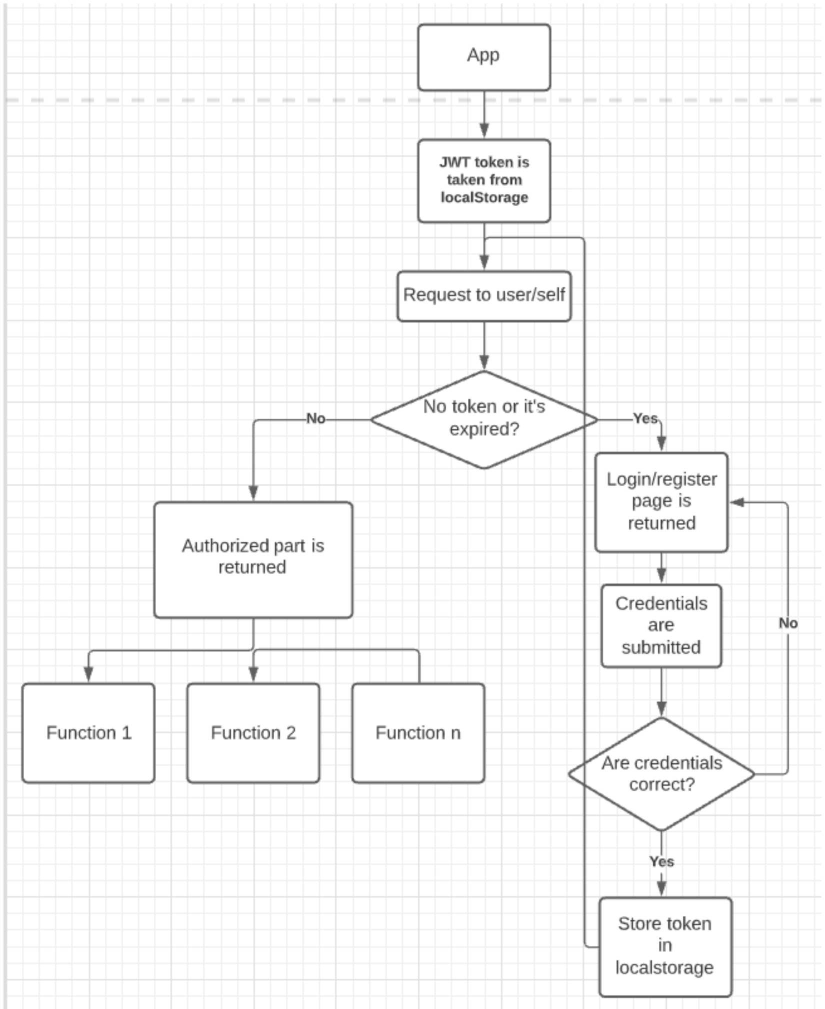

Figure 22. Nettilasku current architecture

4.5.3 Implementation

After investigation of the current application architecture, two ways of resolving the issue were determined: to let the user access the full application with restricted token rights and to create a restricted copy of the application and redirect the user there.

The first solution was considered to be unsafe despite its simplicity. The user will not be able to do anything because the back-end will be returning errors to most of the requests, but this might create security vulnerabilities because the

back-end will be overloaded with error requests. That can lead to the situation when the user accidentally accesses or edit data that he is not supposed to see.

The second solution of creating copies of necessary components with limited functionality is the optimal one. It does not have the parts of the application that should be hidden and it is easier to debug. But it requires more work with adding an authorization branch that will be redirecting the user that followed the link from the email.

The solution implementation can be seen in figure 24.

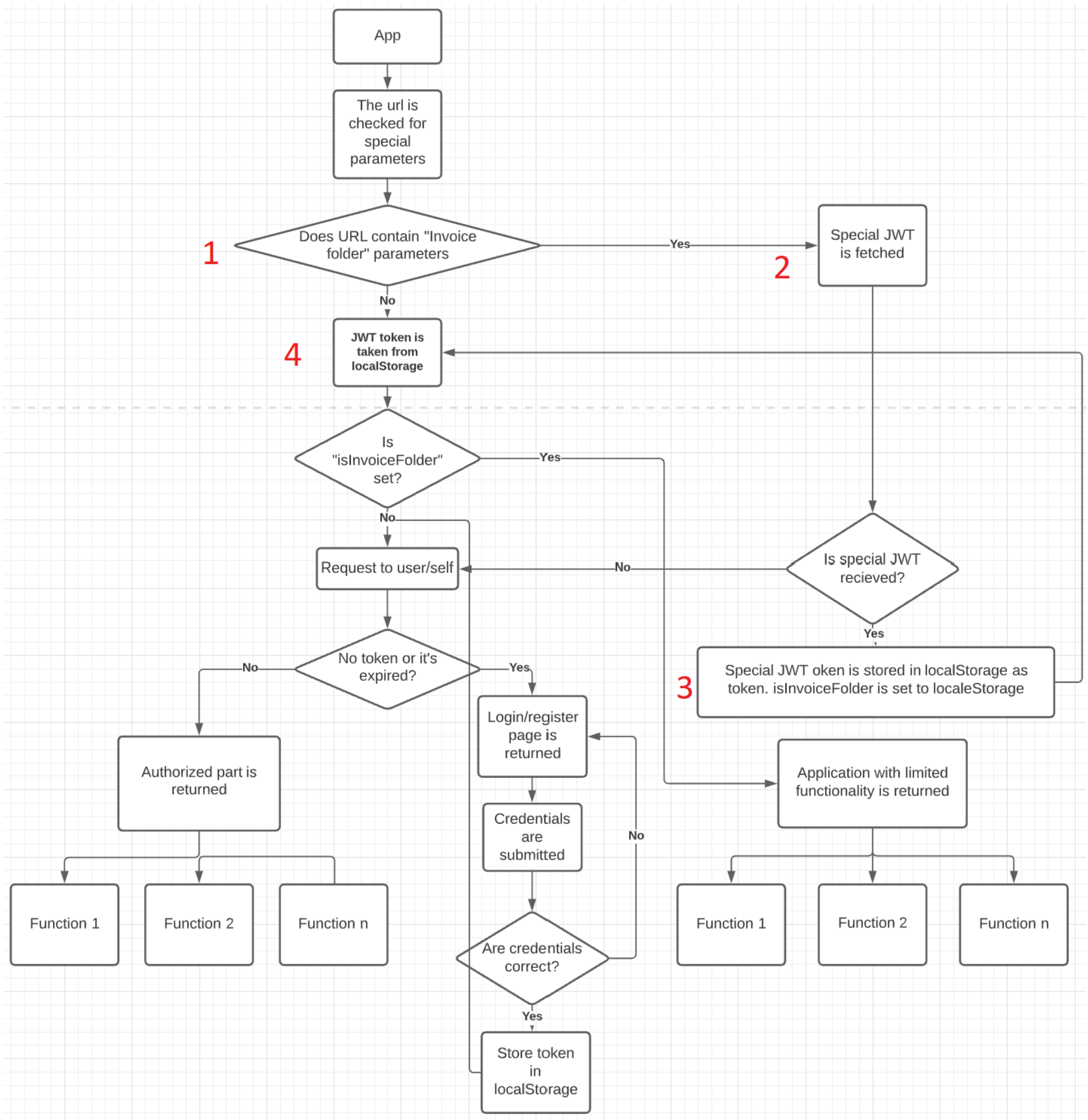

Figure 23. "Invoice folder" solution architecture: Invoice folder verification statement (1), Special JWT fetching block (2), Special JWT and isInvoiceFolder saving block (3), Starting block of usual authentication logic with added "isInvoiceFolder" statement (4).

Figure 14 shows the logic of the applied solution. "Does URL contain 'Invoice folder' parameters" (figure 24, 1) checks if the application is opened via the link with special parameters in the URL and based on them makes a GET request to the back-end (figure 24, 2). If the parameters are correct JWT token with

constrained user rights is returned and saved to the localStorage with isInvoiceFolder variable(figure 24, 3). Then the arrow leads to the previous authorization logic(figure 24, 4) where an additional check for the isInvoiceFolder. If the isInvoiceFolder variable setting is determined the user is redirected to the limited application functions. In the other case, the authorization continues as usual.

The solution allows the user to use the restricted application as a normal one. Additionally, the authorization branch logic is operating before sending a request to the “user/self” (figure 24, 4) which ensures the safety of the full application. If a user accidentally deletes the isInvoiceFolder variable from the local storage GET user/self request will always return an error because the token does not have rights for fetching this data and the customer will be redirected to the Login page.

The technical part of the solution was implemented by following the logic defined above and the basic principles of programming. The code creation included adding if statements to the App component, creating InvoiceFolderView, InvoiceFolderInvoices components by copying and deleting unnecessary functionality from existing elements, and adding some restrictions to SideMenu and Topbar components. The front-end representation of the solution can be seen in figure 25. Additionally “not restricted” version of the application is shown in figure 26 for a better understanding of how the implemented feature limits the functionality of the usual application.

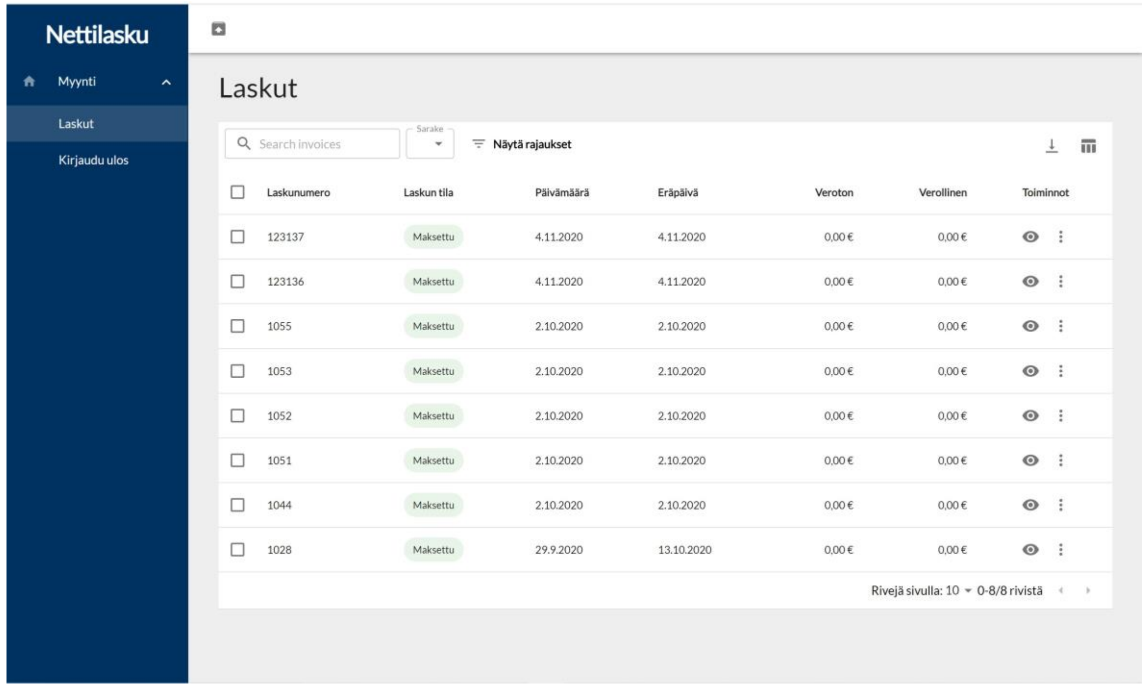

Figure 24. "Invoice folder" front-end view

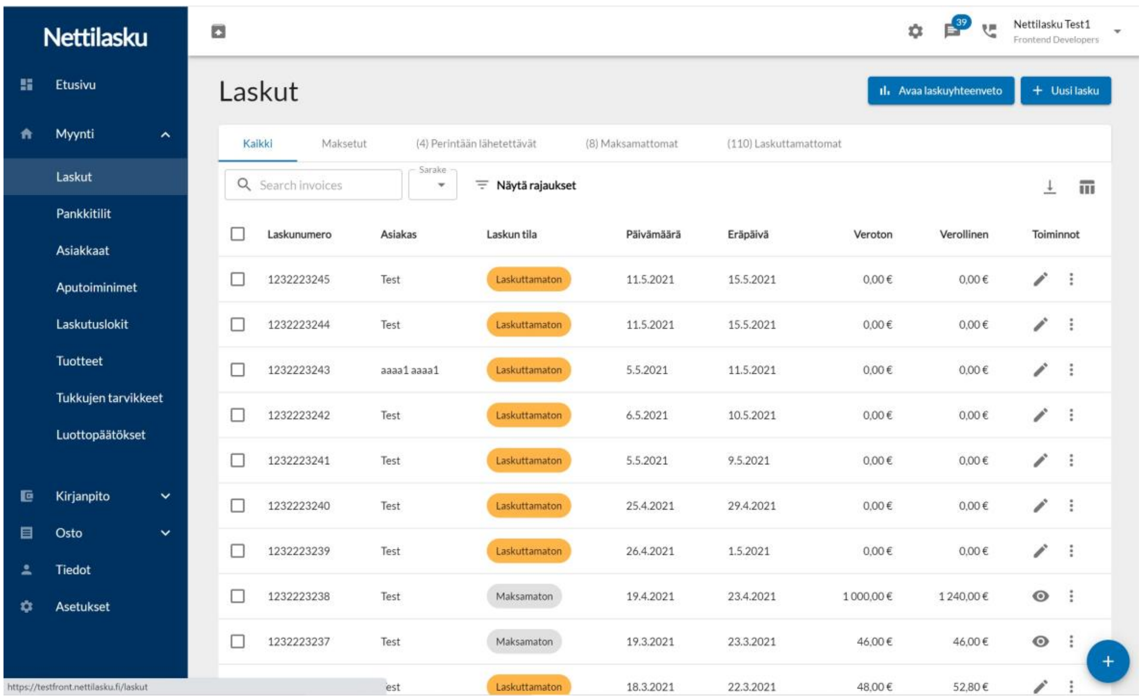

Figure 25. Normal nettilasku.fi front-end view

4.5.4 Conclusion

The implementation of the feature correctness was checked manually firstly by the developer via opening the application using the “invoice folder” link and testing the available options and secondly by the management team in the test branch. There was found a small bug (showed “Send invoice” button that is not supposed to be seen by the user) which was fixed. In other cases, the feature was working as expected.

The implementation required investigation of the project architecture, principles of JWT tokens, and building UML diagrams. The most challenging part of the creation process was designing the solution because it involved creating a second smaller “safe” application area where the user is supposed to be transferred by opening a special link.

The build solution is extendable in terms of adding more features to the “invoice folder” feature due to its architectural comprehensibility. It can be also reused if there is a need of adding similar functionality to the project in the future.

The solution was approved by management and published in the public test project branch.

The process of building the feature took about 60 hours during the second and third thesis weeks.

5 CONCLUSION

The aim of implementing new functionality for the final version of the nettilasku product was achieved. It was done by investigating the principles of the older application features, applying skills of building React programs, and finding methods for implementing enhanced solutions for the encountered issues using required tools. The results of the work were highly appreciated by the management. 3 out of 5 developed solutions were published in the final product and 2 of the remaining ones are in the open testing stage.

Concepts of building and improving React applications using Typescript and Material UI as a front-end library were deeply studied during the implementation process. Also, an important part of the learning process was studying various NPM libraries and their practical application in the project.

The solutions provided in the thesis are maintainable because they are based on the required set of tools and the principle of avoiding adding new NPM dependencies which decrease the chances of having problems with external libraries depreciation. Also, during the implementation, the principle of earlier created components using was applied that simplifies code extendibility and support.

The thesis can be considered applicable for the further improvements of the product. The development path entailed in it clearly describes the main principles of the older application, the way how they are integrated into this project, and new functionality implementation. The researched methods allow extending the application features by improving the current solutions or building new ones.

REFERENCES

[1] Fonecta, "Finder," [Online]. Available: https://www.finder.fi/. [Accessed 11 May 2021].

[2] Wikipedia, "React(JavaScript library)," 2020. [Online]. Available: https://en.wikipedia.org/wiki/React_(JavaScript_library). [Accessed 15 February 2021].

[3] Microsoft, "What is Typescript?," [Online]. Available: https://www.typescriptlang.org/. [Accessed 10 May 2021].

[4] Google, "Introduction," [Online]. Available: https://material.io/design/introduction#principles. [Accessed 10 May 2021].

[5] Material UI, "Material UI," 2021. [Online]. Available: https://material-ui.com/. [Accessed 10 May 2021].

[6] Formium, "Formik docs," 2020. [Online]. Available: https://formik.org/. [Accessed 10 May 2021].

[7] T. Linsley, "React-query overview," 2020. [Online]. Available: https://react-query.tanstack.com/overview. [Accessed 10 May 2021].

[8] Material-UI Store, "Devias Material Kit Pro - React Client & Admin Dashboard," 2020. [Online]. Available: https://material-ui.com/store/items/devias-kit-pro/. [Accessed 10 May 2021].

[9] Material UI, "Dashboard," 2021. [Online]. Available: https://material-ui.com/getting-started/templates/dashboard/. [Accessed 10 May 2021].

[10] Recharts Group, "Recharts," Recharts Group, 2016. [Online]. Available: https://recharts.org/en-US. [Accessed 10 May 2021].

[11] Material UI, "Theming," 2021. [Online]. Available: https://material-ui.com/customization/theming/. [Accessed 10 May 2021].

[12] GitHub Inc., "MUI-Datatables - Datatables for Material-UI," 2020. [Online]. Available: https://github.com/gregnb/mui-datatables. [Accessed 11 May 2021].

[13] Intercom, "The world's first Conversational Relationship Platform," 2011. [Online]. [Accessed 11 May 2021].

[14] npm, Inc., "Npm," 2020. [Online]. Available: https://www.npmjs.com/. [Accessed 11 May 2021].

[15] GitHub Inc., "React-use-intercom," [Online]. Available: https://github.com/devrnt/react-use-intercom#readme. [Accessed 11 May 2021].

[16] Facebook Inc., "Context," 2021. [Online]. Available: https://reactjs.org/docs/context.html. [Accessed 11 May 2021].

[17] P. E. Banks A, "Chapter 7. Enhancing Components with Hooks," in *Learning React, 2nd Edition*, 2nd ed., O'Reilly Media, Inc., 2020.

[18] T. Linsley, "useMutation," 2020. [Online]. Available: https://react-query.tanstack.com/reference/useMutation#_top. [Accessed 11 May 2021].

[19] Material-UI, "Tabs," 2021. [Online]. [Accessed 11 May 2021].

[20] Mozilla and individual contributors, "Window.localStorage," 2004. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. [Accessed 11 May 2021].