Improving Event Management Implementation on Odoo Business Management Software

Bachelor's Thesis

Information and Communications Technology, Häme University of Applied Sciences (HAMK)

Fall 2021

Miika Nissi

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka Tiivistelmä
Tekijä Miika Nissi Vuosi 2021
Työn nimi Tapahtumien hallinnan toteutuksen parantaminen Odoo liiketoimintasoveluksessa
Ohjaaja Toni Laitinen

Tämän opinnäytetyön tarkoituksena oli esitellä avoimen lähdekoodin Odoo (versio 14) toiminnanohjausjärjestelmää ja parantaa järjestelmän olemassa olevaa tapahtumien hallintaominaisuutta. Tämä opinnäytetyö käy aluksi läpi Odoon historiaa, sekä yrityksenä, että ohjelmistotuotteena. Tämän jälkeen perehdytään Odoon ohjelmistoarkkitehtuuriin sekä teknisiin ominaisuuksiin ja etenkin käsitellään kuinka Odoon ohjelmistokehitys toimii ja mitä välineitä, kirjastoja ja ohjelmointikieliä se vaatii. Lopuksi tarkastellaan työn tuloksia, haasteita ja jatkokehitysmahdollisuuksia.

Työn päätavoitteena oli mahdollistaa Odoon tapahtumien hallintaan jonotuslistatoiminnallisuus sekä yhdistää Odoon kyselymoduuli tapahtumien hallintamoduuliin, jotta tapahtumien ilmoittautujilta voidaan kysyä kysymyksiä ilmoittautumisen yhteydessä. Työssä tärkeää oli, että ominaisuutta on helppo jatkokehittää, ylläpitää sekä käyttää usean asiakkaan tarpeissa. Opinnäytetyön tilasi Tawasta Oy.

Avainsanat Odoo, ERP-järjestelmä, ohjelmistokehitys, tapahtumien hallinta
Sivut 36 sivua ja liitteitä 10 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology Abstract
Author Miika Nissi Year 2021
Subject Improving Event Management Implementation on Odoo Business Management Software
Supervisors Toni Laitinen

The purpose of this thesis was to give an introduction to an open source Enterprise Resource Planning (ERP) system, Odoo (version 14), and to improve upon its existing event management implementation. This thesis will first briefly cover the history of Odoo as a company and as a software product. It will then detail the architecture and technical features of Odoo and explore how Odoo software development works and what tools, libraries, and programming languages are used. In the end, the results of the work are shown and explored, along with the challenges faced during development and potential future improvements.

The main improvement goals for Odoo's event management were to add a waiting list feature and to integrate Odoo's survey module with an event module. Integrating the survey module with the event module enables asking questions during event registration. It was important that the new features were easy to further develop, maintain, and be able to be used for the needs of several clients. This thesis was commissioned by Tawasta Oy.

Keywords Odoo, ERP system, software development, event management
Pages 36 pages and appendices 10 pages

Contents

Figures, code snippets and graphs

Appendices

1 Introduction

This thesis looks at the development process of an extension module to Odoo (version 14) ERP system. I chose this subject since I had been introduced to Odoo ERP system during my internship at Tawasta and a client need for further development of the event management module came up. The development of event management module fit my current skill level and it gave me in-depth experience in programming as well as in ERP systems.

Odoo is an open source business application, which includes both Enterprise Resource Plannning system (ERP) as well as Customer Relationship Management system (CRM). It is built to be a modular system capable of fitting the needs of small to medium sized companies and organizations. Each installation of Odoo is built on a selection of modules tailored to the needs of the clients. A module is usually a single application or a feature that is added on top of the core Odoo platform. Examples of these modules include an event module to manage events and registrations, a sales module to aid in the sales process, and a website module to create a publicly visible website for the client (Figure 1). The modularity allows each client to only have the features they need while being easily manageable and expandable in the future.

Figure 1. Core Odoo modules developed by Odoo SA.

The first chapter will take a close look into Odoo, its functionality and technical features. It especially focuses on the event management module, which the implementation is going to to extend and improve upon. Since Odoo is an open source system, this chapter will explain in detail what it means for the development process and things that need to be taken into consideration as a developer.

Chapter 3 gives an overview of the development process and implementation of the event management improvements for Odoo. Specifically, what languages and tools are used, what needs to be taken into consideration in the planning phase and while programming. It is also going to cover the design and architecture of the implementation. At the end, the challenges faced while developing and some ideas and possibilities for future development are discussed.

This thesis was commissioned by Tawasta Oy, which is a part of Mindpolis Group. Tawasta has produced open source solutions to clients since 2004 and their current main product is an Odoo implementation called Futural product family. (Tawasta Oy, n.d.)

Futural product family is a Software as a Service (SaaS) implementation of Odoo with many extensions and additional features. Futural products are fully tailored to the needs of the clients and include everything from websites and online stores to student administration and management solutions. In addition, clients of Tawasta receive instructions and support for the use of the Odoo system. (Tawasta Oy, n.d.)

2 Odoo

Odoo is an open source suite of business management software developed by a Belgian company Odoo SA. (Odoo SA, n.d.)

2.1 History

Odoo has not always gone by the aforementioned name. It started out as TinyERP in 2005. TinyERP was a software product created by an ambitious Belgian entrepreneur who had a goal of overtaking SAP in the ERP business. Three years later, it was renamed to OpenERP to avoid a misconception of the software being tiny. The name OpenERP most likely had roots in the open source nature of the software but this has not been confirmed by the creator. OpenERP continued to grow over the following years as a company and as a software suite. (Pinckaers, 2013)

As a company, OpenERP saw a steady 100% growth a year and was awarded the title of the eighth fastest growing company in Belgium by Deloitte in 2013 because it saw a growth in turnover of 1549% in the preceding five years (Odoo SA, 2013). As a software product, OpenERP continued to grow and improve its technology. It became the most installed management software in the world with 1,000 installations per day and up to 60 new modules released every month (Pinckaers, 2013).

OpenERP kept growing and expanding its software beyond the scope of traditional ERP systems by implementing features such as CMS, eCommerce, Point of Sale, an integrated Business Intelligence engine, and much more in their newly released version 8. At this point, OpenERP was much more than a tiny or a simple ERP system so the name of the company and software was changed to Odoo to distance itself from the term “ERP”. (Pinckaers, 2013)

Today, Odoo is at a version 14 with a total of 7 million users worldwide. Odoo as a company employs over 1,700 people. The Odoo community has contributed over 16,000 modules that cover a wide variety of business needs. (Odoo SA, n.d.)

2.2 Open source

The definition of open source is not as clear as it may seem on the surface. It means more than access to source code and there are multiple competing views on what constitutes open source. Sometimes the term “open source” is paired with the term “free and open source” refering to the meaning of Free software by GNU and the Free Software Foundation. “Free software” means software that respects users' freedom and community. Free Software Foundation (2021) summarises the meaning: “ - - - Roughly, it means that the users have the freedom to run, copy, distribute, study, change and improve the software.”

Open source and free software do not hold the same meaning according to Free Software Foundation (2021). Free software holds an ideological and philosophical weight to it and is not merely about the access to source code but also about the freedoms and rights of the users of the software. In practice, the difference between free software and open source software is that open source software generally has a looser criteria and does not always include copyleft. (Free Software Foundation, 2021)

Copyleft means that a software which is licensed under a free copyleft license can be modified and extended as long as the new version of the software is released under a free copyleft license as well. (Free Software Foundation, 2018)

Odoo is often marketed and referred to as open source software but it fits under the idea of free software (Odoo SA, 2015) as it is currently licensed under the GNU Lesser General Public License (LGPLv3).

The creator and founder of Odoo had a dream of leading the enterprise management market with a fully open source software and, from the beginning, Odoo has provided its core software suite as free and open source (Pinckaers, 2013). Up until version 8, Odoo was released with a GNU Affero General Public License (AGPLv3) (Odoo SA, 2014). Starting from version 9, Odoo transitioned into an open core model, which provides a premium subscription based proprietary version in addition to an open source version (Odoo SA, 2015). The open source version was released under the GNU Lesser General Public License (LGPLv3) with this change (Odoo SA, 2015).

In 2013 the Odoo Community Association, OCA was created as a nonprofit organization (The Odoo Community Association, n.d.). The mission statement of the Odoo Community Association (n.d.) in their words is the following: " - - - Establish and support an Open Source and collaborative community for the development and promotion of the Odoo features and modules."

2.3 Functionality

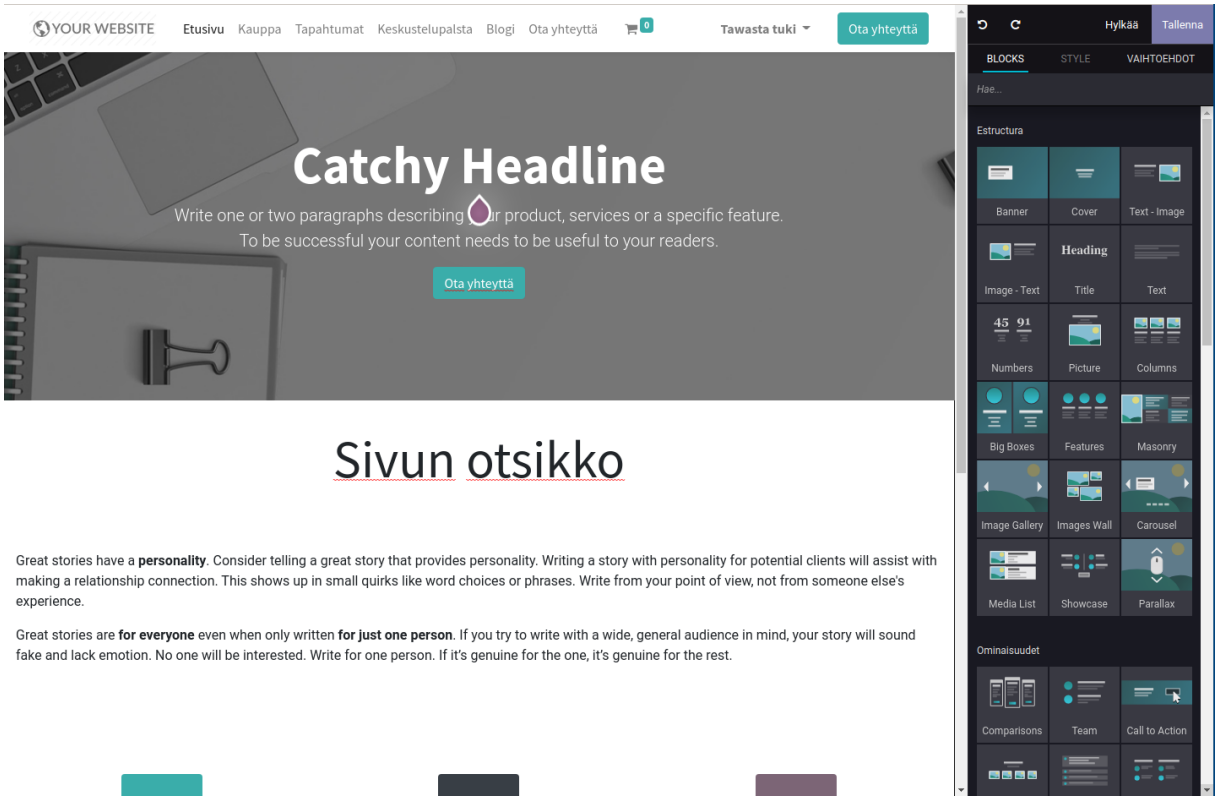
Odoo is a web application that runs on a server and is accessed using a web browser. This allows access to an installation of Odoo from anywhere with an internet connection and on any device that supports a modern web browser.

The main purpose of Odoo is to be an Enterprise Resource Planning system (ERP). ERP systems are intended to manage business processes and track information on them (Oracle, n.d.). They generally include processes such as sales, purchases, stock, accounting, and project management. A common problem with companies as small as startups up to large enterprises is managing and keeping track of data linked to their business processes. What ERP systems aim for is centralization of this information in a single database or location for ease of reach (Oracle, n.d.).

Odoo can also provide Customer relationship management (CRM), which means the administration of interactions between a business or an organization and their customers. This data is recorded to analyze and improve customer relationships leading to a growth in sales and customer retention. (Christensson, 2013)

In addition to providing ERP and CRM systems, Odoo also has a Content Management System or CMS (Figure 2). CMS is a tool that allows the user to create, edit, and publish content on the Web. Usually CMS software tries to have an intuitive user interface to allow modifying and building webpages. WordPress is one of the most popular CMS softwares with over 40% of all websites being made using it (Q-Success, 2021). While Odoo may not directly compete against WordPress in CMS market share, it allows businesses and organizations to have presentable websites while staying inside the same system as their other business processes.

Figure 2. View of Odoo CMS.



2.4 Modularity and customizability

In its core Odoo provides an application server and a framework to build business applications. Each application is a module that implements or extends a specific feature. The core Odoo installation features 30 official modules created and maintained by Odoo SA and the open source community. There are also countless of additional modules built by the community and other third parties. Some are free and some are available for purchase.

With these modules, each Odoo system can be tailored to fit a specific business need while leaving flexibility to add more features down the line by installing more modules. Perhaps at first a startup wants a simple website using the Odoo CMS, but later the system can be expanded to handle eCommerce and event registrations as well.

Odoo modules are also split into two categories: applications and components. Applications are modules which may include multiple smaller components and provide a standalone

application with many features. Event module is an example of an application as it provides features to create events, manage registrations, send emails to registrants and more. Component modules extend an existing application by adding some features. It could be as simple as a module that adds a shortcut button.

The core Odoo applications often do not fully fit the workflow of a business and the applications and features need to be customized for the client. This customization of Odoo is done by the aforementioned component modules to add features, shortcuts, or to automate processes. Customizing Odoo to fit exact business needs will reduce the time and effort spent on using the system. This will support and streamline the processes as opposed to slowing down the workflow by having to fight against an incomplete system.

Some businesses might already use other systems they prefer to keep but also want to be able to transfer data and processes between the old systems and Odoo. This kind of integration of multiple systems is possible to customize for Odoo with a built-in XML-RPC API (Application Programming Interface) or a custom REST API.

2.5 Technical features and development guide

The backend of Odoo server and modules are mostly written in Python with JavaScript and XML to render the frontend.

2.5.1 Architecture

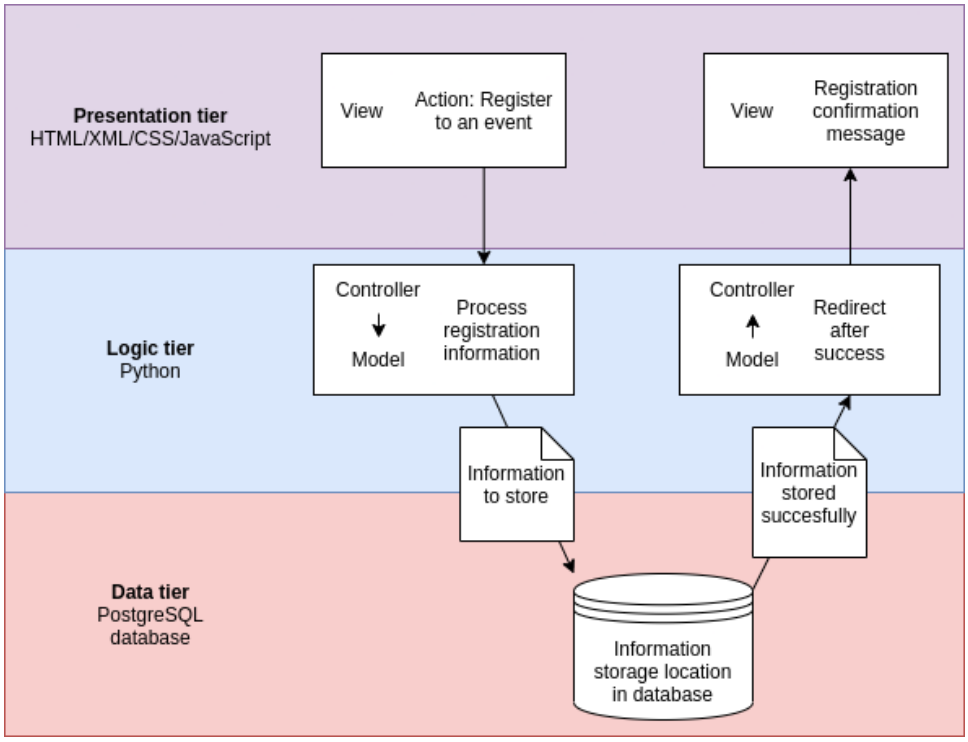
Odoo uses a three-tier architecture (Figure 3), which is commonly used by web applications. In a three-tier architecture, the visual presentation, business logic, and data storage are separated. The presentation tier is the part of the application that is visible to the end user. Its purpose is to show the results and tasks to something which the user can understand. Odoo's presentation tier is written in XML, JavaScript and Sass/CSS. (Odoo SA, n.d.)

The next layer in the application is the business logic tier. This layer is where all of the logical decisions, calculations, evaluations and commands are handled. It is also responsible for

transfering data between the presentation and data layers. The logic tier for Odoo is written in Python. (Odoo SA, n.d.)

The bottom layer of this three-tier architecture is the data tier, where all of the information is stored and retrieved from. The data layer can be a database or a file system and in Odoo's case this layer uses PostgreSQL database. (Odoo SA, n.d.)

Figure 3. Odoo's three-tier architecture.



2.5.2 Creating an Odoo module

Odoo modules contain the following elements (Odoo SA, n.d.):

- Business objects or models, which are Python classes that contain the program logic and fields that map to the database using an ORM layer.

- Object views, which create the UI. These are XML files which are rendered into HTML pages using the QWeb templating engine.

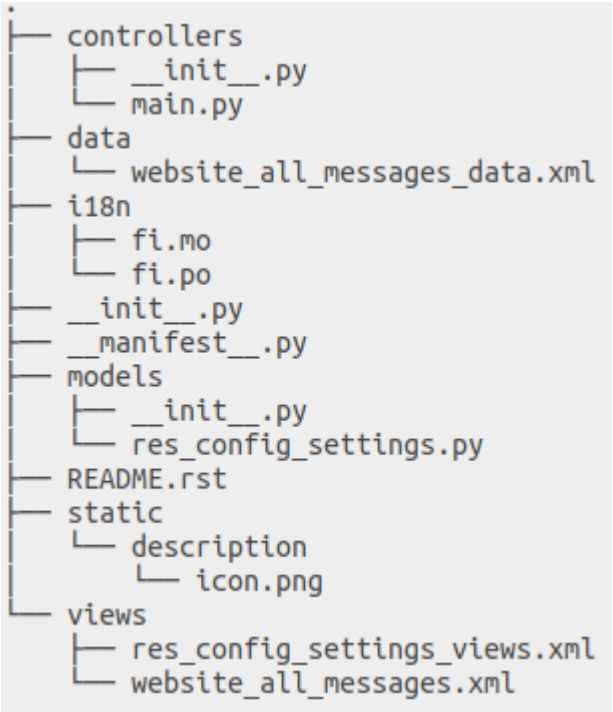
- Data files, which declare the model data. These data files include XML and CSV files that create the backend UI, add security groups and rules, define system parameters, add demonstration data to the system and add localization of different languages.

- Web controllers, which handle browser requests and redirections.

- Static web data, such as, Images, CSS, Sass or JavaScript files.

A single module does not have to use all of these elements and none of them are mandatory by themselves. To create a module, a new directory needs to be created under a module directory (Figure 4). This directory name will be the technical name of the module.

Figure 4. Odoo module directory structure.



The most important file of a module is the __manifest__.py file, which declares the module and defines the module name, dependencies, included files, and other information (Figure 5). Along with a manifest file, Odoo also requires an __init__.py file to tell Python which files to load. If a module has no Python files this init file can be empty but it still needs to be included.

Figure 5. Example of a __manifest__.py file.

```
{
    "name": "Website Event Waiting List",
    "summary": "Adds a waiting list functionality to Events.",
    "version": "14.0.1.2.1",
    "category": "Events",
    "website": "https://gitlab.com/tawasta/odoo/event",
    "author": "Tawasta",
    "license": "AGPL-3",
    "application": False,
    "installable": True,
    "depends": ["website_event_cancellation"],
    "data": [
        "wizard/waiting_mail_list_wizard.xml",
        "security/ir.model.access.csv",
        "data/email_template_views.xml",
        "data/email_template_data.xml",
        "views/assets.xml",
        "views/event_views.xml",
        "views/event_ticket_views.xml",
        "views/event_templates_page_registration.xml",
        "views/event_templates_page_waiting_list.xml",
        "wizard/waiting_mail_list_message.xml",
    ],
}
```

2.5.3 Models

One of the key components of Odoo is the ORM (Object Relational Mapping) API, which connects the logic tier with the data tier. It features a hierarchical structure, constraints consistency, and validation, optimized processing by query, permission optimisation, persistent objects with PostgreSQL database, multi-level caching system, two different inheritance mechanisms, and a variety of field types such as classic types integer, char, boolean and relational as well as functional field types. (Odoo SA, n.d.)

This ORM API saves the developer from having to write most of the SQL queries themselves which provides extensibility and security. Models are created as Python classes which extend the Odoo model (Figure 6). This integrates them into the ORM layer. (Odoo SA, n.d.)

Model fields define what the model can store and where, and these fields are defined as attributes inside the model class. Fields can be divided into two broad categories of simple

and relational fields. Simple fields are stored directly in the models table and relational fields link records. Some examples of simple fields are date, text, and boolean fields. Relational fields can be Many2one fields which link a single or no record from another field, One2many fields which link all the records in a single field, or Many2many fields which can link multiple records from another field. (Odoo SA, n.d.)

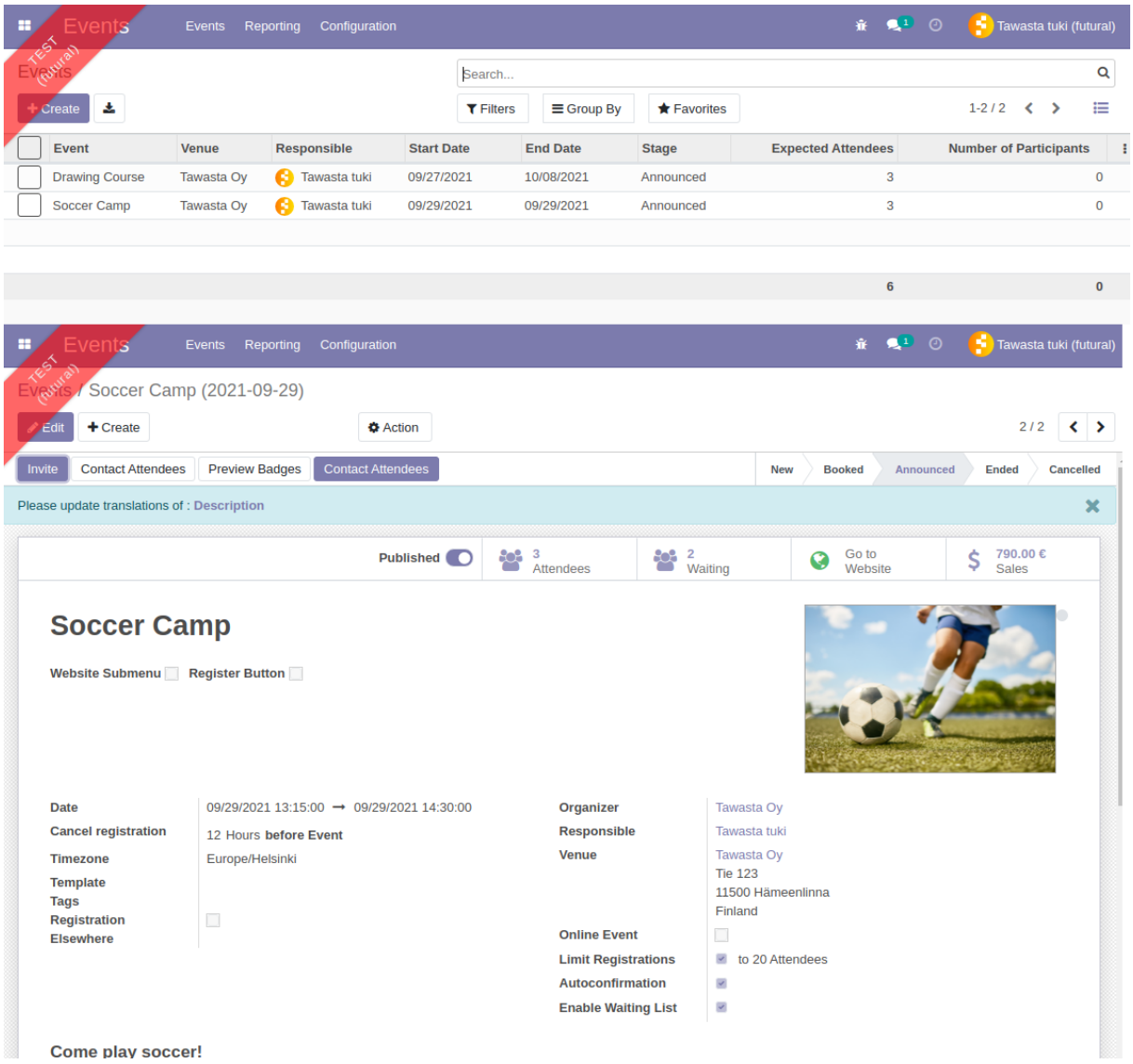
Figure 6. Example of a model inheritance.

```
# 1. Standard library imports:
# 2. Known third party imports:
# 3. Odoo imports (openerp):
from odoo import fields
from odoo import models

# 4. Imports from Odoo modules:

# 5. Local imports in the relative form:

# 6. Unknown third party imports:


class Survey(models.Model):
    # 1. Private attributes
    _inherit = "survey.survey"

    # 2. Fields declaration
    is_event_survey = fields.Boolean(
        "Is an Event Survey",
        help="If checked, this survey can be used as an Event Survey.",
    )

    # 3. Default methods

    # 4. Compute and search fields, in the same order that fields declaration

    # 5. Constraints and onchanges

    # 6. CRUD methods

    # 7. Action methods

    # 8. Business methods
```

2.5.4 Views and data files

Views define how records are displayed in the back-end Odoo system and are a part of the presentation tier. These views are written in XML and are highly customizable. There are multiple types of views such as form, list, and kanban. (Odoo SA, n.d.) List and kanban views generally display an overview of multiple records and a form view displays details on a single record (Figure 7).

Figure 7. Example of a list view (top) and form view (bottom).



Sometimes incorrectly referred to as views, QWeb Templates handle the display of web pages by rendering XML into HTML pages. The QWeb templating engine is able to include

conditional statements, loops, and attributes in web pages. It can also output Odoo record data in different formats. QWeb templates also feature inheritance making it trivial to extend and re-use previous snippets (Figure 8). (Odoo SA, n.d.)

Figure 8. QWeb Template code example with inheritance.

```
95  <!-- Confirmation page add waiting list -->
 1  <template id="registration_complete" inherit_id="website_event.registration_complete">
 2      <xpath expr="//div[@class='col-12']" position="replace">
 3          <div class="col-12">
 4              <t t-if="attendees[0].state == 'wait'">
 5                  <h3>Joined waiting list!</h3>
 6              </t>
 7              <t t-else="">
 8                  <h3>Registration confirmed!</h3>
 9              </t>
10              <span class="h4 text-muted" t-esc="event.name"/>
11              <t t-if="attendees[0].state == 'wait'">
12                  <p class="py-2">You will be contacted if the event has more seats available.</p>
13              </t>
14          </div>
15      </xpath>
16  </template>
```

The main way Odoo handles the layout and styling of web pages is with the help of Bootstrap. Bootstrap is a free and open source CSS framework which allows for responsive web development (Bootstrap team, n.d.). These Bootstrap classes are defined in the QWeb XML code. To further customize the layout of web pages, Odoo also supports plain CSS files or Sass files.

Data files in Odoo play a big role in the value of a module. They are loaded during the installation of a module and define things like access rights, reports, and email templates. They can also add plain data to fill a module with demo data or sensible default values. Data files are mainly defined through XML data files, but access rights are defined in a csv file. (Odoo SA, n.d.)

2.5.5 Web Controllers

Odoo web controllers route and direct users to URLs in a website and handle web requests and responses. The controllers are part of Odoo's logic tier and they connect actions made by the end user in the frontend to an action or calculation in the logic tier. Odoo web controllers are created as Python classes which inherit the core controller. These controllers

can define which QWeb templates to render and provide the context on the pages. (Odoo SA, n.d.)

Controllers can call model functions on the selected records and initiate a process such as registration to an event after a user completes a registration form on the website.

2.5.6 Javascript

Odoo also has a small Javascript framework allowing it to interact with models and records in the database. The three main uses for the Javascript framework are in the backend web client, in the website, and in the point of sale. JavaScript in Odoo is used in the presentation tier, but it can also call actions from Python models or controllers.

The Javascript files are loaded from XML data files where they are defined. The Odoo Javascript framework is used to load the Javascript files in the correct order. In some cases asynchronous processing is required which Odoo handles by returning promises. (Odoo SA, n.d.)

Odoo's Javascript framework also provides useful widgets that help with the rendering of data and provide useful interactive blocks such as a datetime picker. (Odoo SA, n.d.)

3 Development and implementation

The following chapter examines the practical implementation of the thesis where the core Odoo event management module is extended and improved upon according to client specifications. It goes over the tools and technologies used in the implementation of the custom modules as well as a step by step walkthrough of the development process starting from the planning phase.

3.1 Technologies and tools

The core logic of Odoo is mainly written in Python with JavaScript supplementing the frontend. The frontend is rendered from XML files using a QWeb templating engine.

3.1.1 Vim

My text editor of choice for writing the code was an open source text editor, Vim. Vim stands for “Vi IMproved” and is a successor of vi. Vi is one of the oldest text editors, developed around 1976 by Bill Joy (Ritter, 2007). Vim was initially released in 1991 by Bram Molenaar and is still actively maintained by him and the open source community to this day (Brabandt, Vim FAQ, 2019).

Vim is purely a text user interface (TUI) program and is run on the command line. What makes Vim special is its unique way of editing text in different modes. Vim has seven (Brabandt, Vim FAQ Vim Modes, 2019). The most important modes are normal mode, visual mode, insert mode and command mode. In normal mode, the user can enter editor commands such as copy, paste, delete and move the cursor location. In visual mode, text can be highlighted and then normal mode commands can be run on the selected text. Insert mode is similar to editing text in most other editors as typing text inserts it into the file and using the backspace removes text. In command line mode, the user can run Vim commands that do a specific action. For example typing ”wq" in command line mode saves (writes) and exits (quits) the file. These modes make editing text in Vim very different from other editors so Vim beginners often go through a learning curve. After learning the ins and outs of Vim, it can be argued to be one of the fastest ways of editing text.

Vim is also highly customizable and extensible and can be tailored to every users specific needs. For example, I have spent countless hours customizing and modifying Vim to suit my needs for Odoo programming and other text editing needs. Currently I am using 17 additional plugins developed by the community and added some of my own custom features such as snippets to streamline the process of writing code.

3.1.2 Git and GitLab

Git is a free and open source version control system originally developed by Linus Torvalds in 2005 (Chacon & Straub, 2021). Git helps developers collaborate on projects with other developers and it is also useful for keeping backups of older versions of projects. A good git workflow is essential for any software developer. With git, it is possible to divert new branches of the original project to work on additional features or to refactor old features. These changes can later be verified by other developers to avoid possible bugs and to make sure the project works as intended after the changes. Once the changes are acceptable, the branch can be merged with the original project.

Git is often paired with a centralized platform to host the code on and a graphical interface to make some of the workflow more intuitive. At Tawasta, we used GitLab to host the source code of the customized Odoo modules. GitLab can run automated tests and pipelines to check for errors in the source code before merging them to the repository of the production version.

3.1.3 Linters and fixers

Linter is a code analysis tool used to flag programming errors and stylistic errors in code (Wilson, n.d.). In my customized version of Vim, I have integrated some of these linters to automatically check and fix errors in the background while editing.

In Odoo development, we at Tawasta followed the OCA Guidelines closely as they provide a list of suggested linters and tools to use along with configuration files to make them behave in a desired way (Odoo Community Association, 2018). Following the OCA guidelines ensures that every developer who works on the project has a similar programming style making it easier to collaborate and compare changes made by other developers.

For Python, we used the linters Flake8, Pylint, and Black to check for errors and automatically format the style of the code. ESlint and Prettier was used for Javascript. For XML files, we used Prettier.

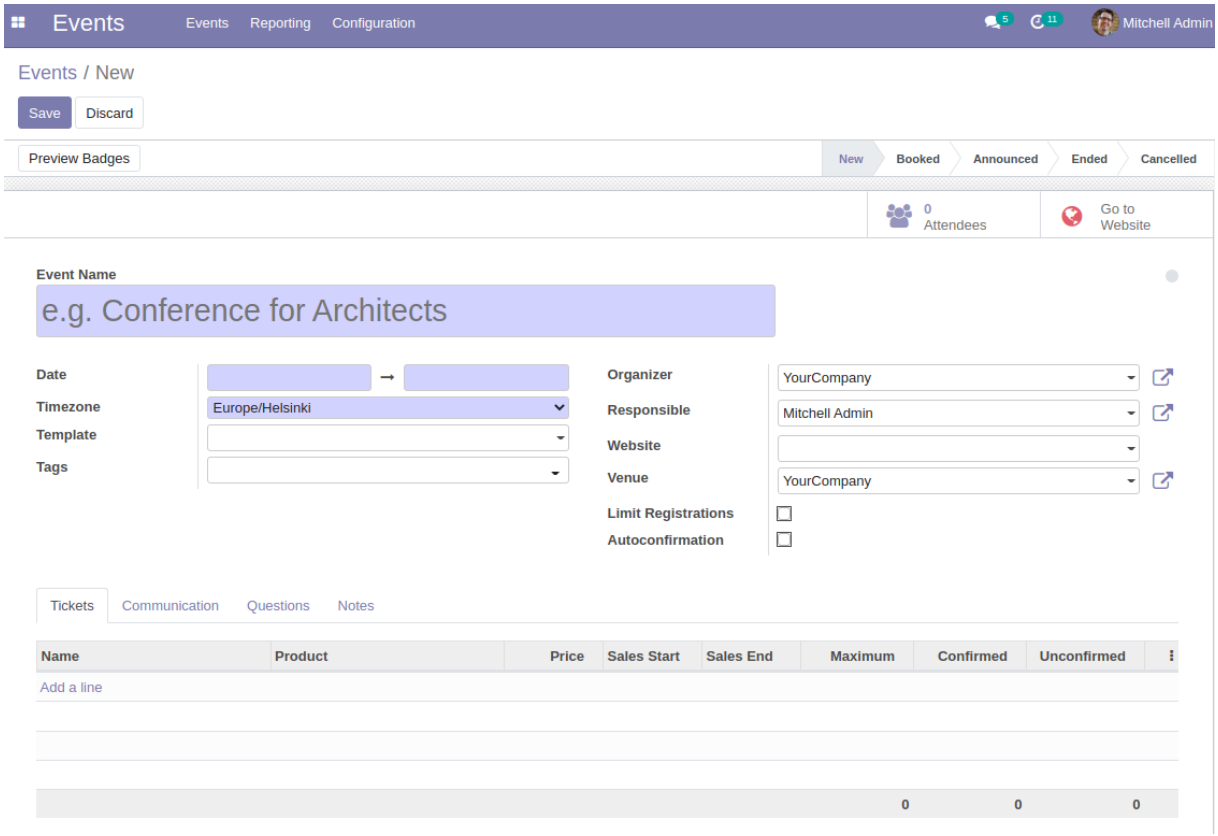
3.2 Scope and planning phase

A need for event management came from one of Tawasta's clients. They had previously used a Joomla Content Management System to manage events and registrations. However, the Joomla version used was no longer supported and they had to find a replacement. It turned out the core Odoo event management module supported most of the features the old Joomla installation did except for a waiting list feature. Around the same time, a new client for Tawasta emerged needing a way to manage events, training sessions and certifications along with other ERP related features. This was a great opportunity for me to tackle as the scope of the project was mostly limited to one area of the system, event management.

Event management module is one of the 30 core Odoo modules developed by Odoo SA (Odoo SA, n.d.). At the time of the planning phase OCA had not yet made any of their own additions to the event module for version 14, so the development could be started from a clean slate. Because we started from a clean slate it was even more important that any modifications made would not break the core event management logic and be extensible and easy to add more features to.

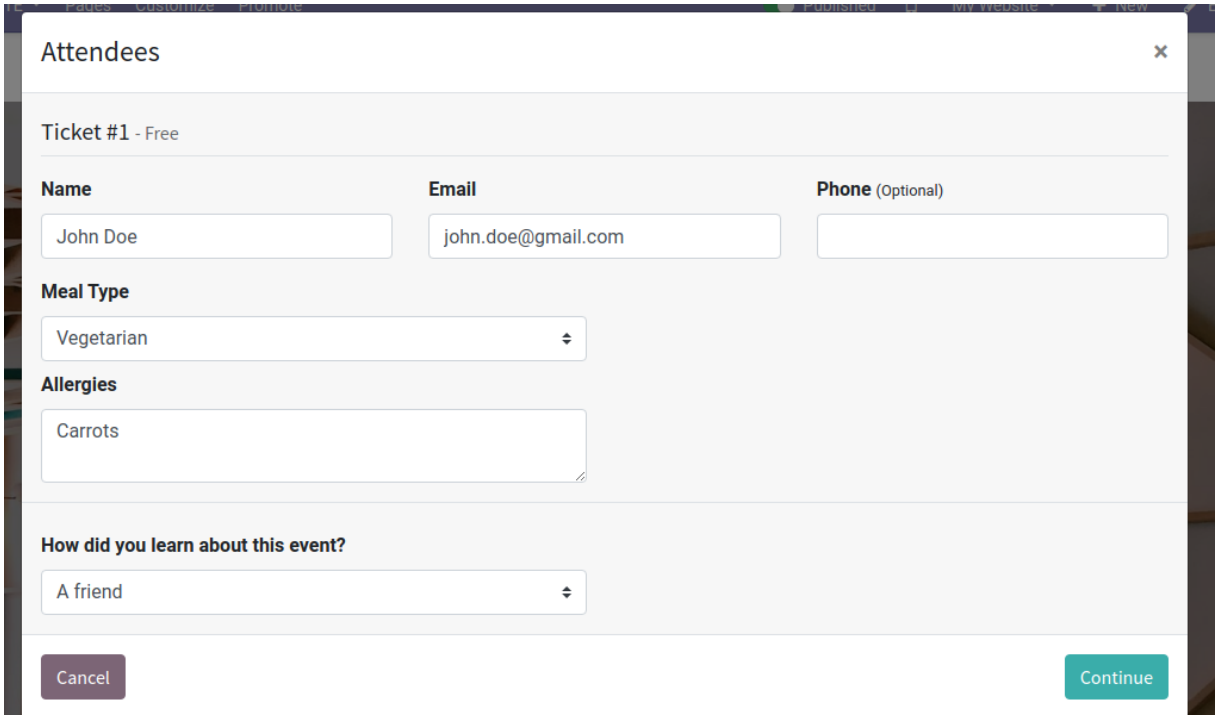
The core event management application had the ability to create events with basic information such as time, place, description, and organizer (Figure 9). The event organizer could also limit the maximum amount of attendees for an event. The application also had the ability to send automated emails after each registration, at a certain time before the event, and after the event had passed. The event oprganizer could also sell tickets for free or with a set cost using the eCommerce module.

Figure 9. Odoo core event creation view.



During registration, the registrants are asked for their name and email as well as an optional phone number. Additional optional questions may be added for the registrants. These questions could be either free text inputs or drop-down boxes where the registrant chooses their answer. (Figure 10)

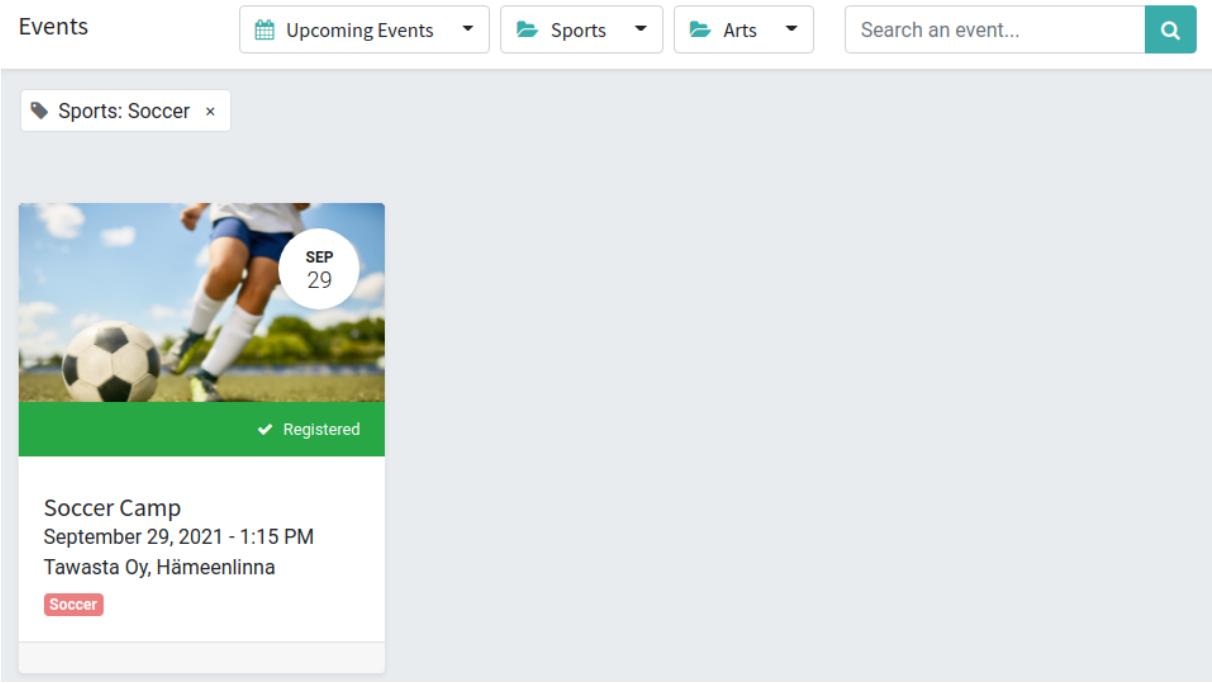
Figure 10. Default registration form with additional questions.



To manage these registrations, the organizer could adjust the state of the registrants whether they were unconfirmed, confirmed, attended or cancelled. The organizer could also send emails to these registrants.

In the frontend part of the web page the organizer could add tags for the events to help categorize them and the user could search events using these tags (Figure 11). Each event could have their own banner image and a custom description on the website.

Figure 11. Event tag search on website.



During the planning phase, we immediately knew we had to implement a waiting list since both of Tawasta's clients needed it. We had to find a solution which made it clear for the users trying to register whether they were registering to a waiting list or not. We also had to plan how the waiting list registrations are managed in the system, what emails they are going to receive from the system, how they are seperated from the confirmed registrations, and how the process of confirming someones registration from the waiting list works. From this, we also realized that the core event module does not support cancelling registrations by the registrants themselves since only the event managers could cancel registrations. This might work well for a small organization with a handful of events where the organizers would be contacted for any possible cancellations. It would be more convenient, however, if the registrants themselves could cancel their attendance without the need to contact the organizers every time.

The second big challenge, perhaps the biggest challenge, was finding a way to ask questions from the registrants during registration. This was possible with the core event module, but it was very limited. It allowed additional questions such as free text boxes and drop-down menus for each event, but these questions would only be tied to a specific event and would

need to be added seperately for each event. This poses major challenges for analysing the data from these questions because each question for each event would be a different recordset. For example, the system could have a dozen questions which are similar, but they would each create a different record. Thus to analyze the overall data the questions would have to be manually combined in an Excel spreadsheet or similar. Not only that, but the core module was very limited when it came to the questions. There were only two options: free text and drop-down menu and the questions were only optional to answer by the registrant.

With these issues at hand we decided to integrate another core Odoo module, surveys, into events. The core survey module allowed a creation of surveys with multiple different questions and options. The user could make questions with free text, datetime, multiple choice, matrix type questions and define what questions are required to answer and what are optional. The intended purpose of the survey module is to send customers emails with a survey they can fill out, for example, to gather customer feedback. The integration of the survey module into events had multiple challenges which are going to be covered in detail in the following sections.

3.3 Development process

The development process could be divided into three main parts which were developed in order. First, we created the waiting list logic and associated features such as automatic emails to waiting list and management of the registration state. We also implemented the logic for paid events and the waiting list. Second, we integrated the survey module into events. Third, we cleaned out any bugs that came up as well as added some additional nice-to-have features.

3.3.1 Waiting list

Before we created the waiting list, we created an ability to cancel registrations from an unique cancellation link which was delivered to the registrants along the registration confirmation email. This unique cancellation link was created using an UUID to make sure

nobody could guess someone elses cancellation link (Figure 12). The link would direct the user to a web page with a button to confirm the cancellation.

Figure 12. UUID access token calculation and unique link computation.

```
    # 2. Fields declaration
    manage_url = fields.Char("Public link", compute="_compute_manage_url")
    access_token = fields.Char(
        "Security Token", store=True, compute="_compute_access_token"
    )

    # 3. Default methods

    # 4. Compute and search fields, in the same order that fields declaration
    @api.depends("event_id", "access_token")
    def _compute_manage_url(self):
        """ Url to cancel registration """
        base_url = self.env["ir.config_parameter"].sudo().get_param("web.base.url")
        for registration in self:
            registration.manage_url = urls.url_join(
                base_url,
                "/event/%s/registration/manage/%s"
                % (registration.event_id.id, registration.access_token),
            )

    def _compute_access_token(self):
        for registration in self:
            registration.access_token = str(uuid.uuid4())

```

The web page (Figure 13) along with the http routing from the link (Figure 14) had to be implemented from scratch. Implementing a new web page on Odoo is usually not a challenge because the QWeb templating engine makes it possible to inherit the base web page layout and only add the additional content inside.

Figure 13. Custom cancellation page redirected from email link.

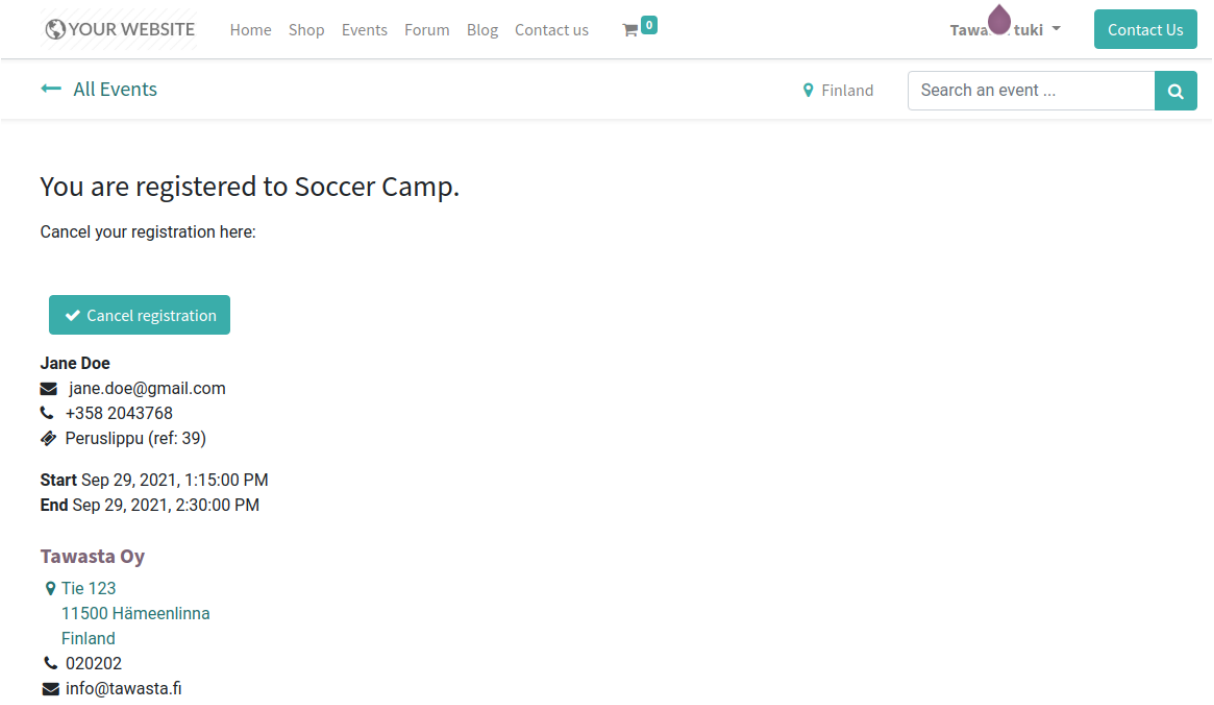
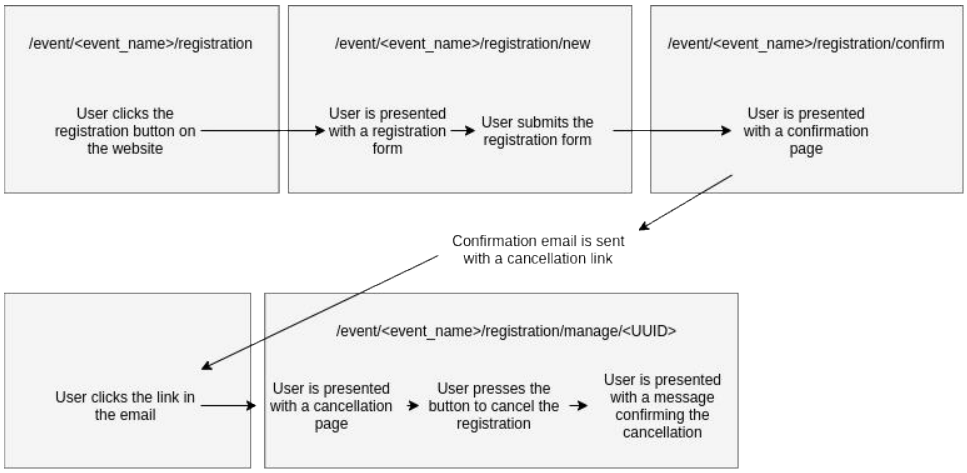


Figure 14. HTTP routing for event registration process.

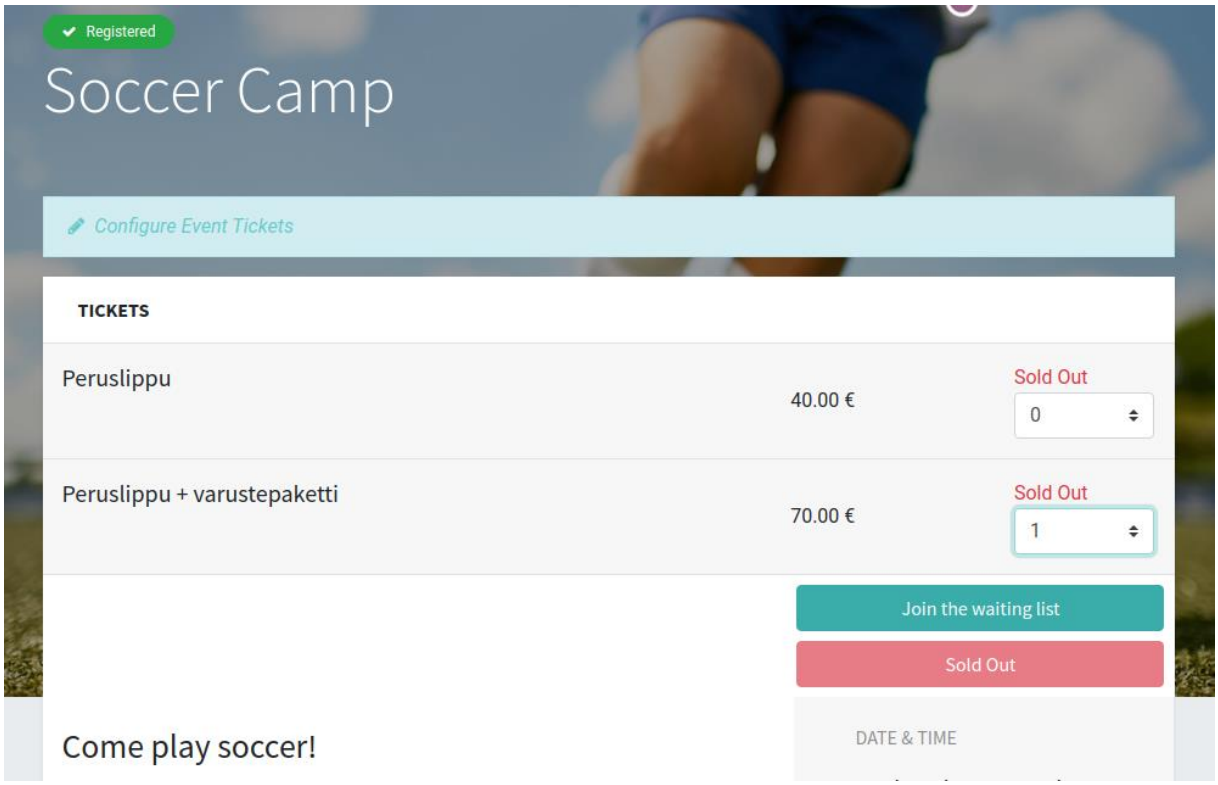


From this change, we also wanted to add an ability to limit the cancellation of registrations to a set time period befor an event. The organizer could define this time period themselves during the creation of the event.

To start the waiting list logic, we first created a boolean toggle for Events which would enable or disable the waiting list feature. The system would throw a validation error by default if the registrations were limited and the cap was reached and we tried to force another registration. The website would also show users that the event was full and the access to the registration form got disabled. To work around this we changed the backend logic so each registration that happened after the limit was reached, and if the waiting list toggle was enabled, would become a registration in the system with a new state called waiting (Appendix 2).

To display the user on the website when they were joining the waiting list we added another button called "Join the waiting list" which would show up after the registrations were full (Figure 15). We also wanted to inform the registrant of their spot in the waiting list with an automated email after they joined the waiting list.

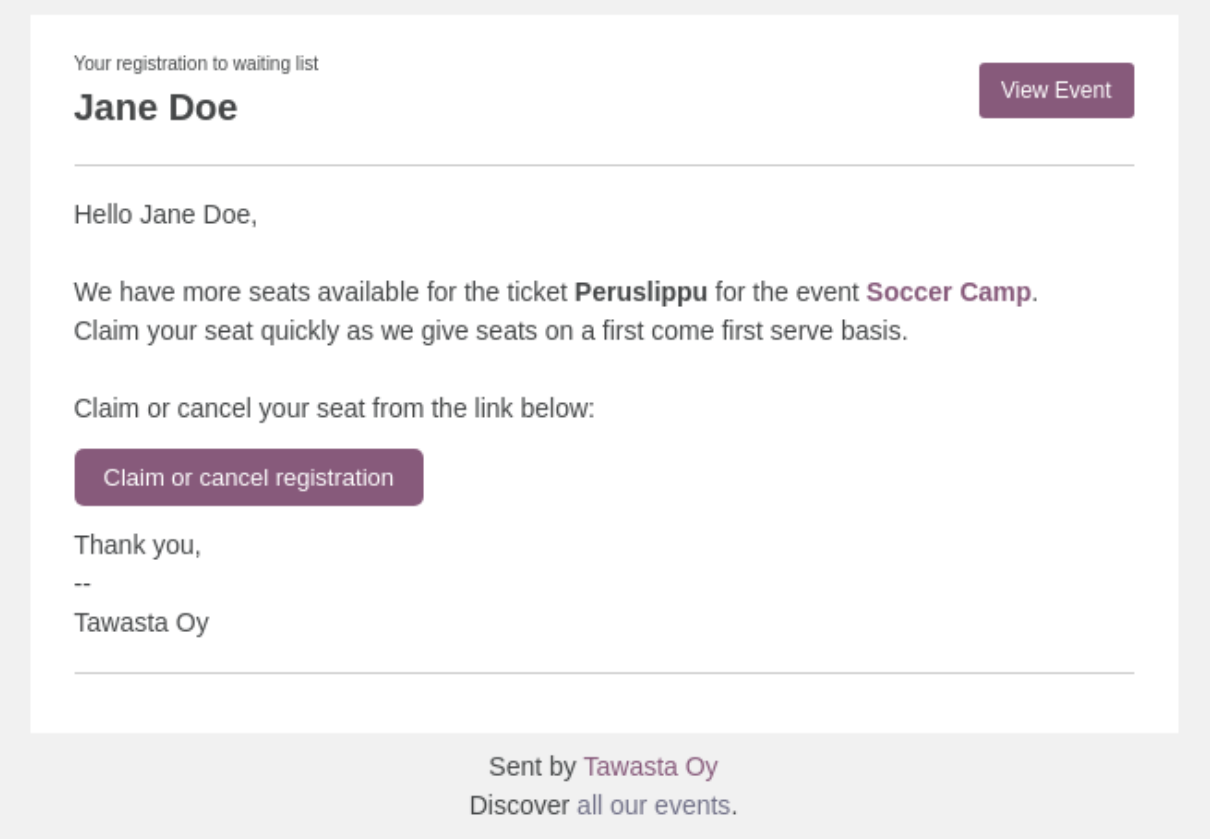
Figure 15. View of an sold out event with a waiting list button.



To confirm the registration from the waiting list, we created another automated email which would be sent to the waiting list members after the event went from fully booked to having

open seats (Figure 16). This automated email would contain a unique registration link that directs the user to the same page as above, but if the user was in the waiting list and there were open seats available, they could also confirm their registration here. Only the fastest members of the waiting list could get this open seat and the rest would stay waiting for the next available seat. If a free seat opened up, a new email would be sent to the waiting list. To avoid spam, a new email is only sent when the event goes from being full to having an open seat. Any subsequent cancellation would not trigger the automated email if the original open seat had not yet been taken.

Figure 16. Automated email to waiting list once event has open seats.



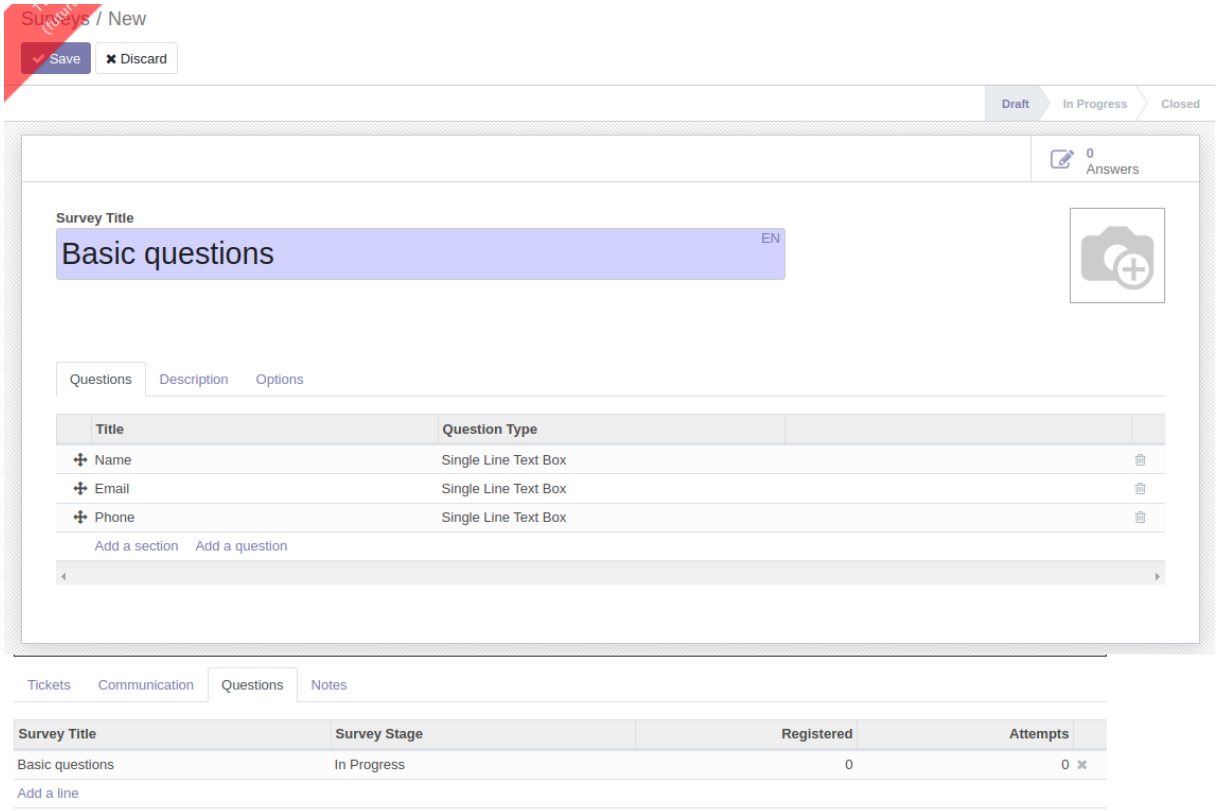
For paid events, the payment would normally happen during the registration process. We had to change this so that joining the waiting list would not require a payment but confirming the registration from the waiting list would redirect to a payment process.

An event could have multiple different tickets of which some tickets were sold out and others still had free seats. The registrant would join a waiting list specific to the ticket they selected in this case. If the ticket had open seats they would get notified of this by email.

3.3.2 Survey module

By default there were no connection between the survey module and the event module. We wanted the ability to link a specific survey or surveys to a single event and then during the registration process the survey or surveys would be used to gather the registration information. To create this connection between the modules, we created a many2many field for events to link one or multiple surveys to itself. We also created the necessary views to make this intuitive from the backend UI (Figure 17).

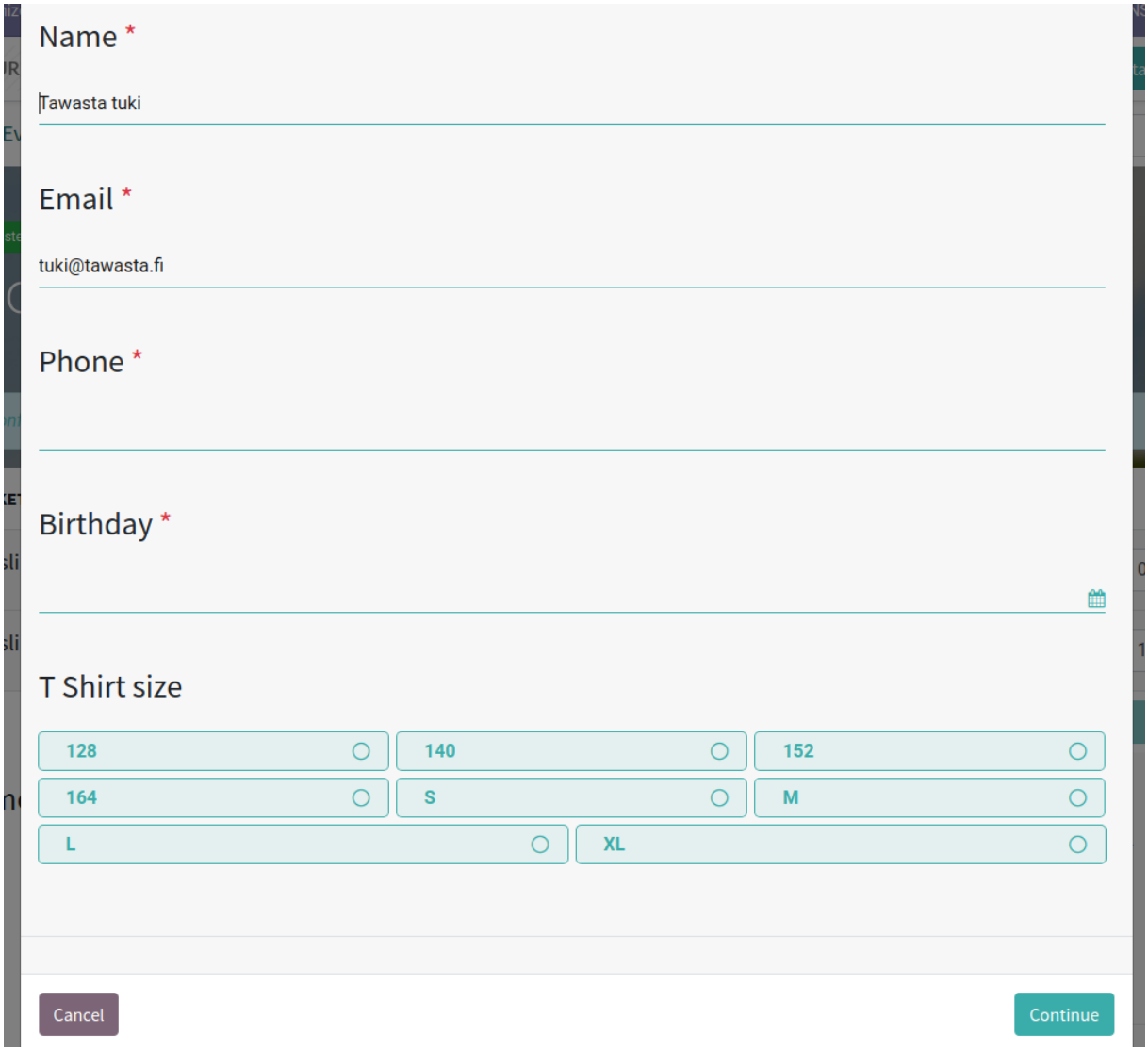
Figure 17. Survey creation view (top) and event question view (bottom).



After a survey or surveys were connected to an event, we made a major change to the event registration form so that instead of the basic form we would get all of the questions from the

surveys instead. This was a major change as we needed to implement over 700 lines of JavaScript code to make each question option work, such as a datetimepicker, multiple selection, and matrix questions. We tried to keep the layout of the form simple and responsive and we also wanted it to behave similarly to the core event registration form (Figure 18). To do this, we were able to mostly utilize Bootstrap as well as reuse the CSS/Sass rules from the survey module. We also needed to create some custom CSS/Sass rules to make the questions and answers fit nicely on a popup form (Appendix 3).

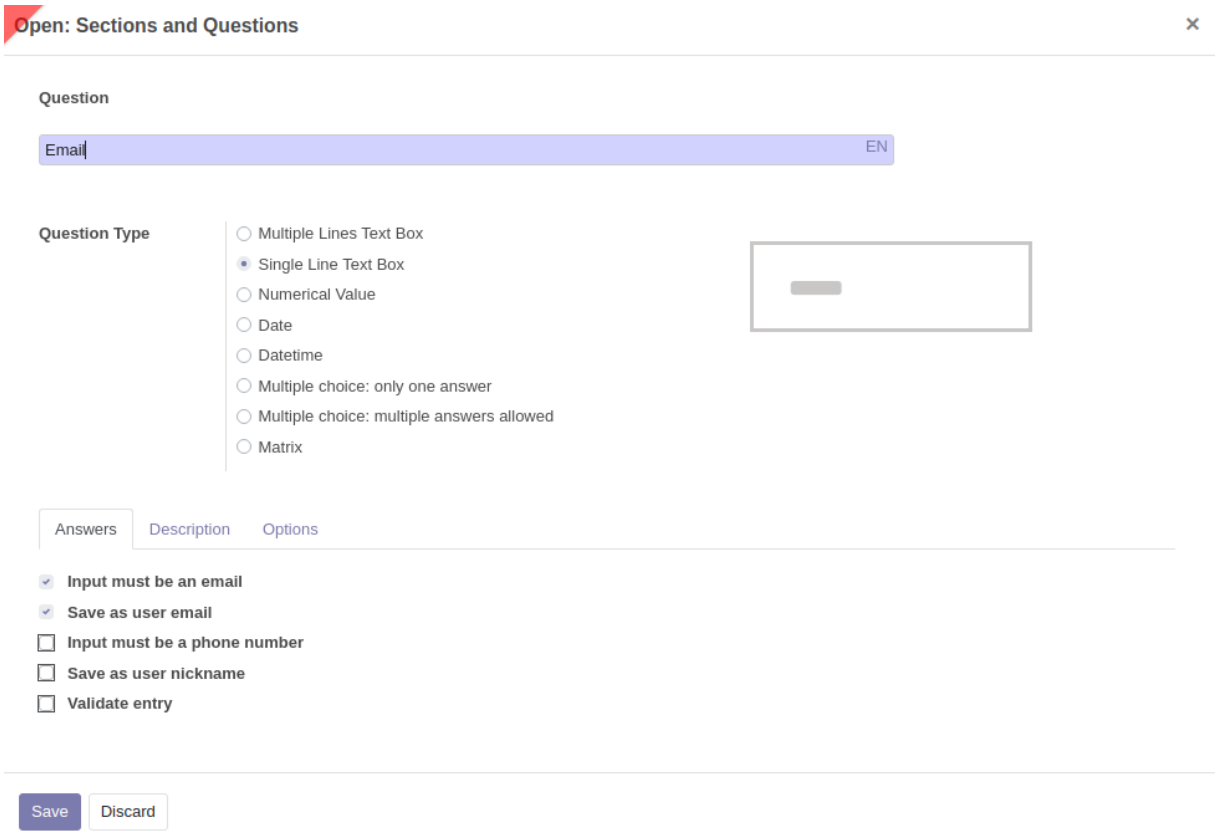
Figure 18. Event registration form with survey.



We implemented a boolean toggle on the survey which if enabled allows it to be linked to an event to know which surveys can be used with event registrations. Each survey question also

has boolean toggles to define if those questions refer to the registrants name, email or phone number (Figure 19). Therefore the correct answer is used to get the three basic pieces of information from the registrant.

Figure 19. Survey question creation view with "Save as user email" field checked.

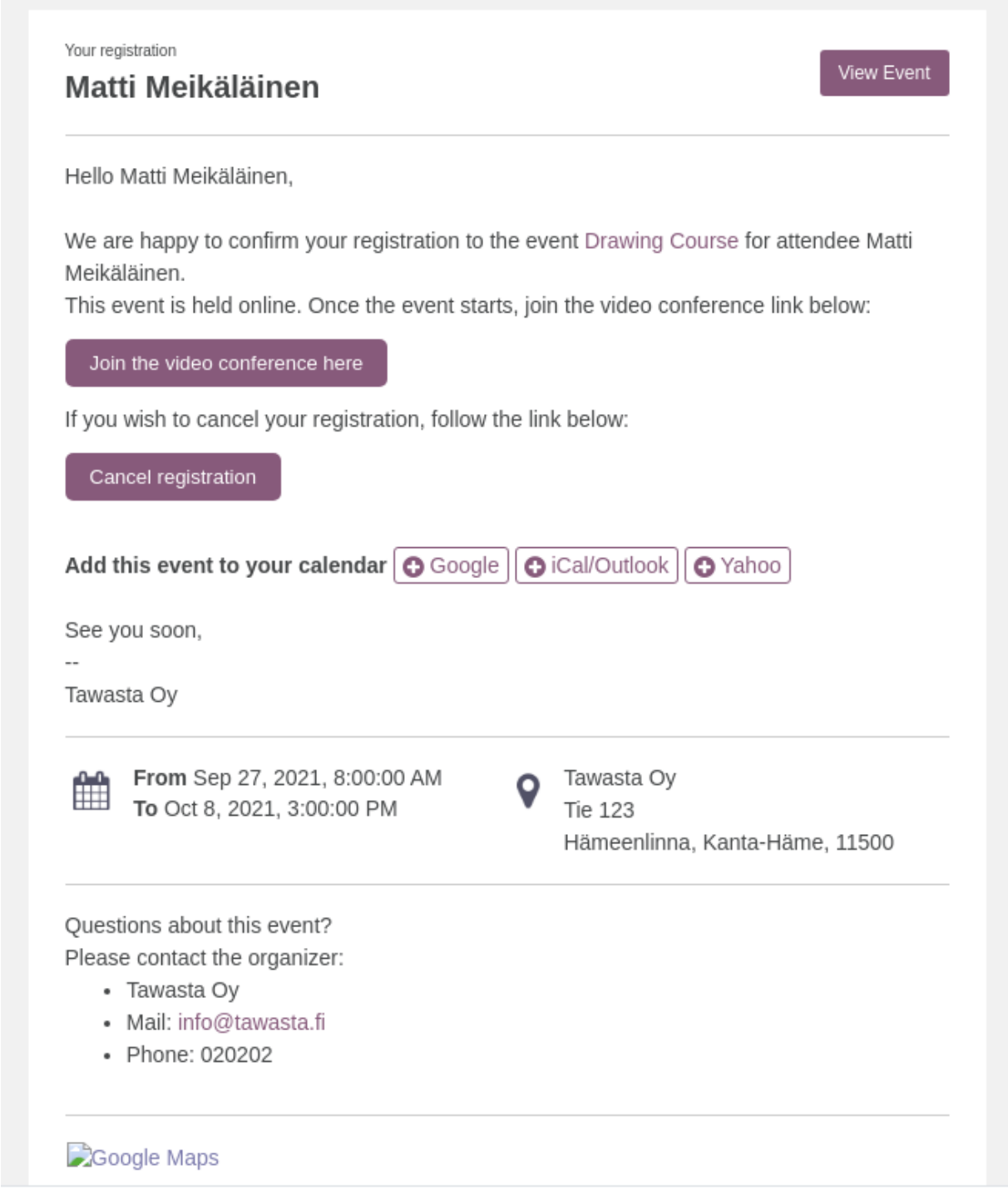


3.3.3 Finishing touches and small extensions

The code was refactored multiple times and small bugs were fixed along the way. The features also went throught internal testing at Tawasta which is how most of the bugs were found. To add some finishing touches, we used an OCA QWeb email template module to our advantage (Odoo Community Association, n.d.). This way, the automated event emails could be written as QWeb templates instead of default Jinja2 templates, which Odoo core uses. The advantage of a QWeb email template is that it is easy to customize the email templates in the future to have different wording or add new content to them.

Tawasta's clients also often host online events in a Zoom or Teams call. We wanted to add the ability to directly link the video conference link to the event and have it sent to each registrant through an automated email (Figure 20). This way the organizer could be ensured that every confirmed registrant got the video conference link.

Figure 20. Automatic registration confirmation email with a video conference link.



4 Summary

In this chapter, the final results of the project are shared and the process of creating and managing events in the improved version of event management is shown. The chapter will also go over the challenges faced and some possible future improvements.

4.1 Final results

The overall results of the thesis project were excellent as each specified goal was achieved and some of the core logic was also improved to make creating and managing events easier. At the time of finishing this thesis, the event management improvements are already in use by one of Tawasta's clients and a release for another client is coming in fall 2021. The event management module improvements are also able to be used by other future clients as the goal of modularity and maintainability were reached.

The process of creating and managing events can now be summarized as follows:

1. First, the event organizer must create a survey using the survey module. This survey must contain questions that ask for the name, email, and phone number of the registrant. Any additional questions could be asked using the same survey or with another separate survey. The event organizer can decide to split questions into different surveys since it mainly effects the reporting and data analysis of the question's answers.
2. Next, the event organizer must create an event and fill in all of the basic information, and decide whether a waiting list is used. The event organizer can create separate tickets for the event, which are either free or sold for a select price. The automated emails for the event need to be specified in the communications tab. If the event is an online event, then the event organizer can include a video conference link to automatically send it with the confirmation email. To finish creating the event, the desired survey or surveys need be selected to be used for the event registration.
3. Finally, the event organizer can publish the event on the website, share it on social media, and wait for the event to begin. If any registrants decide to cancel their

registration, the waiting list functionality will automatically let everyone in the waiting list know of the open seat. The process of cancelling a registration is done by the registrant through a link in the event confirmation email.

Additionally, the organizer can contact the registrants via email at any point, similar to the original version.

4.2 Issues and challenges

The biggest challenge of the thesis project was integrating survey module with the event module. During the planning phase this seemed rather trivial because we expected the new survey registration form to work with minimal changes to the JavaScript side of the module. However, this quickly became an issue as the form would simply not allow any inputs without a large chunk of JavaScript. Luckily we were able to use the code from the survey module as a reference but this part still took longer than planned.

Another challenge was finding a solution to handling paid events with the waiting list. This issue was mainly a logical issue as the code implementation was easy to solve after a plan was made. The challenge was coming up with a decision on when the customer has to go through with their payment. The decision we made was that the customer had to pay the registration fee after they confirmed their spot from the waiting list. Once the payment was complete the registration became fully confirmed and the seat was taken.

4.3 Future improvements

As the goal of maintanability was reached these features can be further developed to change some of the logic behind them or to add additional features to event management. One improvement could be with the logic for the paid events. Some clients may want the event fee to be paid only after the registrant has attended the event and not beforehand. The ideal solution would be to have an ability to decide the payment process seperately for each event.

Odoo allows the ability to create portal users which do not have any access to the backend of the system but only to the frontend website. These portal users are used for example with Odoo's eLearning module so that the users can complete courses and save their answers. Perhaps a future improvement could be to allow registrants to create their own portal users and to manage their registrations as a logged in user from the website. This would work best if the registrants had a good reason to create their own users to the system, maybe to complete some online courses while they wait for the events to begin.

5 References

Bootstrap team. (n.d.). *Bootstrap Homepage*. Retrieved August 29, 2021, from Bootstrap: https://getbootstrap.com/

Brabandt, C. (2019, December 12). *Vim FAQ*. https://vimhelp.org/vim_faq.txt.html

Brabandt, C. (2019, December 12). *Vim FAQ Vim Modes*. https://vimhelp.org/intro.txt.html#vim-modes

Chacon, S., & Straub, B. (2021). Pro Git. In S. Chacon, & B. Straub, *Pro Git* (2nd ed., p. 12). Apress. https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git

Christensson, P. (2013, March 28). *CMS (Content Management System) Definition*. https://techterms.com/definition/cms

Free Software Foundation. (2018, December 15). *What is Copyleft?* https://www.gnu.org/licenses/copyleft

Free Software Foundation. (2021, August 21). *What is Free Software?* https://www.gnu.org/philosophy/free-sw.html

Odoo Community Association. (2018, August 24). *OCA Guidelines*. https://github.com/OCA/odoo-community.org/blob/master/website/Contribution/CONTRIBUTING.rst

Odoo Community Association. (n.d.). *QWeb for email templates*. https://odoo-community.org/shop/product/qweb-for-email-templates-982#attr=10711

Odoo SA. (2013, October 8). *Three Awards for OpenERP in 2013*. https://accounts.odoo.com/blog/odoo-news-5/three-awards-for-openerp-in-2013-139

Odoo SA. (2014, May 15). *Odoo 8.0 License.* https://github.com/odoo/odoo/blob/8.0/LICENSE

Odoo SA. (2015, February 3). *Adapting our open source license*. https://www.odoo.com/blog/odoo-news-5/adapting-our-open-source-license-245

Odoo SA. (2015, Septempber 8). *Odoo 14.0 License*. https://github.com/odoo/odoo/blob/14.0/LICENSE

Odoo SA. (2015, September 8). *Odoo 9.0 License*. https://github.com/odoo/odoo/blob/9.0/LICENSE

Odoo SA. (n.d.). *About Us*. Retrieved August 25, 2021, from Odoo: https://accounts.odoo.com/page/about-us

Odoo SA. (n.d.). *About us: Odoo*. Retrieved August 21, 2021, from LinkedIn: https://www.linkedin.com/company/odoo

Odoo SA. (n.d.). *Odoo 14.0 Documentation: Architecture Overview*. https://www.odoo.com/documentation/14.0/developer/howtos/rdtraining/01_architecture.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: Building a Module*. https://www.odoo.com/documentation/14.0/developer/howtos/backend.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: Data Files*. https://www.odoo.com/documentation/14.0/developer/reference/addons/data.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: Javascript Reference*. https://www.odoo.com/documentation/14.0/developer/reference/javascript/javascript_reference.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: ORM API*. https://www.odoo.com/documentation/14.0/developer/reference/addons/orm.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: QWeb Templates*. https://www.odoo.com/documentation/14.0/developer/reference/javascript/qweb.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: Views*. https://www.odoo.com/documentation/14.0/developer/reference/addons/views.html

Odoo SA. (n.d.). *Odoo 14.0 Documentation: Web Controllers*. https://www.odoo.com/documentation/14.0/developer/reference/addons/http.html

Oracle. (n.d.). *What is ERP?* https://www.oracle.com/erp/what-is-erp/

Pinckaers, F. (2013, April 16). *The Odoo Story*. https://www.odoo.com/blog/odoo-news-5/the-odoo-story-56

Q-Success. (2021, August 27). *Usage statistics and market share of WordPress*. https://w3techs.com/technologies/details/cm-wordpress

Ritter, G. (2007, November 29). *The Traditional Vi*. http://ex-vi.sourceforge.net/

Tawasta Oy. (n.d.). *About us*. Retrieved August 21, 2021, from Tawasta Oy: https://tawasta.fi/yritys

Tawasta Oy. (n.d.). *Futural Homepage*. Retrieved August 21, 2021, from Futural: https://futural.fi/

The Odoo Community Association. (n.d.). *About us*. Retrieved August 26, 2021, from The Odoo Community Association (OCA): https://odoo-community.org/page/About

Wilson, D. (n.d.). *Linting tools to identify and fix code mistakes*. https://www.ibm.com/garage/method/practices/code/tool_lint/

Appendix 1: Glossary

TERM	DEFINITION
BACKEND	Refers to the part of a program in the background which is not directly accessed by the user, but generally does most of the processing and manipulation of data. In the context of Odoo backend sometimes refers to the side of the system which only administrators have access to.
BOOTSTRAP	A responsive free and open source CSS Framework.
CRM	Customer relationship management (system).
CSS	Cascading Style Sheets or CSS is a language used to define the presentation of a document written in a markup language such as HTML or XML.
ERP	Enterprise Resource Planning (system).
FRONTEND	Refers to the graphical user interface of a web page which is visible to the users of the website.
HTML	HyperText Markup Language or HTML is the markup language used to display documents in the web.
JAVASCRIPT	JavaScript or JS is a dynamically typed interpreted or just-in-time compiled programming language mainly used as a scripting language for the web.

ODOO	Open source business application suite.
ODOO MODULE	Odoo modules are server and client extensions which can either extend or add new logic to Odoo.
POSTGRESQL	PostgreSQL is an SQL compliant free and open source relational database management system.
PYTHON	Interpreted high-level programming language with a simple to read syntax.
QWEB	The primary templating engine used by Odoo to generate HTML pages from XML documents.
SASS	Sass is a scripting language used to compile or interpret into CSS.
SQL	Structured Query Language or SQL is a language used in programming for managing data in a relational database management system.
XML	eXtensible Markup Language or XML is a markup language similar to HTML designed to store and transport data.

Appendix 2: Waiting list controller code snippet

```
class WebsiteEventControllerWaiting(WebsiteEventController):
    @http.route(
        ['/event/<model("event.event"):event>/registration/new'],
        type="json",
        auth="public",
        methods=["POST"],
        website=True,
    )
    def registration_new(self, event, **post):
        """
        Registration modal and
        Waiting list modal
        """
        if not event.can_access_from_current_website():
            raise werkzeug.exceptions.NotFound()

        warning_msg = ""
        availability_check = True
        waiting_list_check = post.get("waiting_list_button")
        if waiting_list_check:
            availability_check = False

        tickets = self._process_tickets_form(event, post)
        ordered_seats = 0
        for ticket in tickets:
            ordered_seats += ticket["quantity"]
            if event.seats_limited and ticket.get("ticket"):
                # return error message if trying to register for a ticket that
is sold out
                # or trying to join a waiting list for a ticket that is not sold
out
                # and event is not sold out
                for event_ticket in ticket.get("ticket"):
                    if (
                        not availability_check
                        and waiting_list_check
                        and (
                            not event_ticket.seats_limited
                            or (
                                event_ticket.seats_limited
                                and event_ticket.seats_available > 0
                            )
                        )
                        and event.seats_available > 0
                    ):
                        warning_msg = (
                            "You tried to join a waiting list for "
                            "a ticket that has available seats"
                        )
                        waiting_list_check = False
                    elif (
                        availability_check
                        and not waiting_list_check
                        and event_ticket.seats_max
                        and event_ticket.seats_available <= 0
                    ):
                        warning_msg = "You tried to order a ticket that is sold
out"
```

```
                        availability_check = False
            if (
                event.seats_limited
                and event.seats_available < ordered_seats
                and availability_check
                and not waiting_list_check
            ):
                if not warning_msg:
                    warning_msg = "You tried to order more tickets than
available seats"
                availability_check = False
        if not tickets:
            return False
        return request.env["ir.ui.view"]._render_template(
            "website_event_waiting_list.registration_attendee_details",
            {
                "tickets": tickets,
                "event": event,
                "availability_check": availability_check,
                "waiting_list_check": waiting_list_check,
                "warning_msg": warning_msg,
            },
        )

    @http.route(
        ["""/event/<model("event.event"):event>/registration/confirm"""],
        type="http",
        auth="public",
        methods=["POST"],
        website=True,
    )
    def registration_confirm(self, event, **post):
        if not event.can_access_from_current_website():
            raise werkzeug.exceptions.NotFound()

        registrations = self._process_attendees_form(event, post)
        waiting_list_check = post.get("waiting_list_check")

        # If post was not for waiting_list and trying to register more seats
than available
        # Or trying to register more seats for a ticket than available
        # return 404
        for ticket in event.event_ticket_ids:
            ticket_count = 0
            for registration in registrations:
                if ticket.id == registration["event_ticket_id"]:
                    ticket_count += 1
            if (
                not waiting_list_check
                and ticket.seats_max
                and ticket_count > ticket.seats_available
            ):
                return request.render(
                    "website_event_waiting_list.registration_fail",
                    {
                        "warning_msg": "You tried to order more tickets than "
                        "tickets available.",
                        "event": event,
                    },
                )
        if (
```

```
            not waiting_list_check
            and event.seats_limited
            and len(registrations) > event.seats_available
        ):
            return request.render(
                "website_event_waiting_list.registration_fail",
                {
                    "warning_msg": "You tried to order more tickets than
available seats.",
                    "event": event,
                },
            )

        attendees_sudo = self._create_attendees_from_registration_post(
            event, registrations
        )

        return request.render(
            "website_event.registration_complete",
            self._get_registration_confirm_values(event, attendees_sudo),
        )

    def _process_attendees_form(self, event, form_details):
        """ Process data posted from the attendee details form.
        :param form_details: posted data from frontend registration form, like
            {'1-name': 'r', '1-email': 'r@r.com', '1-phone': '', '1-
event_ticket_id': '1'}
        """
        allowed_fields = request.env[
            "event.registration"
        ]._get_website_registration_allowed_fields()
        registration_fields = {
            key: v
            for key, v in request.env["event.registration"]._fields.items()
            if key in allowed_fields
        }
        registrations = {}
        global_values = {}
        for key, value in form_details.items():
            # skip loop if key not splittable
            try:
                counter, attr_name = key.split("-", 1)
            except ValueError:
                continue
            field_name = attr_name.split("-")[0]
            if field_name not in registration_fields:
                continue
            elif isinstance(
                registration_fields[field_name], (fields.Many2one,
fields.Integer)
            ):
                value = (
                    int(value) or False
                )  # 0 is considered as a void many2one aka False
            else:
                value = value

            if counter == "0":
                global_values[attr_name] = value
            else:
                registrations.setdefault(counter, dict())[attr_name] = value
```

```
        for key, value in global_values.items():
            for registration in registrations.values():
                registration[key] = value

        return list(registrations.values())

    @http.route(

['/event/<model("event.event"):event>/registration/manage/<string:code>'],
        type="http",
        auth="public",
        website=True,
    )
    def confirm_url_template(self, event, code, **post):
        """
        Return correct confirmation page depending on state
        Confirm state changes on post
        """
        for registration in event.sudo().registration_ids:
            if registration.sudo().access_token == code:
                render_values = {
                    "event": event,
                    "registration": registration,
                    "ticket": registration.event_ticket_id,
                }
                if post:
                    new_state = post.get("new_state")
                    cur_state = post.get("current_state")
                    if (
                        cur_state == "wait"
                        and new_state == "open"
                        and event.sudo().seats_available >= 1
                    ):
                        registration.sudo().write({"state": "open"})
                    if new_state == "cancel":
                        registration.sudo().write({"state": "cancel"})

                if registration.sudo().state in ["wait", "cancel", "open"]:
                    return request.render(
                        "website_event_waiting_list.confirm_waiting",
render_values
                    )
        return request.render("website.page_404")
```

Appendix 3: Custom Sass rules for survey registration form

```
/*************************************************************
                        Common Style
 *************************************************************/
.o_survey_form,
.o_survey_print,
.o_survey_session_manage,
.o_survey_quick_access {
    .o_survey_question_error {
        height: 0px;
        transition: height 0.5s ease;
        line-height: 4rem;
        &.slide_in {
            height: 4rem;
        }
    }

    fieldset[disabled] {
        .o_survey_question_text_box,
        .o_survey_question_date,
        .o_survey_question_datetime,
        .o_survey_question_numerical_box {
            padding-left: 0px;
        }
    }

    .o_survey_question_text_box,
    .o_survey_question_date,
    .o_survey_question_datetime,
    .o_survey_question_numerical_box {
        border: 0px;
        border-bottom: 1px solid $primary;
        &:disabled {
            color: black !important;
            border-color: $gray-600;
            border-bottom: 1px solid $gray-600;
        }
        &:focus {
            box-shadow: none;
        }
    }

    .o_survey_form_date .input-group-append {
        right: 0;
        bottom: 5px;
        top: auto;
    }

    .o_survey_choice_btn {
        transition: background-color 0.3s ease;
        flex: 1 0 300px;
        color: $primary;

        span {
            line-height: 25px;
        }
        i {
            top: 0px;
            font-size: large;
```

```
            &.fa-check-circle,
            &.fa-check-square {
                display: none;
            }
        }

        &.o_survey_selected i {
            display: none;
            &.fa-check-circle,
            &.fa-check-square {
                display: inline;
            }
        }
    }

    input::placeholder,
    textarea::placeholder {
        font-weight: 300;
    }


.o_survey_page_per_question.o_survey_simple_choice.o_survey_minimized_display,

.o_survey_page_per_question.o_survey_multiple_choice.o_survey_minimized_display,
    .o_survey_page_per_question.o_survey_numerical_box,
    .o_survey_page_per_question.o_survey_date,
    .o_survey_page_per_question.o_survey_datetime {
        // 'pixel perfect' layouting for choice questions having less than 5
choices in page_per_question mode
        // we use media queries instead of bootstrap classes because they don't
provide everything needed here
        @media (min-width: 768px) {
            width: 50%;
            position: relative;
            left: 25%;
        }
    }

    .o_survey_question_matrix {
        td {
            min-width: 100px;
            i {
                font-size: 22px;
                display: none;
                &.o_survey_matrix_empty_checkbox {
                    display: inline;
                }
            }
            .o_survey_choice_key {
                left: 10px;
                right: auto;
                top: 12px;
                > span > span {
                    top: 0px;
                }
            }

            &.o_survey_selected {
                i {
                    display: inline;
                    &.o_survey_matrix_empty_checkbox {
```

```
                        display: none;
                    }
                }
            }
        }
        thead {
            th:first-child {
                border-top-left-radius: 0.25rem;
            }
            th:last-child {
                border-top-right-radius: 0.25rem;
            }
        }
        tbody tr:last-child {
            th {
                border-bottom-left-radius: 0.25rem;
            }
            td:last-child {
                border-bottom-right-radius: 0.25rem;
            }
        }
    }
}

.o_survey_form,
.o_survey_session_manage {
    .o_survey_question_matrix {
        th {
            background-color: $primary;
        }
        td {
            background-color: rgba($primary, 0.2);
        }
    }
}

/*************************************************************
                    Form Specific Style
 *************************************************************/

.o_survey_form {
    min-height: 25rem;

    .o_survey_choice_btn {
        cursor: pointer;
        background-color: rgba($primary, 0.1);
        box-shadow: $primary 0px 0px 0px 1px;

        &.o_survey_selected {
            box-shadow: $primary 0px 0px 0px 2px;
        }

        &:hover {
            background-color: rgba($primary, 0.3);
            .o_survey_choice_key span.o_survey_key {
                opacity: 1;
            }
        }
    }

    .o_survey_choice_key {
```

```
        width: 25px;
        height: 25px;
        border: 1px solid $primary;
        span {
            font-size: smaller;
            top: -1px;
            &.o_survey_key {
                right: 21px;
                border: 1px solid $primary;
                border-right: 0px;
                height: 25px;
                transition: opacity 0.4s ease;
                white-space: nowrap;
                opacity: 0;
                span {
                    top: -2px;
                }
            }
        }
    }

    .o_survey_question_matrix td:hover {
        background-color: rgba($primary, 0.5);
        cursor: pointer;
        .o_survey_choice_key span.o_survey_key {
            opacity: 1;
        }
    }
}
```