Freedom Manufaktur – Innovation Project

Intro

Solution Overview

The proposed solution will be a bot, developed with the Microsoft Bot Framework to support employees by taking away stupendous work and sourcing it out to the bot. Additionally, Azure Functions will be conbined with the bot to perform database queries based on the users demands. The solution will include the ability to create workflows, so-called "Trekk"s. The options are automatically populated by taking a look at an existing option database while keeping past decisions in mind. In addition, Azure Functions are used to start and stop each Trekk, list exisiting ones and to save data in a blob storage.

Key technologies

- Microsoft Bot Framework
- Microsoft Cognitive Services (LUIS = Language Understanding Intelligent Service)
- Azure Functions
- Azure SQL Database

Core team

- Christian Zanler | Chief Entfessler
- Martin Geißler | Wegbereiter
- Oliver Keller | AEM Microsoft Germany
- Manuela Rink | Technical Evangelist Microsoft Germany
- Daniel Heinze | Technical Evangelist Microsoft Germany

Customer profile

Established 2015, René Vierkorn and Christian Zander address the problems and challenges of small and medium sized businesses and large scaled enterprises, local and global. They develop human 2 human (b2b) software in different lines of business. Solutions for exactly one problem within 9-month timeframe and costs below costumer's budget limit. This results fast sales cycles and happy costumers. They accompany companies as well as coach or mentor in the field of company growth, marketing, sales or staff. Special attention is given to development of personality of the management. Gearing startups up for the market is part of our competence, too. freedom manufaktur – path of in freedom

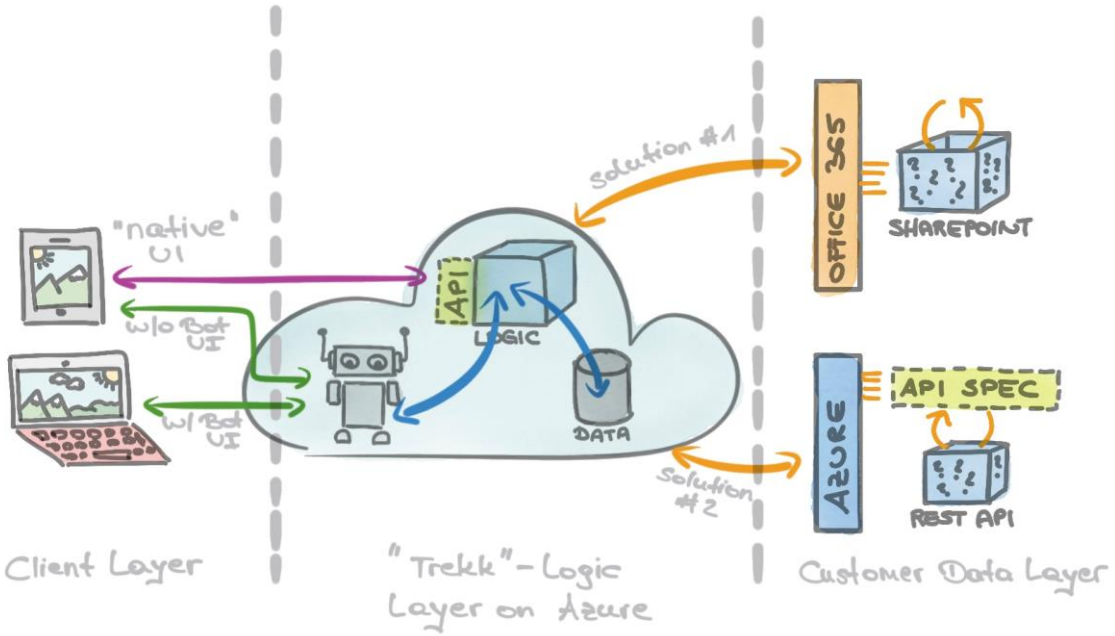
Problem statement

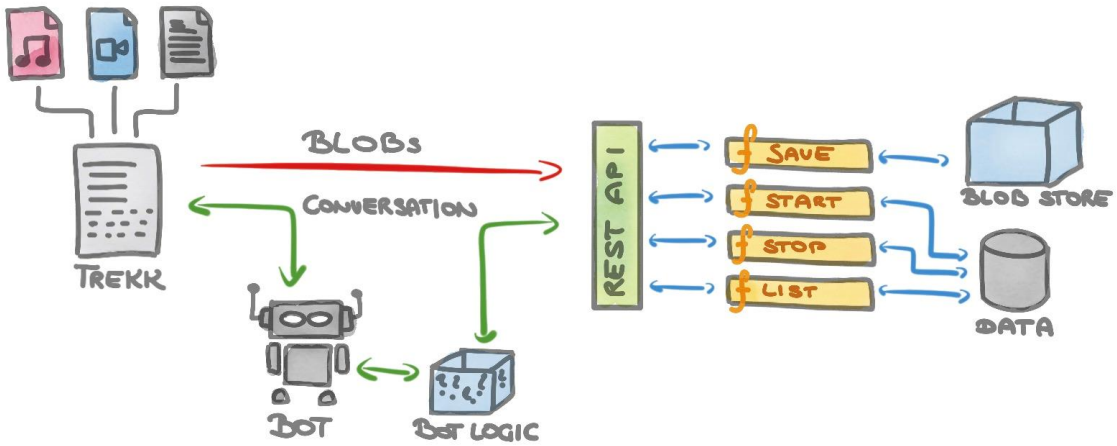
Craftsmen Companies, Reseller, Facility Management, any company which is sending out employees for providing services have the problem that the employees did not file their work properly with the result that invoices are incomplete or even not written. Existing software is cumbersome in using it and way too complicated, so employees doesn't enter their tasks properly.

Solution and steps

The user of the system is presented with options at the start of the bot interaction to guide him through the experience. He can choose to start a new Trekk, which gathers relevant info and opens the work on a specific task. This Trekk can be closed at any time through the second option to list the Trekks and editing them. Furthermore, in the third interaction the user can upload data to the specific Trekk or he is presented with company information by asking for a specific trait.

Architecture





Technical delivery

To get started working with bots, take a look at the following links first:

- Documentation Bots
- Step-by-step guide

To get started with Azure functions, start by looking at the introduction:

- Introduction Azure Functions
- Get Started Azure Functions

Bot Patterns

The implemented bot consist of multiple dialogs, these are:

- Root Dialog: The main dialog which handles the routing of the requests, sends the welcome message and displays the results. The RootDialog greets the user by using the SendWelcomeMessageAsync() method. The choseable options are implemented as a PromptDialog.Choice with its options populated based on the already existing Trekks. The methods QuestionDialogResumeAfter and NewTrekkDialogResumeAfter are called, when the child dialogs based on the chosen options are finished.

```
public class RootDialog : IDialog<object>
{
    private const string Selectedtrekk = "selectedTrekk";
    private const string Trekklist = "TrekkList";

    public Task StartAsync(IDialogContext context)
    {
        context.Wait(this.MessageReceivedAsync);

        return Task.CompletedTask;
    }

    private async Task MessageReceivedAsync(IDialogContext context, IAwaitable<object>
result)
    {
        await this.SendWelcomeMessageAsync(context);
    }

    private async Task SendWelcomeMessageAsync(IDialogContext context)
    {
        string welcomeMessage = "Hallöchen, ick bin deen Trekka. Was willste machen?";

        List<Trekk> trekkList = await AzureFunctionAPI.GetAllTrekksAsync();

        context.PrivateConversationData.SetValue(Trekklist, trekkList);

        PromptDialog.Choice(
            context: context,
            options: options,
            prompt: welcomeMessage,
            retry: "Hab ick nich verstanden.",
            promptStyle: PromptStyle.Auto,
            attempts: 2,
            resume: this.welcomeMessageResumeAfterAsync);
    }

    private async Task welcomeMessageResumeAfterAsync(IDialogContext context,
IAwaitable<string> result)
    {
        string choice = await result;

        switch (choice)
        {
            case "Neuer Trekk":
                context.Call(new NewTrekkDialog(), this.NewTrekkDialogResumeAfter);
                break;
```

```
            case "Frag mich was":
                context.Call(new QuestionDialog(), this.QuestionDialogResumeAfter);
                break;
            case "Trekk anzeigen":
                break;
            case "Aktuellen Trekk stoppen":
                await this.actionOnSelectedTrekkAsync(context, new
AwaitableFromItem<string>("Stopp"));
                return;

            case "Trekk auswählen":
                List<Trekk> trekks;
                if (!context.PrivateConversationData.TryGetValue<List<Trekk>>(Trekklist,
out trekks))
                {
                    context.Fail(new Exception("Missing trekks"));
                }
                PromptDialog.Choice(
                    context: context,
                    options: trekks.Select(t => t.DisplayName),
                    prompt: "Bitte trekk auswählen",
                    promptStyle: PromptStyle.Auto,
                    attempts: 1,
                    resume: this.selectedTrekkAsync);

                break;
            default:
                break;
        }
    }

    private async Task selectedTrekkAsync(IDialogContext context, IAwaitable<string>
result)
    {
        string choice = await result;
        List<Trekk> trekks;
        if (!context.PrivateConversationData.TryGetValue<List<Trekk>>(Trekklist, out
trekks))
        {
            context.Fail(new Exception("Missing trekks"));
        }

        Trekk trekk = trekks.First(t => t.DisplayName == choice);
        context.PrivateConversationData.SetValue(Selectedtrekk, trekk);

        PromptDialog.Choice(
            context: context,
            options: new string[] { "Stopp", "So lassen" },
            prompt: "Was willst du tun?",
            promptStyle: PromptStyle.Auto,
            attempts: 1,
            resume: this.actionOnSelectedTrekkAsync);
    }

    private async Task actionOnSelectedTrekkAsync(IDialogContext context,
IAwaitable<string> result)
    {
        string choice = await result;
        if (choice == "Stopp")
        {
            Trekk trekk;
            if (!context.PrivateConversationData.TryGetValue<Trekk>(Selectedtrekk, out
trekk))
            {
                context.Fail(new Exception("Missing trekk"));
            }
            await AzureFunctionAPI.StopTrekk(trekk.Id);
```

```
            await context.PostAsync("Ok, hab ich gestoppt");
        }

        await this.SendWelcomeMessageAsync(context);
    }

    private async Task QuestionDialogResumeAfter(IDialogContext context, IAwaitable<string>
result)
    {
        string response = await result;

        await context.PostAsync(response);

        await this.SendWelcomeMessageAsync(context);
    }

    private async Task NewTrekkDialogResumeAfter(IDialogContext context,
IAwaitable<StartObject> result)
    {
        StartObject startObject = await result;

        // Save new trekk
        await AzureFunctionAPI.StartTrekk(startObject);

        await context.PostAsync("Neuer Trekk wurde angelegt.");

        await this.SendWelcomeMessageAsync(context);
    }
}
```

- NewTrekk Dialog: The dialog that performs the creation of a new Trekk. This dialog guides the user through multiple options, based on the selections made throughout the dialog. For example, if a user chooses a new Trekk for a customer assignment, at first the selection for a car is presented.

```
public class NewTrekkDialog : IDialog<StartObject>
{
    private List<TrekkTemplate> _templates = new List<TrekkTemplate>();

    private List<TrekkActionParameter> _actionParameters = new
List<TrekkActionParameter>();

    private readonly StartObject _startObject = new StartObject();

    private string _selection = "";

    public Task StartAsync(IDialogContext context)
    {
        this._templates = AzureFunctionAPI.GetTrekkTemplates();

        List<string> options = this._templates.Select(x => x.DisplayName).ToList();

        PromptDialog.Choice(
            context: context,
            options: options,
            resume: this.DisplayNameResumeAfter,
            promptStyle: PromptStyle.Auto,
            prompt: "Wähle deinen Trekk aus.",
            retry: "Bitte einen Trekk aus der Liste wählen.",
            attempts: 2
        );

        return Task.CompletedTask;
    }
```

```
    private async Task DisplayNameResumeAfter(IDialogContext context, IAwaitable<string>
result)
    {
        // Get the users selected track type
        this._selection = await result;

        // Get the TrekkTemplateId based on the unique DisplayName of the TrekkTemplate
        int id = this._templates.Where(x => x.DisplayName == this._selection).Select(x =>
x.Id).FirstOrDefault();

        // Set command to Start
        string actionCommand = "Start";

        // Get the ActionId needed to get the ActionParameters
        int actionId = AzureFunctionAPI.GetActionId(id, actionCommand);

        // Get the ActionParameters
        this._actionParameters = AzureFunctionAPI.GetActionParameters(actionId);

        await this.PerformDataGathering(context);
    }

    private Task PerformDataGathering(IDialogContext context)
    {
        if (this._actionParameters.Count == 0)
        {
            this._startObject.DisplayText = this._selection;
            context.Done(this._startObject);
        }
        else
        {
            if (this._actionParameters.First().Type == "SelectResource")
            {
                int resourceCategoryId = this._actionParameters.First().ResourceCategoryId;

                PromptDialog.Choice(
                    context: context,
                    options: AzureFunctionAPI.GetResources(resourceCategoryId),
                    resume: this.SelectResourceResumeAfter,
                    promptStyle: PromptStyle.Auto,
                    prompt: "Wähle dein Nummernschild.",
                    retry: "Bitte wähle eines der Nummernshilder aus der Liste.",
                    attempts: 2
                );
            }
            else
            {
                PromptDialog.Number(
                    context: context,
                    resume: this.ResourcePropertyResumeAfter,
                    prompt: "Gib den Kilometerstand an.",
                    retry: "Die Eingabe wurde nicht erkannt, bitte einen valide
Kilometerzahl ohne Zeichen eingeben.",
                    attempts: 2
                );
            }
        }

        return Task.CompletedTask;
    }

    private async Task SelectResourceResumeAfter(IDialogContext context, IAwaitable<string>
result)
    {
        string licenseNumber = await result;
```

```
        this._startObject.LicenseNumber = licenseNumber;

        this._actionParameters.Remove(this._actionParameters.First());

        await this.PerformDataGathering(context);
    }

    private async Task ResourcePropertyResumeAfter(IDialogContext context, IAwaitable<long>
result)
    {
        long km = await result;

        this._startObject.Km = (int)km;

        this._actionParameters.Remove(this._actionParameters.First());

        await this.PerformDataGathering(context);
    }
}
```

- Question Dialog: The question dialog handles the users questions towards the company, product and more by using the Cognitive Service LUIS. It is pre-trained to recognize intents for uploading and analysis, answering questions to the company and regarding the product.

```
public class QuestionDialog : IDialog<string>
{
    private LuisResult luisResult;

    public async Task StartAsync(IDialogContext context)
    {
        await context.PostAsync("Was möchtest du wissen, oder schick mir was?");

        context.Wait(this.MessageReceivedAsync);
    }

    public async Task MessageReceivedAsync(IDialogContext context,
IAwaitable<IMessageActivity> argument)
    {
        IMessageActivity activity = await argument;
        if (activity.Attachments != null && activity.Attachments.Count > 0)
        {
            await context.PostAsync("Ich habe attachments gefunden");

            byte[] bytes = await this.getContentFromAttachment(activity,
activity.Attachments[0]);

            string url = await AzureFunctionAPI.SaveBlob(bytes);

            HeroCard heroCard = new HeroCard()
                {
                    Images = new List<CardImage>()
                        {
                            new CardImage(url, alt: url),
                        },
                    Text = url
                };
            IMessageActivity response = context.MakeMessage();
            response.Attachments = new List<Attachment>();
            response.Attachments.Add(heroCard.ToAttachment());

            await context.PostAsync(response);

            context.Done("Fertig");
            return;
```

```
        }
        string TextToLuis = activity.Text;

        this.luisResult = await LuisApi.GetLuisResult(TextToLuis);

        await this.HandleLuisMessage(context);
    }

    private async Task<byte[]> getContentFromAttachment(IActivity activity, Attachment
attachment)
    {
        using (ConnectorClient connectorClient = new ConnectorClient(new
Uri(activity.ServiceUrl)))
        {
            string token = await
((MicrosoftAppCredentials)connectorClient.Credentials).GetTokenAsync();
            Uri uri = new Uri(attachment.ContentUrl);
            using (HttpClient httpClient = new HttpClient())
            {
                if (uri.Host.EndsWith("skype.com") && uri.Scheme == "https")
                {
                    httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);
                    httpClient.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/octet-stream"));
                }
                else
                {
                    httpClient.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue(attachment.ContentType));
                }
                return await httpClient.GetByteArrayAsync(uri);
            }
        }
    }

    private Task HandleLuisMessage(IDialogContext context)
    {
        string responseText;

        switch (this.luisResult.topScoringIntent.intent)
        {
            case "ProductInfo":
                responseText = "MyTrekka, das supertolle Erfassungssystem
#FreedomManufaktur #SuperTE #isso";
                break;
            case "CompanyInfo":
                responseText = "Freedom Manufaktur, besucht uns unter https://freedom-
manufaktur.com";
                break;
            default:
                responseText = "Geht nicht #fail";
                break;
        }

        context.Done(responseText);

        return Task.CompletedTask;
    }
}
```

Azure Functions

The functional solution utilizes four different Azure functions to provide an easy interface for the bot to gather and alter information in the background databases. The four functions are implemented as follows:

1. **Start Trekk:** Takes in the information given by the user which creates a database entry in the underliing Azure database.
2. **Stop Trekk:** Sends the command to stop a Trekk to the database. This will trigger additional actions in the backend, like automatic receipt creation.
3. **List Trekks:** Returns all Trekks, that are currently not stopped to the requesting user. This is done by performing a query on the database.
4. **Save blob:** Saves an image, that is related to an existing, non-stopped Trekk in a blob storage.

Core Bot Capabilities

Azure function connection

The connection with the Azure functions is mainly handled by the class **AzureFunctionAPI**. Contained in this class are methods, performing HHTP requests to connect to each Azure function:

```
internal static async Task StartTrekk(StartObject s)
{
    StartTrekkInput input = new StartTrekkInput()
        {
            DisplayText = s.DisplayText,
            Km = s.Km,
            LicenseNumber = s.LicenseNumber
        };

    using (HttpClient h = new HttpClient())
    {
        HttpResponseMessage response = await h.PostAsJsonAsync(_saveTrekkUrl, input);
        response.EnsureSuccessStatusCode();
    }
}

internal static async Task StopTrekk(string id)
{
    StopTrekkInput input = new StopTrekkInput()
        {
            Id = id,
        };

    using (HttpClient h = new HttpClient())
    {
        HttpResponseMessage response = await h.PostAsJsonAsync(_stopTrekkUrl, input);
        response.EnsureSuccessStatusCode();
    }
}

internal static async Task<List<Trekk>> GetAllTrekksAsync()
{
    List<Trekk> trekks = new List<Trekk>();
    using (HttpClient h = new HttpClient())
    {
        string response = await h.GetStringAsync(_listTrekskUrl);
        ListTrekksOutput muh2 = JsonConvert.DeserializeObject<ListTrekksOutput>(response);

        foreach (Dictionary<string, object> pair in muh2.Data)
        {
            trekks.Add(new Trekk()
                {
                    Id = (string)pair["id"],
                    DisplayName = (string)pair["displayText"]
```

```
                });
            }
        }
        return trekks;
    }

    internal static List<TrekkTemplate> GetTrekkTemplates()
    {
        return _trekkTemplates;
    }

    internal static int GetActionId(int id, string actionCommand)
    {
        int actionId = _actions.Where(x => x.DisplayName == actionCommand && x.TrekkTemplateId
    == id).Select(x => x.Id).FirstOrDefault();

        return actionId;
    }

    internal static List<TrekkActionParameter> GetActionParameters(int actionId)
    {
        List<TrekkActionParameter> actionParameterList = _actionParameters.Where(x =>
    x.ActionId == actionId).ToList();

        return actionParameterList;
    }

    internal static List<string> GetResources(int resourceCategoryId)
    {
        List<string> selectedResources = _resources.Where(x => x.RessourceCategoryId ==
    resourceCategoryId).Select(x => x.DisplayName).ToList();

        return selectedResources;
    }

    internal static async Task<string> SaveBlob(byte[] bytes)
    {
        using (HttpClient h = new HttpClient())
        {
            HttpResponseMessage response = await
    h.PostAsync($"{_saveBlobUrl}&imgName={Guid.NewGuid():N}.png", new ByteArrayContent(bytes));
            response.EnsureSuccessStatusCode();

            string url = await response.Content.ReadAsStringAsync();
            return url;
        }
    }
```

Bot Intelligence

The Cognitive Service called LUIS is used, to support the QuestionDialog and blob creation for existing Trekks. The query made by the user is send to the service, which then analyses it and specifies the intent of the query. The existing intents are:

- **None**: No intent is recognized.
- **ProductInfo**: Returns the product information to the user.
- **CompanyInfo**: Returns the company information to the user.
- **SaveBlob**: Saves a blob with the given properties on the blob storage through an Azure function.

The following LUIS Bot Sample explains how to develop a LUIS bot.

SDKs used, languages, etc.

The following technologies are used for the implementation of the application:

- C#: The language the bot is build in.
- Bot Builder SDK: The SDK provided by Microsoft that is used to build the bot
- JSON: The response of the API is given as a JSON file. It is deserialized by the Newtonsoft.Json library
- REST: The LUIS API is a REST interface, which is called by the bot by using the built-in library WebRequest from C#. For more info on LUIS, go to the following link: LUIS code story

Conclusion

The developed myTrekka bot solution enables all users to have a simple, fast, intuitive and familiar visual interface to perform work tracking throughout the whole customer assignment process. This tool therefor allows for a more focussed and optimized workflow by the user of the bot. Additionally, the whole tracking process is streamlined and optimized be available as a highly scaleable system for other companies to deploy.

General lessons:

- LUIS needs around 10 or more samples for each intent to work as desired for some cases. To optimize the bot includes continuous training of the service.
- To reduce bandwidth and the performance of the bot, it is better to call Azure functions directly and not implementing a database connection for each method.

Next steps:

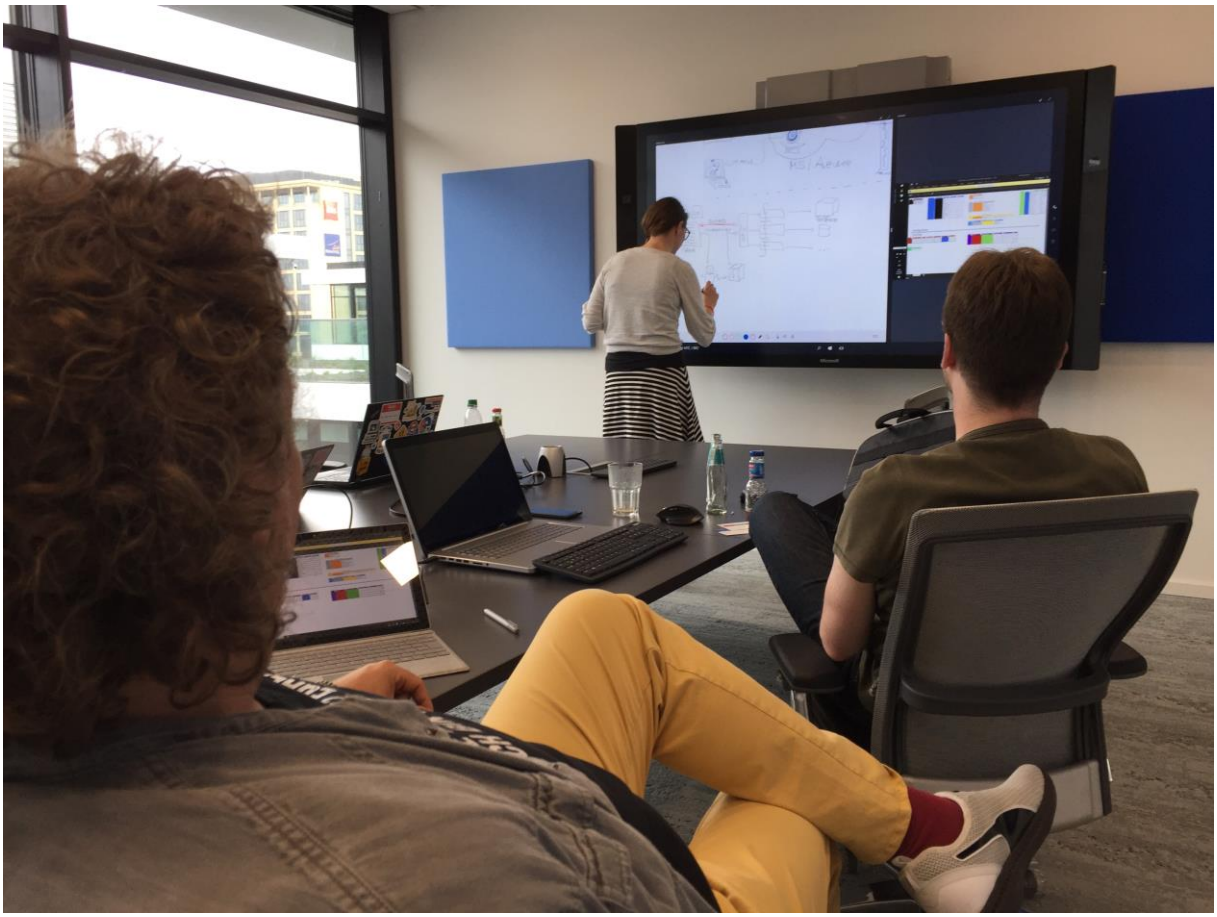
- Customer testing
- Expension with audio data
- Connection to workflow for automated receipt creation

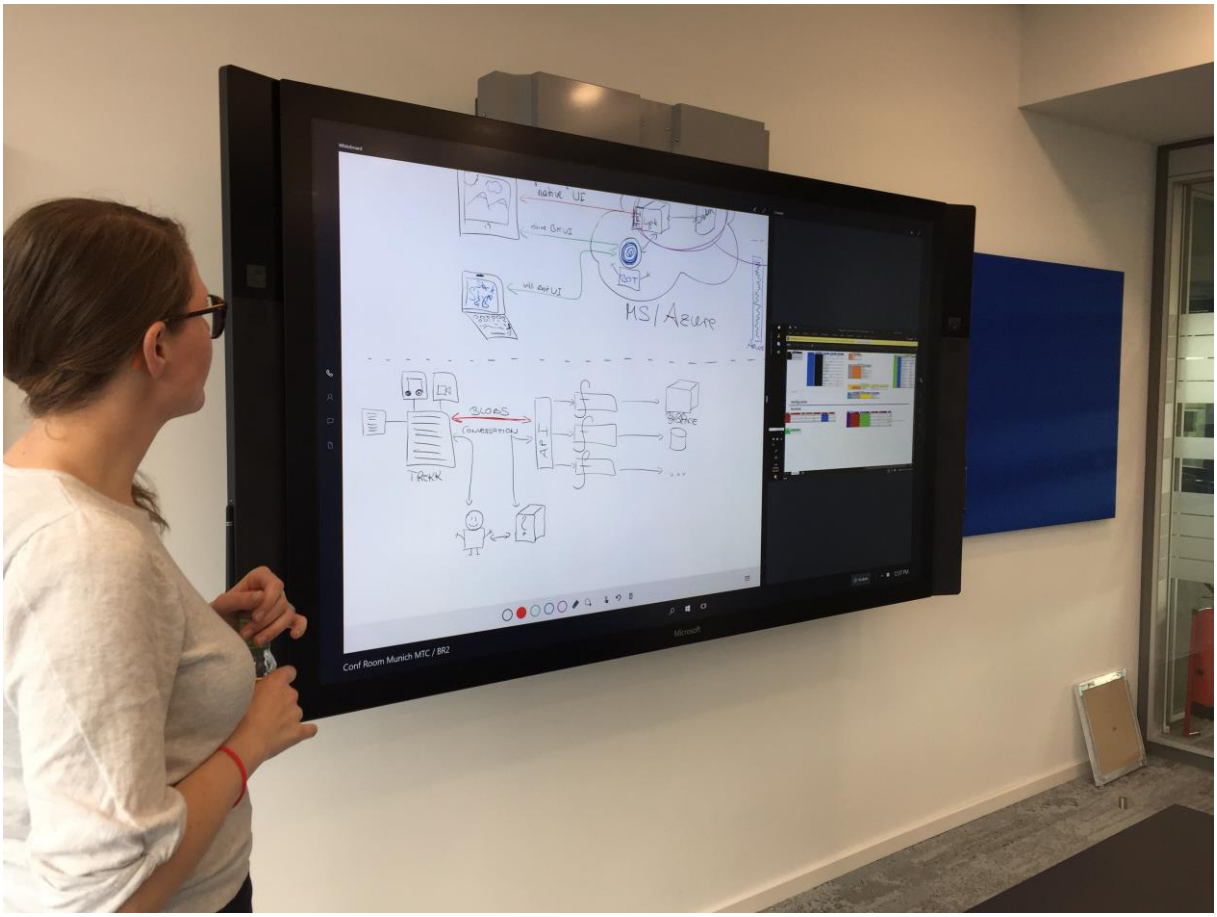
Additional resources

In this section, include a list of links to resources that complement your story, including (but not limited to) the following:

- Documentation Bots Framework
- Blog posts
- GitHub repos
- LUIS
- Cognitive Services
- Azure Functions

Workshop images




MS/Azure
Conf Room Munich MTC / BR2