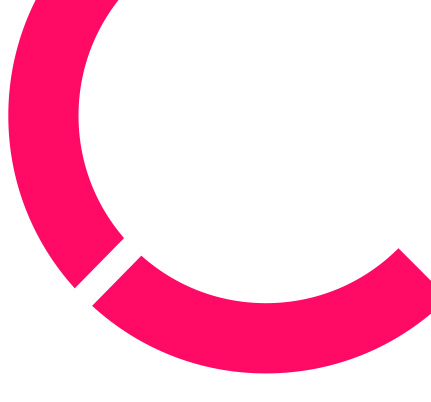
**Konsta Karhula, Roope Kotiaho, Joni Priuska**

**CentriaAi Software Documentation**

**CENTRIA-AMMATTIKORKEAKOULU**

**Huhtikuu 2024**

****

**TABLE OF CONTENTS**

[1 FOREWORD 1](#_Toc165280613)

[2 installation and setup instructions 2](#_Toc165280614)

[2.1 Getting your API-key 2](#_Toc165280615)

[2.2 Downloading the application 3](#_Toc165280616)

[2.3 Installing the application 3](#_Toc165280617)

[3 USER GUIDES 5](#_Toc165280618)

[3.1 Sending a Message 5](#_Toc165280619)

[3.2 Starting a New Chat 6](#_Toc165280620)

[3.3 Closing the application 6](#_Toc165280621)

[4 TROUBLESHOOTING AND FAQS 7](#_Toc165280622)

[4.1 Troubleshooting 7](#_Toc165280623)

[4.2 Frequently asked questions 8](#_Toc165280624)

[5 RELEASE NOTES AND VERSION HISTORY 9](#_Toc165280625)

[5.1 Release notes of each version of Centria AI. 9](#_Toc165280626)

[5.2 Upcoming features 10](#_Toc165280627)

[6 Source code REVIEW 11](#_Toc165280628)

[6.1 Libraries and GUI-initialization 11](#_Toc165280629)

[6.2 Semantic Kernel 13](#_Toc165280630)

[6.3 Chat Functions 15](#_Toc165280631)

[6.4 Button Clicks 16](#_Toc165280632)

[6.5 Message Bubbles 17](#_Toc165280633)

[6.6 Calling Bubbles 18](#_Toc165280634)

[6.7 Bubble’s construction 18](#_Toc165280635)

[7 development process 20](#_Toc165280636)

[7.1 The early graphical interface 20](#_Toc165280637)

[7.2 Newer versions of the AI 22](#_Toc165280638)

[7.3 API Keys and documentation 22](#_Toc165280639)

[7.4 Exporting the app 23](#_Toc165280640)

**FIGURES**

FIGURE 1. Essentials files underlined 3

FIGURE 2. Changing the API-key 4

FIGURE 3. The view of the application at the start 5

FIGURE 4. Welcome message on GUI 16

FIGURE 5. The GUI of the user and AI message bubbles 17

FIGURE 6. CMF version of the AI 20

FIGURE 7. List of prompts 20

FIGURE 8. The problem with the spacing of the bubbles 21

# FOREWORD

The CentriaAI is currently a simple AI which can answer individual questions. The future planned use case would be implementations into websites as a helper, or into applications for easy question answering service. With memory and information embedded into the AI it would be able to answer complex questions about the application it is running with, for more in depth problem solving. Other use cases would be personal assistants for desktop applications which could be configured to the user.

This documentation will go through firstly user instructions on installing the application and common problems. Then version history and upcoming features. Source code is commented and explained line-by-line. And in the end a chapter about the development process will be found.

For a common user, ff you just want to use the application, you can refer to the next few chapters on user guides. There you will find instructions on installing the application and getting your very own API-key from OpenAI. On the other hand, if you’re a developer interested in the inner workings of this application, you’ll find the source code documentation and details about the development process towards the end of this documentation.

# installation and setup instructions

The final version of the CentriaAI is quite simple to use as it comes with an executable file. The only change the user must make is to insert their own API key in the secrets.json file that is included inside the folder. You can follow these instructions to start using the application.

## Getting your API-key

Our CentriaAi uses OpenAI’s ChatGPT model 3.5 which isn’t free on our software. Visit the OpenAI website and sign up with an OpenAI account to gain access to their API, if you don’t already have an account. The registration process requires some basic information and your agreement to their terms of service. Select a payment plan that best fits your usage needs. You can determine this by considering the limits and pricing details provided by OpenAI.

Once your account is approved and set up. Click your profile icon at the top-right corner of the page and select "View API Keys". Click “Create New Secret Key” to generate a new API key. Make sure to save the API key as soon as possible. Once the window showing it closes, you won’t be able to reopen it.

Please note that you can create an OpenAI API key for free. New free trial users receive five US Dollars’ worth of credit. However, this expires after three months. Once your credit has been used up or expires, you can enter billing information to continue using the API of your choice. Keep in mind that if you don’t enter any billing information, you will still have login access but won’t be able to make further API requests.

Please remember, your API key is sensitive information. It’s crucial to keep it secure and avoid losing it, as you cannot retrieve the same key if lost. Furthermore, your API key links your account to you as a user. If your API key were to leak, third parties could potentially use it, resulting in tokens being billed to your account. If you need more information on getting a API-key from OpenAI you can refer to their own instructions.

## Downloading the application

Installing the application is easy, you just need to head to CentriaAI’s GitHub site and down-load the application package. To do that you need to navigate to the main page of the repository on GitHub. Above the list of files, click the green button labelled “Code”. In the dropdown menu, select “Download ZIP”. A download will start, and the ZIP file of the repository will be downloaded to your default download location.

Please note that the downloaded file will be a ZIP file. This file is a compressed archive of all the files in the repository. You will need to extract the ZIP file to access the files inside. Most operating systems have built-in tools to extract ZIP files, but there are also many third-party tools available for this purpose.

## Installing the application

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti, logo

Kuvaus luotu automaattisestiFIGURE 1: Essential files underlined.

Once that is done you have to open the CentriaAI folder. Among the various files in this folder, you need to focus on two key files: secret.json and CentriaAI.exe. Secret.json file is the file where your API-key will be located and CentriaAI.exe is the application itself. You can refer to figure 1.

Now, you need to find your API-key that you created in previous chapter. Open the secret.json file in a text editor of your choice. You will find the text “Api key is inserted here” within this file. Replace this placeholder text with your personal OpenAI API key. Please do ensure that your API key is enclosed in double quotes, you can refer to the figure 2. Do not modify the file in any other way.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, viiva, Fontti

Kuvaus luotu automaattisesti

FIGURE 2: Changing the API-key

Once your API key is saved in the secret.json file, remember to save any changes you make to the secret.json file before launching the application. Do not make modifications in other files nor delete them.

Locate the CentriaAI.exe file in the same folder. Double-click on CentriaAI.exe to launch the application. You should now be inside the CentriaAI application and ready to use it. If you’re unsure how to navigate or use the application, refer to the next chapter on user guides for detailed instructions and assistance.

# USER GUIDES

Using this app is quite straightforward. You can use the following instructions if you find yourself lost.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, ohjelmisto, multimedia

Kuvaus luotu automaattisesti

FIGURE 3: The view of the application at the start.

## Sending a Message

To send a message to the AI chatbot you need to locate the text box at the bottom of the chat interface and then type your message into the text box. Click the send button, which is located right from the text box marked as two angle brackets. You can refer to figure 3. Please do note that the enter-function to send the message doesn’t work. This allows user to use paragraph breaks in the text box.

Your message will appear as a bubble in the chat interface, followed by the AI’s response in a separate bubble. The AI’s answer should take only few seconds, if after ten seconds you haven’t got an answer please do refer to the next chapter on Troubleshooting and FAQs.

After enough bubbles have been added to fill the screen, you can scroll through them and the newest message should be at the bottom of the screen above the text box. Please do note that this version of Centria AI does not remember pervious messages.

## Starting a New Chat

If you wish to start a new conversation with the AI chatbot you need to locate and click the ‘Uusi Keskustelu’ button on the chat interface. It is located on the centre top bar. This will clear the chat interface and print the welcome message again and you can start a new conversation.

Again, please remember that this feature is not really needed as the app doesn’t remember previous messages. As it is your very own API-key that is being used to generate the AI-answers this app doesn’t have any token limits.

## Closing the application

To close the application you need to locate the ‘X’ button at the top-right corner of the application window. Click the ‘X’ button and the application will close. Please note that closing the application will not save your chat history. Make sure to save any important information before closing the application.

# TROUBLESHOOTING AND FAQS

The application should work as it is, but if you come across some problems, hopefully these will help you to find a solution.

## Troubleshooting

**The AI doesn’t respond to my messages.**

* Make sure your device is connected to the internet.
* Try refreshing the application or restarting your device.
* It is possible that the problem is with your API-key, do check that it has not expired.
* If the problem persists, contact the support team.

**My messages aren’t being sent.**

* Check your internet connection and try again.
* Make sure you’re clicking the ‘Send’ button after typing your message.
* If the issue continues, it might be a temporary server issue. Wait for a few minutes and try again.

**The ‘New Chat’ button isn’t working.**

* Try refreshing the application.
* If the button still doesn’t work, there might be a bug. Report it to the support team.

**The application closes unexpectedly.**

* Make sure your device has enough memory to run the application.
* Restart you PC
* Reinstall the application
* If the issue persists, contact the support team.

**The AI is not understanding my questions.**

* Make sure your question is clear and specific.
* Try rephrasing your question or ask it in a different way.
* If the AI still doesn’t understand, it might be a limitation of the AI’s current capabilities.

**The application is slow.**

* Check if your device meets the minimum system requirements for the application.
* Close any unnecessary applications running in the background.
* If the application continues to be slow, it might be due to server issues. Please wait and try again later.

## Frequently asked questions

**Does the Centria AI save my chat history?**

No, the AI does not save your chat history. When you close the application or start a new chat, your previous conversation is not saved.

**Can the Centria AI understand any language?**

The AI is primarily designed to understand and respond in English. However, it may have some capability to understand and respond to other languages.

**Is my conversation with the Centria AI private?**

Yes, your conversations with the AI are private and are not shared with third parties.

**Can the Centria AI learn from my conversations?**

The AI does not have the ability to learn or remember information from individual conversations. It uses pre-existing knowledge and algorithms to generate responses.

**Can I use the Centria AI offline**?

No, the Centria AI requires an internet connection to function as it needs to communicate with the server to generate responses.

**How does the Centria AI generate responses?**

Centria AI uses Open AI’s advanced algorithms and pre-existing knowledge to understand your question and generate a relevant response. It does not learn or remember information from individual conversations.

# RELEASE NOTES AND VERSION HISTORY

## Release notes of each version of Centria AI.

**Version 1.0 Release (April 2024)**

New Features:

* API-key is now stored at a separate json-file instead of being hardcoded.

Bug Fixes:

* Cleared code.

**Version 0.9 Pre-release (March 2024)**

New Features:

* Added ‘Uusi Keskustelu’ button to clear the chat interface and start a new conversation.
* Improved text location on bubbles.

**Version 0.8 Pre-release (February 2024)**

Bug Fixes:

* Fixed a bug where the AI was not waiting user’s messages.
* Fixed a bug where the bubbles left gaps between each other.
* Cleared code

**Version 0.5 Pre-release (February 2024)**

New Features:

* GUI
* Introduced a feature to display user and AI messages as bubbles in the GUI.

Bug Fixes:

* Fixed a bug where the AI was not responding to messages.
* Fixed a bug where the bubbles where not in correct positions.

**Version 0.1 Pre-release (January 2024)**

Initial release of the Centria AI.

New Features:

* Console version.
* Semantic Kernel

Please note: Each new version includes all the bug fixes and improvements from the previous versions. Always update to the latest version for the best experience with the Centria AI.

## Upcoming features

Here are some upcoming potential features and improvements on the development list.

* Users can enter their API-key within the application.
* Voice Recognition: Integration with voice recognition technology to allow users to interact with the chatbot using voice commands.
* Improved Contextual Understanding: Upgrades to the AI’s algorithms to improve its understanding of context, allowing it to provide more accurate and relevant responses.
* Personalization Features: Options for users to customize the chatbot’s behaviour to their liking, such as setting preferred topics of conversation or adjusting the chatbot’s tone.
* Integration with Other Services: The chatbot could be integrated with other services to provide more comprehensive assistance to users.
* Offline Mode: A feature that allows the chatbot to function offline for basic queries could be a valuable addition.
* Rounded corners on all elements

Please note that these are potential improvements and may not reflect the actual plans for the Centria AI’s development.

# Source code REVIEW

Centria AI applications source code can be divided into four parts. First, we have the included libraries and then initialization of the AI which is done by using Microsoft’s Semantic Kernel SDK. Then the chat functions for the user-AI-interface are introduced and lastly the code for the graphical user interface.

Source code explanation is a systematic process that aims to understand and explain the functionality of the code. It involves breaking down complex code structures into simpler, more manageable parts, and explaining what each part does and why it is needed. This process is important for maintaining and improving the codebase.

The documentation will go through the code in a sequential manner, explaining each part in the context of its role in the overall application. It will start with the included libraries, explaining what each library does and why it is included. Then it will move on to the AI initialization, detailing how the AI is set up and prepared for interaction. The chat functions will be explained next, followed by the GUI. Each explanation will include details of what the code does, how it does it, and why it is necessary.

## Libraries and GUI-initialization

The foundation of the Centria AI application lies in the various libraries it includes. These libraries provide a wide range of functionalities and tools that are essential for the operation of the application. They encompass everything from basic programming utilities to advanced AI and machine learning capabilities.

Included libraries:

using CentriaAI.ChatItems; // Internal folder for bubble conf.

using Microsoft.Extensions.Configuration; // Part of .NET-libraries

using Microsoft.SemanticKernel; // Semantic Kernel-library

using Microsoft.SemanticKernel.Connectors.OpenAI; // Connector between Open AI and

Semantic Kernel

using System.Runtime.InteropServices; // .NET-library

CentriaAI.ChatItems is a custom library specific to the CentriaAI application. It contains classes and methods that handle the settings and initialization of the bubbles in the application. Microsoft.Extensions.Configuration is a part of .NET libraries that provides a standard way to access configuration settings from various sources such as JSON files, environment variables. It’s used to handle configuration data of the application.

Microsoft.SemanticKernel is a part of Microsoft’s Semantic Machines technology. It’s used for building conversational AI models. Microsoft.SemanticKernel.Connectors.OpenAI is a library connector between Microsoft’s Semantic Kernel and OpenAI. It allows the CentriaAI to use models from OpenAI for generating responses or understanding user input.

System.Runtime.InteropServices is a .NET library that provides a collection of classes, interfaces, and value types that provide a way to access native functions and APIs of the underlying operating system. It’s used when there is a need to interact with the operating system directly or to use functionality not provided by the .NET framework.

Next, the program initializes rounded corners. the purpose of this code is to make applications external edges rounded instead of rectangular. It starts by importing a function from the Gdi32.dll library to create a rectangular region with rounded corners. The InitializeComponent() method is then called to initialize the form’s components.

[DllImport("Gdi32.dll", EntryPoint = "CreateRoundRectRgn")] // Imports library that rounds corners

private static extern IntPtr CreateRoundRectRgn( // Creates rectangular region

int nLeftRect, int nTopRect, int nRightRect, // Coordinates for corners

int nBottomRect, int nWidthEllipse, int nHeightEllipse); // Coordinates for corners

InitializeComponent(); // Method that initializes form.

this.FormBorderStyle = FormBorderStyle.None; // Set border style to none

Region = System.Drawing.Region.FromHrgn // Creates new region for the form

(CreateRoundRectRgn(0, 0, Width, Height, 20, 20)); // Rounded edges for region

btnClose.FlatStyle = FlatStyle.Flat; // Set Close button to flat

btnClose.FlatAppearance.BorderSize = 0; // Set Close button to borderless

btnSend.FlatStyle = FlatStyle.Flat; // Set Send-button to flat

btnSend.FlatAppearance.BorderSize = 0; // Set Send-button to borderless

The form’s border style is set to None, and a new region with rounded corners is created for the form. This gives the form a borderless appearance with rounded corners. Finally, the styles of the close and send buttons are set to flat and their borders are removed. This results in a clean, modern look for the buttons. This feature is purely aesthetic and doesn’t affect the application’s functionality in any way.

## Semantic Kernel

The next part involves the initialization of the AI. This is achieved using Microsoft’s Semantic Kernel SDK, it provides the necessary infrastructure for building and deploying AI models. The initialization process involves setting up the AI model, loading the necessary data, and preparing the AI for interaction.

The code begins by declaring two private variables, GPT and questionAnswering, named Kernel and KernelFunction respectively. These variables are used to hold the GPT model and the question-answering function that will be created later.

Next, a ConfigurationBuilder object is created. This object is used to build a configuration that can read settings from various sources. In this case, it’s set up to read from a JSON file named secrets.json located in the base directory of the application. The Build method is then called on the builder to create an IConfiguration object. This object represents the built configuration and can be used to retrieve configuration settings.

private Kernel GPT; // Declare an object for the model

private KernelFunction questionAnswering; // Declare an object for AI function

var builder = new ConfigurationBuilder() // Create a configuration builder

.SetBasePath(AppDomain.CurrentDomain.BaseDirectory) // Set directory path to .json file

.AddJsonFile("secrets.json"); // Add .json file to config

IConfiguration configuration = builder.Build(); // Build the configuration

Two configuration settings are retrieved from the IConfiguration object: OpenAIChatCompletion:ModelName and OpenAIChatCompletion:ApiKey. These settings are stored in the modelName and apiKey variables respectively. They represent the name of the OpenAI model to be used and the API key for accessing OpenAI’s services.

string modelName = configuration["OpenAIChatCompletion:ModelName"]; // Get AI-model name from .json

string apiKey = configuration["OpenAIChatCompletion:ApiKey"]; // Get API-key from .json

A KernelBuilder object is created using the Kernel.CreateBuilder method. This builder is used to construct a Kernel object, which represents the GPT model. The AddOpenAIChatCompletion method is called on the builder to add the OpenAI Chat Completion model to the kernel. The model name and API key retrieved earlier are passed as arguments to this method. Finally, the Build method is called on the builder to create the Kernel object. This object is stored in the GPT variable declared earlier.

var kernelBuilder = Kernel.CreateBuilder(); // Create Kernel builder

kernelBuilder.AddOpenAIChatCompletion(modelName, apiKey); // Add OpenAI model to builder

GPT = kernelBuilder.Build(); // Build the Kernel

A prompt for the question-answering function is defined as Answer the following question: {{$question}}. This prompt is used to create a KernelFunction object, which represents the question-answering function.

The CreateFunctionFromPrompt method is called on the Kernel object to create the function. The prompt and an OpenAIPromptExecutionSettings object are passed as arguments to this method. The settings object specifies that the maximum number of tokens for the function’s output should be 200. The created KernelFunction object is stored in the questionAnswering variable declared earlier. This completes the setup of the GPT model and the question-answering function.

var questionAnsweringPrompt

= @"Answer the following question: {{$question}}"; // Define AI prompt

questionAnswering = GPT.CreateFunctionFromPrompt

(questionAnsweringPrompt, executionSettings:

new OpenAIPromptExecutionSettings { MaxTokens = 200 }); // Create function from the prompt

The code initializes the GPT model and a question answering function using Microsoft’s Semantic Kernel SDK and the OpenAI API. It also reads the model name and API key from a secrets.json file. Which is presented here:

"OpenAIChatCompletion": { // Namespace

"ModelName": "gpt-3.5-turbo", // Defines the AI model name

"ApiKey": "**Api key is inserted here**"} // User’s API-key is inserted here

Now for the processing the AI’s answer, this code takes a question, passes it to the GPT model, processes the response, and displays it in the GUI.

**async Task AI(string** Question\_TBA**) // Async-method for saving AI answer**

{

var vastaus = await GPT.InvokeAsync(questionAnswering, new() // Invokes the GPT model asynchro-

nously with 'questionAnswering'

and a new object.

{ ["question"] = Question\_TBA });

string vastausString = vastaus.ToString(); // Converts the response to a string

Add\_AiMessage(vastausString); // Adds Ai answer to GUI

}

The code begins with an asynchronous invocation of the GPT model. The InvokeAsync method is called on the GPT object, which represents the GPT model. This method takes two arguments: the questionAnswering function and a new object.

The questionAnswering function is the question-answering function that was created earlier. It’s designed to take a question and return an answer generated by the GPT model. The new object is an anonymous object with a single property named question. The value of this property is set to Question\_TBA, which is a variable holding the question to be answered.

Once the GPT model returns a response, it’s stored in the vastaus variable. This response is an object that represents the answer generated by the GPT model. The ToString method is then called on the vastaus object to convert the response to a string. This string is stored in the vastausString variable.

Finally, the Add\_AiMessage method is called with vastausString as an argument. This method is responsible for adding the AI’s response to the GUI. It ensures that the response is displayed in the chat interface, allowing the user to see the AI’s answer to their question.

## Chat Functions

Once the AI is initialized, the chat functions are introduced. These functions handle the interaction between the user and the AI. They manage everything from receiving the user's input, processing it, generating the AI's response, and sending it back to the user.

When the app is launched, a welcome message from AI is displayed. The message itself is a static string and AI is not used.

Kuva, joka sisältää kohteen kuvakaappaus, teksti, Fontti, logo

Kuvaus luotu automaattisesti

FIGURE 4: Welcome message on GUI.

To achieve this a function named AddListMessage calls Add\_AiMessage which then adds included string to a bubble and shows it on GUI.

void AddListMessage() { Add\_AiMessage("Welcome to Centria AI"); } // Method for a message from AI

The following code adds user’s message to a bubble and shows it on GUI. Function calls the Add\_UserMessage-method, which is explained on Calling bubbles-section.

void AddUserMessage() { Add\_UserMessage(txtMessage.Text); } // Method adding user input to GUI

## Button Clicks

When the send button is clicked, the AddUserMessage method is called. This method adds the user’s message to the chat interface. Next, the AI method is called with the text of the user’s message as an argument. This method processes the user’s message and generates a response in async Task AI. Finally, the text of the message input box represented by txtMessage is cleared. This prepares the input box for the user’s next message.

**private void btnSend\_Click(object sender, EventArgs e) // Method for sending a message**

{

AddUserMessage(); // Adds message to GUI

AI(txtMessage.Text); // Calls AI

txtMessage.Clear(); // Clears the input box

}

To shut down the app a close button is needed. When the button is clicked, the Close method is called and closes the app.

**private void button1\_Click(object sender, EventArgs e) // Method for closing the app**

{

this.Close(); // Closes the app

}

To remove previous messages and to start a new chat without re-starting the app a simple code was needed. The reset\_Click method in tied to a reset button named “Uusi keskustelu” in the app. The purpose of this functionality is to clear the current state of the app and return it to its initial state.

**private void reset\_Click(object sender, EventArgs e) // Method for starting a new chat**

{

this.Controls.Clear(); // Empties the form

this.InitializeComponent(); // Re-initializes the form

AddListMessage(); // Adds the welcome message

}

When the reset button is clicked, the application first removes all the elements (like buttons, labels, textboxes, etc.) that are currently on the form. Next, the application re-initializes the form to its initial state as defined in the InitializeComponent-method. This method returns the GUI to its initial state. Finally, the AddListMessage method is called. This method adds the welcome message to the screen.

## Message Bubbles

The final part of the source code is dedicated to the graphical user interface. The GUI is the visual part of the application that the user interacts with. It includes elements like text boxes for input, buttons for sending messages and starting new chats, and areas for displaying the conversation. To implement chat function to GUI a bubble system was needed.

Kuva, joka sisältää kohteen kuvakaappaus, saha, työkalu

Kuvaus luotu automaattisesti

Kuva, joka sisältää kohteen kuvakaappaus, teksti, Multimediaohjelmisto, ohjelmisto

Kuvaus luotu automaattisesti

FIGURE 5: The GUI of the user and AI message bubbles.

## Calling Bubbles

In the main code in Form1.cs the bubbles are called. The method named BubbleSettings takes two parameters, a UserControl object that represents the bubble and a string that represents the message. This method adds the bubble to app background named pnlContainer. It then sets the Dock property of the bubble to Top, which means the bubble will align with the top of the pnlContainer. It ensures that the latest chat bubble is always visible and not obscured by any other control by bringing the bubble to the front. It also sets the width of the bubble to be twenty pixels less than the width of the pnlContainer. Finally, it sets the Message property of the bubble to the input message.

The Add\_UserMessage method creates a new UserMessage object and calls the BubbleSettings method to set up the bubble for the user’s message. Similarly, the Add\_AiMessage method creates a new AiMessage object and calls the BubbleSettings method to set up the bubble for the AI’s message. The BubbleSettings method is used to avoid code duplication when setting up the properties of the message bubbles. The Add\_UserMessage and Add\_AiMessage methods are used to add messages from the user and the AI, respectively.

**void BubbleSettings(UserControl bubble,string message)** **// Method for bubbles’ settings**

{

pnlContainer.Controls.Add(bubble); // Adds the bubble to the app

bubble.Dock = DockStyle.Top; // Docks the bubble to top

bubble.BringToFront(); // Bubble brought to front

bubble.Width = pnlContainer.Width - 20; // Bubble width 20pix<app width

((dynamic)bubble).Message = message; // Message converted to text

}

**void Add\_UserMessage(string message)** **// A unique method for Users message**

{

BubbleSettings(new UserMessage(), message); // Calls BubbleSettings-method

}

**void Add\_AiMessage(string message) // A unique method for AI message**

{

BubbleSettings(new AiMessage(), message); // Calls BubbleSettings-method

}

## Bubble’s construction

The actual bubbles are constructed in separate files, which are named Utils.cs, UserMessage.cs and AiMessage. The bubble has four components: The bubble’s background named “UserMessage”, text’s background named “outpanel1”, text label named “label1” and finally the image named “outpic”. Here is the code for UserMessage.cs file:

public partial class UserMessage : UserControl

{

public UserMessage() { InitializeComponent(); } // Constructor init’s the component.

**public string Message** // String -> user message is saved

{

get { return label1.Text; } // Gets the text that user wrote

set { label1.Text = value; AdjustHeight(); } // Sets the text in the bubble and

} calls AdjustHeight-method

**void AdjustHeight()** **// Method that adjusts bubble height**

{

label1.Height = Utils.GetTextHeight(label1) + 10; // Adjusts the text’s height

outpanel1.Height = label1.Top + outpanel1.Top + label1.Height; // Adjusts the text’s base height

this.Height = outpanel1.Bottom + 10; // Adjusts bubble’s total height

}

public Image Avatar // Property for User’s image

{get => outpic.Image; set => outpic.Image = value; } // Gets and sets the image

**private void Incomming\_Resize(object sender, EventArgs e) // Method for resizing the bubble**

{ AdjustHeight(); } // Calls AdjustHeight-method

**private void Outgoing\_DockChanged(object sender, EventArgs e) // Method for whoknowswhat**

{ AdjustHeight(); } // Calls AdjustHeight-method

}

Firstly, the code makes a property where the text that user has written is saved, then code will get the height of the text itself from Utils.cs. Then it will accordingly adjust the height of the other elements in the bubble based on the text height. Finally, it will resize the default bubble size to proper height. This code with taking account the renaming of the properties is identical to the AiMessage.cs. Utils.cs is needed to calculate the text height. Here is the code for that:

**public static int GetTextHeight(Label lbl) // Method for calculating the height**

{

using ( Graphics g=lbl.CreateGraphics() )

{

SizeF size = g.MeasureString(lbl.Text, lbl.Font, 495); // Measures the text size and wraps

it, if it exceeds 495 char.

return (int)Math.Ceiling(size.Height); // Rounds the height to nearest int

} and returns value to message.cs

}

This method uses the Graphics.MeasureString method to measure the size of the text in the Label control. It considers the font of the text and a specified width for the text. If the text exceeds this width, it will wrap to a new line, and the height of the text will increase accordingly. The height is rounded up to the nearest integer using Math.Ceiling to ensure it is an integer and that it fully accommodates the text.

# development process

The first version of the CentriaAI was a simple CMD application where we tested the AI’s capabilities of answering questions and translating texts. This version was mostly used as a groundwork for building the future graphical interface with AI.

Kuva, joka sisältää kohteen teksti, Fontti, kuvakaappaus

Kuvaus luotu automaattisesti

FIGURE 6: CMD version of the AI

## The early graphical interface

The early graphical interface had a similar functionality to the CMD version. It contained a list of 6 different prompts from which the user could choose (Figure 7), and the AI would be called to perform that specific function. The code was built with a SWITCH-CASE working inside an IF-ELSE structure. The functionality and GUI were basically identical to the final version, but the code was 3 times longer. The prompt list was removed after it was discovered that any prompt could achieve other functionalities too. For instance, the translate prompt could write summaries or code as well.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Fontti

Kuvaus luotu automaattisesti

FIGURE 7: List of prompts.

Problems with this version included problems with the chat bubbles being printed on top of each other, the distance between the bubbles growing as the bubbles reached the end of the GUI (Figure 8), and problems with the AI not being able to understand the questions in one of the SWITCH-CASES.

Kuva, joka sisältää kohteen teksti, kuvakaappaus, Matkapuhelin, Mobiililaite

Kuvaus luotu automaattisesti

FIGURE 8: The problem with spacing of bubbles.

## Newer versions of the AI

We ended removing the IF-ELSE and SWITCH-CASE structures from the code and simplifying the AI to treat all user inputs as questions to be answered. The bubble creation was changed to dynamically anchor each new bubble to the previous one instead of calculating the height of the previous bubble and placing the next one underneath.

We also tried to include memory of the conversation to the AI with Semantic Memory. The vision was to have a chat AI which could remember your conversation and provide specifications to previously answered questions if necessary. We eventually gave up on the memory after its implementation made the code unable to compile. This is something for future development.

The final implementation was the clear chat button ‘Uusi Keskustelu’, which clears the chat and starts a fresh one. This was planned together with the memory and currently serves mostly a visual purpose rather than a functional one.

## API Keys and documentation

In the earlier versions our API keys were simple strings with which the AI was constructed and called with. This was changed and the new versions stored the key and AI model information inside a secret.json file. This file would be included in the GitIgnore and not be shown in the repository.

During the development process, we wanted to try GitBook for documenting our code. The way we decided to implement the GitBook was to create a new public repository of our code where we have cleaned our API key information from the code. After the creation of the public repository our API keys were leaked. This was because through version history it was still possible to access the previously overwritten parts of the code where the keys were stored.

## Exporting the app

As this application uses several external libraries, it was essential that the finished application could be run in any Windows device without needing installing any third-party software. This was achieved by changing Visual Studio’s publish settings. Deployment mode was set to “Self-Contained” and target runtime was set to "win-x86”. Finally selecting the option to produce single file. These settings allow the exported application to run in both 32- and 64-bit Windows and the whole application is contained in only 10 files, with two of them being the .exe and .json files. Application size is approximately 150 megabytes.