Project Plan

Team 5D - Bug Busters

Contents

1 Problem Analysis

1.1 Problem Statement

Context: Insocial, our client, offers comprehensive survey analysis to various businesses based on selected topics. To collect essential data, like employee contact emails for survey distribution, Insocial requires specific data points or access to endpoints for data extraction. This process necessitates technical proficiency on the client's side and a specialist from Insocial to synchronize the two APIs.

Challenge: With a limited team of developers, Insocial finds it challenging to commit resources to manual API integration, especially when the process is potentially automatable, facilitating a non-technical user to handle the integration. Consequently, we've been assigned to develop a user-friendly platform that enables a non-technical individual to link their company to Insocial's API, visualizing the connections through an intuitive graph-like structure.

1.2 Stakeholders

Stakeholders for this project encompass the Insocial platform, companies using the platform (Insocial customers), our development team, and developers accountable for future development and maintenance of our product.

1.3 Products use cases

The product's core use case is connecting different systems to Insocial for automated invite sending and enabling users to build a flow that connects their system with Insocial by performing actions such as GET, POST, PUT on any given endpoint. Our product will eventually be integrated into the Insocial platform, so these connections will be created by one of Insocial's customers who will select the data sources and the functionalities to process these data to reach some end goal.

A concrete example use case that we have sketched out in consultation with our client is given below:

As the user who uses VISMA HRM system I want to automatically send out invites to the employees who just left the company to ask them about their experience as an employee of mine. Via the flow builder I must be able to do the following:

1. Set up as schedule or a trigger that starts the execution of the flow (For example each day at 09:00 AM GMT +1)
2. Add a GET endpoint with ClientID or ClientSecret as parameters of the request (storing the credentials should be safe and not accessible by any user not connected to the customer)
3. Add optional parameters or headers to the request for example TenantID can be provided as a request header
4. The flow builder should recognize and use data that comes from the requests in earlier steps of execution to use in the next step.

5. Create a condition using previously fetched data such as "Only use the data for the invite if the field {{contractEndDate}} of the employee is equal to todays date"
6. Using the filtered data from the GET request using {{email}} field for recipient and including fields {{example1...}} as metadata send out invite using provided email template with a link to selected survey, and sender name.
7. I must be able to save and enable or disable the flow, optionally save it as draft
8. I could be able to test the flow without actually sending the invites to verify if the data fetched using provided parameters is valid.

1.4 Existing similar solutions

One of the already existing products is Boomi, which provides a platform for software integration. Many of the features it provides are similar to the ones we will be building: Boomi has a flow builder which supports conditions on whether the flow should continue or fail, error handling and many various ways to set up, filter, and request data from the customer's sources. We could take inspiration from the UI itself as it is understandable and clean, expanding information only when a specific object in the flow is selected. Furthermore the structure for connecting endpoints could hint a possible backend structure and flow that we might incorporate.

The main difference between Boomi and the product we will be building is that Boomi is a generic iPaaS (integration platform as a service) solution that isn't specifically tailored to the needs of Insocial. A crucial requirement in our project is seamless integration with the Insocial platform and its existing MySQL database, which requires a more customised solution. Additionally, Boomi is a proprietary product, so incorporating it may incur licensing costs and potential limitations on customization.

Other existing products that perform similar functions include Zapier and Integromat. We can learn from their interfaces and robustness of API integrations, but they are not tailored to the specific needs of Insocial, so we cannot incorporate them into our project. That is because these products are complex as they solve a general problem, while our client only requires integration with their specific API and not all miscellaneous ones such as Gmail, WhatsApp or Slack. Our application will focus on solving a more narrow problem effectively and in a way which can be incorporated into the broader Insocial system.

1.5 Discussions with stakeholders

Our client, being the provider of the Insocial platform, has an intimate understanding of their customers' needs and has been able to provide us with extensive information about the challenges they face and the solutions they are looking for. They are our primary point of contact and we trust in their expertise to guide us in developing a product that will meet the needs of their users. In particular, we are in active discussion with one of Insocial's software engineers and project manager, which gives us insight into different aspects of our client's needs.

2 Feasibility Study & Risk analysis

We are sure that we have enough time to deliver a stable working product which satisfies the minimum completion requirements and tackle some extra ones. We think the project is challenging as we have some more requirements that definitely would be useful to have but are time consuming to implement at the same time. For example an in depth error stack for flows debugging purposes. As for the complexity part of the project, there will be specific conditions and triggers that will certainly make it more difficult to process the requests in the flow. The graphical representation of the graph might also be a challenge as there are a lot of variables regarding the usability of the product. However, we have already encountered those problems in planning our project – and we know that with enough research and considerations we can manage those challenges.

2.1 Technologies and frameworks:

For our project, the choice of technologies and frameworks is mostly beyond our control due to our client's requirement for easy integration with their existing platform. This means that we will use PHP with the Symfony framework for the backend, Typescript with Angular for the frontend, and MySQL for our database. Below we discuss the advantages and disadvantages of these technologies, alternatives we might have considered, and other tools we will be using.

Docker

Although we didn't get to choose the languages and frameworks for this project, we do have control over how we manage dependencies and project deployment. To this end, we decided to use Docker, an open platform which can be used for building, distributing, and running applications in a portable, lightweight runtime and packaging tool known as Docker Engine [1]. This ensures a consistent environment for our application, meaning it will be possible to build it and run it on other machines regardless of what software and tools they provide. While alternatives to Docker exist, Docker containers are the de-facto industry standard [2] and we believe that the speed, portability and scalability that this tool provides make it a good choice for our product.

PhpStorm and Webstorm

To ensure a uniform development environment and ease of code sharing and collaboration, our team will use PhpStorm and Webstorm as the Integrated Development Environment (IDE). Using the same IDEs reduces the overhead of setting up different environments and allows for easy collaboration and assistance among team members.

PHP and Symfony

We believe that PHP and Symfony are not the best choices for our given use case. PHP is a flexible language, but it is also known for being difficult to work with in larger projects, with one study calling debugging and refactoring PHP code "a nightmare" [3]. Additionally, PHP scores poorly on developer satisfaction [4] and in recent years its popularity has been falling [5], meaning it could be hard to finding developers willing to maintain a PHP tool in the future. As Symfony is a framework for PHP, it inherits these problems.

Nonetheless, while not being the best choice, we believe these tools will provide us the needed tool kit for building the back-end for the project. The potential risks of working in these technologies, as opposed to in more modern alternatives, are mitigated by the fact that two developers on our team have prior professional experience in these technologies.

Additionally, we have spent time onboarding the other team-members to get them familiar with the PHP and Symfony workflows and potential issues they might encounter, which should help the development process proceed smoothly.

Typescript and Angular

The front-end of our application will be built using TypeScript and Angular. We use these tools as this is required for easy integration with our client's existing systems, but our research suggests that these tools are also simply a good choice for our application. Below we summarize our conclusions about these two tools.

Typescript

Typescript is a natural choice for us as our developers have experience with typed languages with some object oriented characteristics, making the use of TypeScript preferable over JavaScript, which is not statically typed. TypeScript is a compiled language that builds on top of JavaScript, which makes it appropriate for building a web application. It was designed to enable easier development of large-scale projects and adds optional types to JavaScript, with support for integration with existing JavaScript libraries via interface declarations [6].

Angular

Angular, as a framework, boasts a substantial and active user community that consistently contributes new open-source libraries, pre-made components, and endeavors to enhance the framework itself. Utilizing the Node Package Manager, developers can employ pre-tested solutions for routine tasks, such as crafting a user interface (UI) kit, thereby enabling them to focus on customizing it to suit specific requirements. Employing Angular, or any comparable framework, streamlines the development process by allowing developers to concentrate on higher-level functionalities while delegating lower-level scope concerns (e.g., JQuery).

Furthermore, Angular excels in the context of large-scale applications requiring straightforward extensibility and scalability when performance is not the foremost priority. This distinction is evidenced by several criteria, including Data Binding (the process of connecting business logic data with the front-end UI), Templating (dividing the view into smaller segments and displaying them as needed), Extensibility (augmenting the existing HTML capabilities through embedded tags representing additional functionalities), Variable Observation (monitoring changes in variable properties), Routing (storing application states linked to views), Testing (incorporating testing capabilities), and Advanced Features (such as Dependency Injection and Form Validation) [7].

2.2 Plan in case of infeasibility

Should the execution of the current project prove infeasible, one potential strategy would involve simplifying the interface of our solution. For instance, the current graphical-like structure could be transformed into a more straightforward user interface, such as a list that enumerates nodes and their respective connections. Moreover, on the backend, it could be feasible to either eliminate debugging functionality or restrict the integration with external APIs.

2.3 Risk Analysis:

The primary risk associated with our solution pertains to the management of confidential user data (e.g., passwords or API tokens) and the thorough testing of our solution to prevent unintended outcomes (such as sending multiple emails instead of a single one). Furthermore, we wish to reassure our client that we do not retain any actual data procured from requests to external APIs during the execution of a flow. While these risks do not pose an immediate threat, they could potentially tarnish the reputation of our client. We appreciate that our client is often accessible for immediate queries, given our shared working environment during office hours. This accessibility extends to Angular and Symfony experts, who are available to assist should we encounter any coding hurdles. As for data-related issues, the client possesses a significant volume of mock data for testing, which we can utilize during our implementation.

To minimize the risks of encountering difficulties, we aim to clarify most of the details with the client regarding their precise expectations in the backlog. We will present them with our conceptualized designs before initiating their implementation. We have already engaged in discussions surrounding safety and risk concerns, reaching the following consensus:

1. We will not store any data obtained from executing a request to any specified endpoint beyond the scope of the flow execution.
2. We will utilize the client's existing authentication system during the development process, integrating it into the current infrastructure upon project completion.
3. While we may need to store passwords or logins to execute the flow, we will strongly discourage this practice and instead encourage users to provide API keys.
4. Due to the existing authentication and user-customer connection, no user will be able to access the flows and data of customers to whom they are not assigned.

3 Requirements

3.1 Introduction

The following section details the essential features and functionalities that the system must possess, along with desirable and optional components. These requirements were derived through a collaborative and iterative process involving in-depth discussions with the client, thorough analysis of the project's goals, stakeholders' needs, and the technical constraints of the Insocial platform. This approach allowed us to refine and prioritize the requirements, ensuring they accurately represent the client's vision and expectations. The requirements have been divided into four categories: must haves, should haves, could haves, and won't haves. Additionally, this section includes a list of non-functional requirements to ensure the system's performance, compatibility, and usability align with the project's objectives.

3.2 Must haves:

1. The system must have an intuitive drag-and-drop interface for connecting systems with Insocial by creating control flow graphs using nodes translated from company endpoints, which is easy for non-technical users to navigate.
2. The user must be able to edit, disable, or enable created flows.
3. The system must support graphical representation of API connections.

4. The system must allow users to define nodes that execute arbitrary API calls in the flow builder.
5. The system must enable users to specify the endpoint for each node representing an API call. 6 The system must enable users to specify the request method (e.g., GET, POST, PUT) for each node representing an API call.
6. The system must enable users to specify the request body as needed for each node representing an API call, provide arguements with which the endpoint will beexecuted such as headers, authenication method
7. The system must enable nodes in the flow builder to perform GET, POST, PUT actions on any given endpoint and trigger actions based on input received from API calls, facilitating the integration of external data sources in the flow.
8. The system must provide support for logic flow and control, including triggers, conditions, schedules, and possibly loops.
9. The system must provide support for conditional branching, allowing different actions to be taken based on data and conditions within the flow.
10. The system must provide support for scheduling flows to run at specific times or intervals.
11. The system must execute the flow using the schedule specified by the user.
12. The system must create user profiles that store and update graphs in the database.
13. The user must be able to execute the flow manually.
14. The user must be able to view the flows they've created.
15. The user must be able to select a previously created flow to edit it.
16. The user must be able to save the changes they've made to the flow they're editing.

3.3 Should haves:

1. The system should have an error handling system that allows users to see what and where in the flow a problem occurred.
2. The system should support a "testing functionality" that enables users to execute the flow and check control flow outcomes and endpoint responses without making actual changes, ensuring that the flow works as expected without errors.
3. The system should match the design style used in Insocial, following the brand book for components and colours.
4. The system should be able to reuse nodes for multiple graphs created by the user.
5. The system should provide a pre-built template for a node specifically designed to handle Mail API calls.
6. The system should be able to save the structure of an existing flow as a template for future reuse.
7. The users should be able to save a node with its API parameters as a template.
8. The user should be able to reuse the nodes saved as a template.

3.4 Could haves:

1. The system could support debugging functionality, executing the flow step by step.
2. The system could support XML data request formats in the body.
3. The system could allow users to test API responses to check data format.
4. The graphs could have an auto-save feature based on a certain time interval.
5. The system could provide support for loops, enabling repetitive actions within the

flow based on conditions or a fixed number of iterations.
6. The system could store logs of failed flow executions which the user can retrieve.

3.5 Won't haves:

1. The system won't create it's own user authentication method.
2. The system won't provide a comprehensive tutorial targeted at non-technical users (as the focus is on developing an intuitive and user-friendly interface that minimizes the need for extensive guidance).
3. The system won't support a tool to adjust edges of the graph. Edges will be auto generated based on position of the nodes.
4. The system won't support hardcoded API connections other than the one from Insocial.
5. The system won't support possibility to import CSV representation of API connection.
6. The system won't support triggers to API actions such as CRUD changes such as deletion/creation of survey response, as this would require us to change already existing client backend codebase.

3.6 Non-functional requirements

1. The backend must be implemented using PHP 8.
2. The database must use MySQL.
3. The frontend must be implemented using Angular CLI with TypeScript.
4. The product must be well documented, including source code, README files, and information about dependencies, build processes, APIs, and other relevant information.
5. The system must integrate with the Insocial platform and its existing MySQL database.
6. The system must enable easy integration with existing authentication of Insocial platform after the development process is finished and use hardcoded bearer tokens during the development process.
7. The system must support the JSON data format.
8. The system should have the ability to be translated into Dutch in later stages of development.

4 Project approach

We are going to be implementing an application that relies on both back-end and front-end. The front-end will be implemented with Angular as a Single Page Application. Here the user will be able to see a web page that follows Insocial's design book and contains a graphing field. In the said graphing field the user will be able to drag and drop nodes that should perform some task like a GET request for fetching data or an IF condition block for continuing the flow. The list of all possible tasks can be found in the MOSCOW requirements. The nodes will be able to be connected with edges that would pass the result of the node to the input of another one thus building a flow. We have decided to leave creation of the graph entirely on the front-end so the user could easily and intuitively customise it the way they like with a simple interface.

While the building of the flow is done on the web page, the execution will be done on the back-end that makes use of principles that are in REST. The graph will be run in a topological order with details specified by the user, like the endpoints and their authorisation tokens. When the flow is run, the server is going to fetch the graph and then execute the node with the information provided on the front-end. Then the result will be passed further down the flow to be, for example, filtered or saved somewhere as specified by the next node in a topological manner. We opted for this because we wanted all of the complex processing to be done on a single side so that it would be easier to test and implement.

We also brainstormed the design and researched the possibilities of implementing this. Below is our analysis for some potential aspects of the problem.

4.1 UI

For the User Interface we will follow the Insocial style of having a navigation menu on the left and a header at the top. The rest of the space is going to be used for our tool. There will be an option to create a new graph and select/edit old graphs. When in edit mode we will be able to drag and drop to create a new kind of node (e.g. a GET or POST node or an Email node, terminal node, condition nodes) and use pre-saved nodes from the tool box. When we click on a node we can edit its fields, e.g. the endpoint and body in a menu that latches to the right side of the graph editing screen.

4.2 Graph storage

One of the requirements for the project is for the user to be able to save their graph with the flow for scheduled execution. First we thought of having an edge matrix with a node table to store it more efficiently but then ran into an issue of reconstructing it for the client to see. Also to avoid overhead while fetching the data from the graph we have decided to store the edges within the nodes rather than in separate table. In the end we decided to go with a simple JSON approach to store the entire graph in order to save time as we only have a little over 7 weeks of coding.

4.3 CI/CD

We will be setting up a typical software development pipeline which will include building our project, running tests, and checking code formatting through Checkstyle.

4.4 Authentication and security

We ran into the trouble of each user having a unique token from Insocial. For the project duration we agreed with the client to set up a functionality to support adding different tokens while at the same time using only one admin token for ease of development and testing. There is also a problem with the user given tokens expiring in the future. We have settled on the approach of passing this to the error handling specifying when the authentication in the customer API side goes wrong that the token may have expired.

4.5 Graph saving

At first we opted for manual saving done by the user, but the client mentioned that it would be nice to have an auto-save feature. We are keen on the timed approach set on a certain interval but this may change in the future.

4.6 Graph design

Because our tool is made for people with low technical knowledge, an important part of our solution is to make the tool usable and easy to understand. Therefore, we have decided to follow some of the principles mentioned in Midway's (2020) "Principles of Effective Data Visualization" [8]. In this research paper, the author mentions 10 important rules to effectively visualize the data. We have decided to focus on the following points:

1. "*Diagram first [. . .] don't necessarily think of the geometries (dots, lines) you will eventually use, but think about the core information that needs to be conveyed [. . .]"*.
2. *"Use the right software"* the implemented solution should be adjusted to users needs – in our case we want to simplify the process of graph making as much as possible and therefore our graph making tool also should not be complex.
3. *"Use an Effective Geometry and Show Data [. . .] Colors Always Mean Something"* – a key concept that can make the graph usable is a proper usage of colours and shapes while designing the graph. They should be simple and limited to a few core concepts, and also represent some meaning.
4. *"Simple Visuals, Detailed Captions"* – according to this principle the end solution should result in graphs that are visually simple and not cluttered, and yet well explained.
5. *"Get an Opinion – Although there may be principles and theories about effective data visualization, the reality is that the most effective visuals are the ones with which readers connect"*. In other words, the solution should be tested and reviewed for usability throught the process of development.

5 Development methodology

5.1 Development approach

We are going to use an Agile development process. That means that even though we do our best to prepare as well as possible for each of the the steps (requirements gathering, analysis and design, development and implementation, testing and verification, deployment and ongoing maintenance), we want to stay flexible and maintain a possibility to adjust our concepts and solutions throughout the development of the project.

Moreover, a student team, in a situation similar to ours with the lack of specific domain and framework knowledge succeeded using this same approach [9]:

> The team observed that the continuous learning competency is especially reinforced through the iterative development of Scrum. Team members described revising and improving design and implementation decisions throughout the project as they experimented and learned more.

5.2 Acceptance requirements

We have agreed to only accept new functionalities, if they are reasonable and easy to incorporate into our solution. As a team we would like to do our best to assure that we meet our client's expectations to the best of our ability, however for the sake of quality of the solution, we will not agree to incorporate unreasonably large changes during the development of the project. We agreed, that the recognized functionalities fulfil the core requirements, and further improvements (if any) should be discussed when they occur.

5.3 Definition of done

During our talk with the client we derived a following definitions of done:

1. Drag-and-drop interface to connect systems with Insocial:
   a. The drag-and-drop interface is implemented and functional.
   b. Systems can be easily connected to the Insocial platform using the interface.
2. Graphical representation of API connections:
   a. The API connections are visually represented in a form of a graph.
   b. Users can easily create, understand and interpret the connections through the graphical representation.
3. Support for various types of API connections:
   a. Users can configure and establish connections with APIs of various types.
4. User authentication and security measures:
   a. User authentication is implemented, allowing only authorized users to access and use the solution the data assigned to them.
   b. Appropriate security measures are in place to protect user data and API credentials.
5. Integration with the Insocial platform:
   a. The solution is seamlessly integrated with the Insocial platform.
   b. Data and actions can be exchanged between the solution and the Insocial platform.
6. Ability to perform GET, POST, PUT actions on any given endpoint:
   a. Users can execute GET, POST, and PUT actions on any desired endpoint.
   b. The solution communicates with APIs and performs the specified actions accurately.
7. Trigger actions based on input received from API calls:
   a. The solution is capable of receiving input from API calls.
   b. Based on the input received, the solution triggers the specified actions effectively.

5.4 Team communication

As specified in our code of conduct:

> We will treat each other with respect and professionalism. We will address disagreements constructively and seek consensus. Our coach and TA could be involved in reaching consent if needed, but this will be used as a last resort. If someone is late during a group meeting, they will notify the group in advance and catch up on missed information. We will handle conflicts within the group through open communication, understanding, and seeking a resolution that benefits the entire group.

6 References

[1] Bashari Rad, B., Bhatti, H., & Ahmadi, M. (2017). An introduction to Docker and analysis of its performance. IJCSNS International Journal of Computer Science and Network Security, 173, 8.

[2] Wu, Y., Zhang, Y., Wang, T., & Wang, H. (2020). An empirical study of build failures in the Docker context. In Proceedings of the 17th International Conference on Mining Software Repositories (MSR '20). ACM. https://doi.org/10.1145/3379597.3387483

[3] Alomari, Z., Halimi, O., Sivaprasad, K., & Pandit, C. (2015). Comparative studies of six programming languages.

[4] Stack Overflow. (2022). Stack Overflow developer survey 2022 results. https://survey.stackoverflow.co/2022/

[5] GitHub. (2022). The state of open source on GitHub: 2022 report. https://octoverse.github.com/

[6] Bhattacharyya, S., & Nath, A. (2007). Application of TypeScript language: A brief overview. International Journal of Innovative Research in Computer and Communication Engineering, 4, 10585-10590.

[7] Sultan, M. (2017). Angular and the trending frameworks of mobile and web-based platform technologies: A comparative analysis. In Future Technologies Conference (FTC) 2017.

[8] Midway, S. R. (2020). Principles of effective data visualization. Patterns, 1(9), 100141. https://doi.org/10.1016/j.patter.2020.100141

[9] Rover, D., Ullerich, C., Scheel, R., Wegter, J., & Whipple, C. (2014). Advantages of agile methodologies for software and product development in a capstone design project. In 2014 IEEE Frontiers in Education Conference (FIE) Proceedings. IEEE. https://doi.org/10.1109/fie.2014.7044380