P5-Software-project Home

Project Overview:

Welcome to the landing page for our Telegram Bot project! This page serves as the central hub for information and updates related to our work-in-progress Telegram Bot project. Here, you will find key project details, version control information, and other relevant resources to help you stay informed about our progress and contribute to the project's success.

Project Description:

Our Telegram Bot project aims to create a versatile and user-friendly bot for the Telegram messaging platform. The bot is designed to let students search for jobs in their field that are posted on the job teaser website. The app lowers the threshold of applying for a job because it uses a popular messenger app that most people already have installed on their phones..

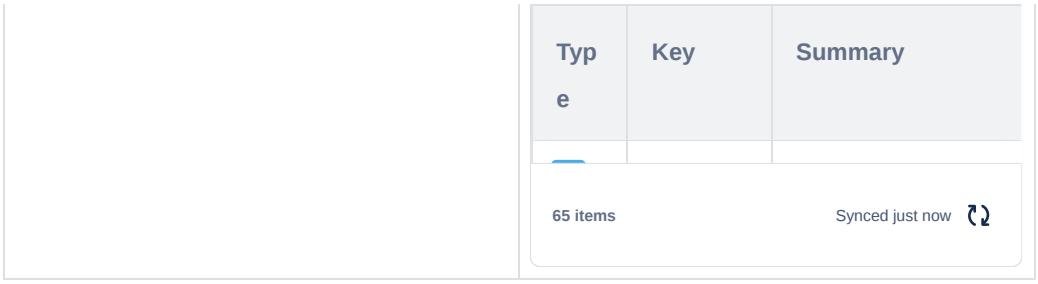
Key Features:

- Job Search Feature (Users might perform thorough job searches utilizing a number of advanced search filters)
- Job Alert Feature (Enable users to set up personalized job alerts based on their preferences. Users can define the following parameters)

Version Control:

To keep track of our project's development and changes, we are using version control. Here's a table summarizing our version control system:

<table>
<tr><th>Version Control System</th><th>GIT</th></tr>
<tr><td>Repository URL</td><td>https://gitlab.tamk.cloud/sw-project-group-5/group-5-telegram-bot</td></tr>
<tr><td>Branches</td><td>main
https://gitlab.tamk.cloud/sw-project-group-5/group-5-telegram-bot/-/tree/main?ref_type=heads</td></tr>
<tr><td>Issue Tracker</td><td>
<table>
<tr><th>Type</th><th>Key</th><th>Summary</th></tr>
<tr><td></td><td>PSP2-66</td><td>Take backup of eve</td></tr>
<tr><td></td><td>PSP2-65</td><td>Send code to team</td></tr>
<tr><td></td><td>PSP2-64</td><td>Send documents a</td></tr>
<tr><td></td><td>PSP2-63</td><td>Git readme should</td></tr>
<tr><td></td><td>PSP2-62</td><td>Git readme is upda</td></tr>
</table>
</td></tr>
</table>



Project Team:

- Anmol Arora - Project Manager
- Sviatoslav Vasev - Developer - UI
- Ozan Topcu - Developer - Backend
- Jennifer Hensley - Developer - UI, Documentation
- Toan Tran - Developer - Tester
- Sudikshya Bhattarai - Developer - Support

Project Status:

Our project is currently in the development phase. We encourage contributions from team members and stakeholders to help us achieve our project goals.

Resources:

- Technical Documentation
- Design Prototype
- Testing page

Contact Information:

For any questions, suggestions, or feedback, please contact anmol.arora@tuni.fi

Notes

Here will be notes from meetings and other misc notes … for instance if task is to gather background information, document results here - can be very free style content.

NOTE: every time you have a meeting - write notes/summary/minutes here. Who was present, what was discussed and what was conclusion and APs.

Link to Child Confluence pages, 1 for each sprint

Sprint 1 - Secretary @Jennifer Hensley

Sprint 2 - Secretary @Sviatoslav Vasev

Sprint 3 - Secretary @Sviatoslav Vasev

Sprint 4 - Secretary @Sviatoslav Vasev , @Jennifer Hensley

Meeting notes in space

Create meeting note

Incomplete tasks from meetings

Description	Due date	Assignee	Task appears on
☐ Remove TG token from GitLab @Ozan Topcu		Ozan Topcu	2023-11-03 Meeting notes
☐ Try to do more			2023-10-04 Meeting notes
☐ Create Jira issues if you're doing something we didn't plan			2023-10-04 Meeting notes
☐ Log hours			2023-10-04 Meeting notes
☐ Choose new secretary every week			2023-10-04 Meeting notes

Decisions from meetings

Page Title	Decisions
2023-10-25 Spring planning	Don't forget to add a reviewer when you move a task to review column
2023-11-03 Meeting notes	Log in more hours
2023-11-08 Meeting notes	Move the weekly half an hour earlier
Sprint 2	Use public JobTeaser API for now and then set up API auth (probably never). Every task needs acceptance criteria (Either created by scrum master or assignee).

All meeting notes

Title	Creator	Modified
2023-12-13 Meeting notes	Sviatoslav Vasev	Dec 13, 2023
2023-10-04 Meeting notes	Sviatoslav Vasev	Dec 08, 2023
2023-11-29 Meeting notes	Jennifer Hensley	Nov 29, 2023
2023-11-22 Meeting notes	Sviatoslav Vasev	Nov 22, 2023

2023-11-17 Meeting notes	Sviatoslav Vasev	Nov 17, 2023
2023-11-15 Meeting notes	Sviatoslav Vasev	Nov 15, 2023
Sprint 4	Sviatoslav Vasev	Nov 10, 2023
2023-11-08 Meeting notes	Sviatoslav Vasev	Nov 08, 2023
2023-11-03 Meeting notes	Sviatoslav Vasev	Nov 03, 2023
2023-10-25 Spring planning	Sviatoslav Vasev	Oct 25, 2023
Sprint 2	Anmol Arora	Oct 06, 2023

Sprint 1

2023-10-04 Meeting notes

🗓️ Date

4 Oct 2023

👥 Participants

- @Sviatoslav Vasev - Secretary
- @Anmol Arora - Scrum master
- @Jennifer Hensley
- @Ozan Topcu
- @Sudikshya Bhattarai
- @Toan Le Tran

🗣️ Discussion topics

Time	Item	Presenter	Notes
11:07	Report	@Anmol Arora	• Pouta can be used • New tasks were added to Jira
11:09	Report	@Sviatoslav Vasev	• Nothing done
11:10	Report	@Ozan Topcu	• Can't set up the db because it's not chosen
11:11	Report	@Toan Le Tran	• Researched testing on Friday • **You should log hours when working**
11:11	Report	@Jennifer Hensley	• Discussed with @Sudikshya Bhattarai plans • Won't be able to come on Friday
11:12	Report	@Sudikshya Bhattarai	• Been working with examples from Petteri • User needs should be concrete examples
11:15-11:25	Progress review	Everyone	• **Create Jira issue for every thing you do** • DragonFly seems going nicely • PolarBear is extended for next week • You can change roles if you want • Try writing small description to the task that is created • Keep all notes from a sprint in a separate page for each sprint

✅ Action items

- [ ] Try to do more
- [ ] Create Jira issues if you're doing something we didn't plan
- [ ] Log hours
- [ ] Choose new secretary every week

27.09.2023, 29.09.2023 and 22.09.2023

Meeting 29.09.2023

Present Group Members:
Anmol Arora, Sviatoslav Vasev, Jennifer Hensley, Toan Tran, Sudikshya Bhattarai

What was discussed/done:

- Toan researched ways to test telegram bot

  Sudikshya worked on Specifications Confluence page

  Jennifer worked on Project Organization pages in Confluence
- Testing options were discussed
- What did we do, what will we do
- Problems were discussed
- Virtual Machine was discussed briefly
- New tasks were assigned
- Excel checklist was filled out

Notes for tasks to be finished:

- Specification Page gets child pages, for user needs, use case etc.
- User needs should include concrete example of what the bot does, practical example, how it is used

Meeting 27.09.2023 (teams)

Present Group Members:

Anmol Arora, Sviatoslav Vasev, Jennifer Hensley, Sudikshya Bhattarai, Ozan Topcu, Toan Tran

What was discussed/done:

- Thong left the project, the re-assignment of his tasks was discussed
- How to log work hours
- Every member gave updates on what they have done and what they are planning to do in the next days
- Confluence needs work during the next days
- Toan plans to develop CI/CD for gitlab and use pytest for writing tests
- Jennie needs defined project member roles to complete the Project Organization page and will work on DoD
- Sudi will keep working on the “Specifications” page on Confluence (needs some help with user needs)
- A “Technical Documentation” page was added to the Specification pages, which will be kept updated in the future.
- Future meetings will be held in teams on Wednesdays at 11 and Fridays on campus during class

Meeting 22.09.2023

Present Group Members:

Anmol Arora , Sviatoslav Vasev, Thong Hoang, Jennifer Hensley, Sudikshya Bhattarai, Ozan Topcu, Toan Tran

What was discussed/done:

- Jira setup

- Confluence setup
- issues were created and assigned
- Gitlab setup
- some roles were assigned :

– Scrum Master: Anmol Arora

– Project Owner: Jennifer Hensley/Sudikshya Bhattarai

- looking for API
- structure of the project was discussed and agreed on

Sprint 2

🗓️ Date

6 Oct 2023

👥 Participants

- @Anmol Arora - scrum master
- @Sviatoslav Vasev - secretary
- @Sudikshya Bhattarai
- @Toan Le Tran

🗣️ Discussion topics

Time	Item	Presenter	Notes
15:59	Tasks review	@Anmol Arora	• Should add reviewer field to tasks • If review needed from specific person, add them. • If scrum master creates issue, they will add acceptance criteria. If someone else creates it, they need to do it themself. • **Every task needs acceptance criteria.**
16:03	Work allocation	@Toan Le Tran	• Set up CI/CD • Simple test set up
16:04	Work allocation	@Sudikshya Bhattarai	• Do the Figma design. • Create acceptance criteria.
16:05	Work allocation	@Sviatoslav Vasev	• Create database schema
16:15	Work allocation	@Anmol Arora	• Look at python bot himself instead of Ozan (probably).

Decisions

- Use public JobTeaser API for now and then set up API auth (probably never).
- Every task needs acceptance criteria (Either created by scrum master or assignee).

2023-10-25 Spring planning

🗓️ Date

25 Oct 2023

👥 Participants

- @Sviatoslav Vasev - secretary
- @Anmol Arora
- @Jennifer Hensley
- @Ozan Topcu
- @Toan Le Tran

🗣️ Discussion topics

Time	Item	Presenter	Notes
11:02	Report	@Jennifer Hensley	• Taken over Sudi's figma task • Copying task is done. Review is needed. • Need to learn Figma, will try to do it today
11:06	Report	@Ozan Topcu	• Database type is selected. • Simple bot is created on telegram.
11:11	Report	@Sviatoslav Vasev	• Did database schema and chose API • Will work on tg bot code, @Ozan Topcu will do the scraping.
11:13	Report	@Toan Le Tran	• Trying to make CI/CD work
11:16	Report	@Anmol Arora	• Setting up Pouta, will add others there later • Hope that we'll be able to present POC on Friday

✅ Action items

- [ ]

Decisions

Don't forget to add a reviewer when you move a task to review column

Sprint 3

2023-11-03 Meeting notes

🗓️ Date

3 Nov 2023

👥 Participants

- @Sviatoslav Vasev - secretary
- @Anmol Arora - scrum master
- @Ozan Topcu
- @Toan Le Tran
- @Jennifer Hensley
- @Sudikshya Bhattarai

🗣️ Discussion topics

Time	Presenter	Notes
14:32	@Anmol Arora	• Sprint was started one day later • 2 epics in progress, focus on Crocodile • PolarBear needs some reviews • Password login will be disabled today
	@Ozan Topcu	• Will do backend: database to the cloud
	@Sviatoslav Vasev	• Reviewed Figma prototype
	@Jennifer Hensley	• Did Figma prototype
	@Sudikshya Bhattarai	• Was supposed to do the prototype but had to ask
	@Toan Le Tran	• Basic unit tests are set up • Next week set up CI/CD for gitlab
	@Anmol Arora	• People still don't log hours to Jira
	Jere	• Hour tracking: too few hours logged • GitLab doesn't have enough in it • TG token shouldn't be in version control

✅ Action items

- [ ] Remove TG token from GitLab @Ozan Topcu

Decisions

Log in more hours

2023-11-08 Meeting notes

🗓️ Date

8 Nov 2023

🗣️ Participants

- @Sviatoslav Vasev - secretary
- @Anmol Arora - scrum master
- @Jennifer Hensley
- @Toan Le Tran

🗣️ Discussion topics

Time	Presenter	Notes
	@Toan Le Tran	• Created .env file • We need to generate new api keys
	@Sviatoslav Vasev	• Figuring out how to use the library • It's tough because @Ozan Topcu was supposed to use another one.
	@Jennifer Hensley	• Working on the job listing page • Probably won't finish it before Friday
	@Toan Le Tran	• Reviewed the tasks in Jira

✅ Action items

- [ ]

🔀 Decisions

- Move the weekly half an hour earlier

Sprint 4

🗓️ Date

10 Nov 2023

👥 Participants

- @Sviatoslav Vasev
- @Anmol Arora
- @Ozan Topcu
- @Sudikshya Bhattarai

🥅 Goals

- Connect MySQL with TG bot
- Sync up the test with what the bot actually does

🗣️ Discussion topics

Time	Presenter	Notes
16:13	@Ozan Topcu	• Said many things but I didn't catch them
	@Anmol Arora	• Will make CI/CD on the machine • Will add Sviat to db
	@Sudikshya Bhattarai	• Don't know what to do • Will take Ozan's task
	@Sviatoslav Vasev	• Wrote boilerplate code for the bot

✅ Action items

🚰 Decisions

2023-11-15 Meeting notes

🗓️ Date

15 Nov 2023

👥 Participants

- @Sviatoslav Vasev - secretary
- @Anmol Arora - scrum master
- @Ozan Topcu
- @Jennifer Hensley
- @Toan Le Tran

🗣️ Discussion topics

Time	Presenter	Notes
10:33	@Anmol Arora	• Revising prev meeting: we need to connect all the systems together • Completed all reviews • Need to do the CI/CD
	@Sviatoslav Vasev	• Haven't tested if I have access to the db • Maybe we should just create one db user for everyone
	@Toan Le Tran	• Haven't done it but the next thing is to fix the tests
	@Jennifer Hensley	• Will keep working on job listing page
10:45	@Ozan Topcu	• Created a task for @Sudikshya Bhattarai, she will add a script that checks for duplicates. • Will add timestamps to job ads • We should add to CI/CD the db scripts

✅ Action items

- [ ]

Decisions

2023-11-17 Meeting notes

🗓️ Date

17 Nov 2023

👥 Participants

- @Sviatoslav Vasev - secretary
- @Anmol Arora - scrum master
- @Toan Le Tran
- @Jennifer Hensley
- @Sudikshya Bhattarai

🗣️ Discussion topics

Time	Presenter	Notes
15.45	@Sviatoslav Vasev	• Will work on adding new user to the db when a new chat is started • Will help @Jennifer Hensley with her task about job listing
	@Jennifer Hensley	• Will do the job listing page • Will review the slides for next Friday's presentation
	@Sudikshya Bhattarai	• Working on duplicates check (Ozan will do the other part) • Started but it's confusing
	@Toan Le Tran	• Fixed the tests and updated the Confluence • Testing the code • Will separate the tests into multiple files - categorize them (unit, etc) • Will add the test command to CD/CD file
	@Anmol Arora	• Will make a slide set and help @Toan Le Tran with CI/CD • Will be working demo for the Friday

✅ Action items

- [ ]

Decisions

2023-11-22 Meeting notes

🗓️ Date

22 Nov 2023

🫂 Participants

- @Sviatoslav Vasev - secretary
- @Anmol Arora - scrum master
- @Jennifer Hensley
- @Ozan Topcu
- @Sudikshya Bhattarai
- @Toan Le Tran

🗣️ Discussion topics

Time	Presenter	Notes
10:07	@Anmol Arora	• Still have to make the slides, will start today-tomorrow • Shared db user is working • Not started .yml file, waiting for an update
	@Jennifer Hensley	• Not done anything, having some other • Will push the code soon and still need to update the acceptance criteria
	@Ozan Topcu	• Updated the table with the codes, haven't finished tho - timestamps • Need to change the acceptance criteria
	@Sudikshya Bhattarai	• Done something but haven't tested it, will finish today-tomorrow
	@Sviatoslav Vasev	• Will do the things after this meeting
	@Toan Le Tran	• Updated the tests so that we can run as many files as possible • CICD not done - do together with @Anmol Arora

2023-11-29 Meeting notes

🗓️ Date

22 Nov 2023

👥 Participants

- @Jennifer Hensley - secretary
- @Anmol Arora - scrum master
- @Ozan Topcu
- @Sudikshya Bhattarai

🗣️ Discussion topics

Time	Presenter	Notes
10:30	@Anmol Arora	• recap and discussion of open tasks • will notify members of tasks that needs to be done
	@Jennifer Hensley	• will update Confluence
	@Ozan Topcu	• will add crontab
	@Sudikshya Bhattarai	• will support Jennifer with Confluence, if needed

Sprint 6

2023-12-13 Meeting notes

🗓️ Date

13 Dec 2023

🗣️ Participants

- @Sviatoslav Vasev
- @Anmol Arora
- @Toan Le Tran
- @Sudikshya Bhattarai

🗣️ Discussion topics

Time	Presenter	Notes
	@Toan Le Tran	• Will do the MIT license
	@Sviatoslav Vasev	• Saving users to the db is almost done - still need to research legal applications.
	@Anmol Arora	• cron task is done • Tasks are assigned in Jira
	@Sudikshya Bhattarai	•

✅ Action items

Decisions

Retrospectives

Add Retrospective

Error while fetching page properties report data: Bad Request

PSP2 Sprint 1

📋 Overview

Reflect on past work and identify opportunities for improvement by following the instructions for the Retrospective Play.

Date	06.10.2023
Team	P5
Participants	@Anmol Arora @Sviatoslav Vasev @Toan Le Tran @Jennifer Hensley @Sudikshya Bhattarai @Ozan Topcu

💭 Retrospective

ℹ Add your Start doing, Stop doing, and Keep doing items to the table below. We'll use these to talk about how we can improve our process going forward.

Start doing	Stop doing	Keep doing
• Creating a PoC by next Sprint, atleast a Hello World Telegram bot	• Skipping description field in issues.	• Loggin Hours to Jira

✅ Action items

- [ ] Discuss PoC working locally
- [ ] Take API into use and sort data which is needed
- [ ] @Ozan Topcu Look into the possibility of extracting Jira Timesheet reports via automation in Jira

PSP2 Sprint 2

📋 Overview

Reflect on past work and identify opportunities for improvement by following the instructions for the Retrospective Play.

Date	27 Oct 2023
Team	
Participants	@Anmol Arora (scrum master), @Sviatoslav Vasev (secretary), @Ozan Topcu , @Sudikshya Bhattarai , @Toan Le Tran

Retrospective

Add your Start doing, Stop doing, and Keep doing items to the table below. We'll use these to talk about how we can improve our process going forward.

Went well	Didn't go well
Local testing was set up successfully	Some tasks missing acceptance criteria and DoD
Sprint target was almost achieved	Reviewers weren't assigned
Small POC was created	Sigma wasn't completed
Confluence up to date	@Ozan Topcu hasn't logged any hours (worked lots tho)
Some people feel good	Some people feel bad

✅ Action items

- [ ] If you don’t log hours, you die :(
- [ ] Add DoD and acceptance criteria when starting working on a task
- [ ] Add measurable sprint goals

@Sviatoslav Vasev stays as a secretary

sprint 2 retro

PSP2 Sprint 3

📋 Overview

Reflect on past work and identify opportunities for improvement by following the instructions for the Retrospective Play.

Date	10 Nov 2023
Team	P5 Telegram bot
Participants	

Retrospective

ℹ️ Add your Start doing, Stop doing, and Keep doing items to the table below. We'll use these to talk about how we can improve our process going forward.

Start doing	Stop doing	Keep doing
• Connecting VM to telegram bot and sql DB	• Skipping log hours	• Updates in teams and whatsapp for communication

✅ Action items

- [ ] CI/CD should be setup
- [ ] Inform if you are going to skip team meeting in advance
- [ ] Push code on time, and inform other team mates

PSP2 Sprint 4

📋 Overview

Reflect on past work and identify opportunities for improvement by following the instructions for the Retrospective Play.

Date	24 Nov 2023
Team	P5-Telegram-Bot
Participants	

Retrospective

ℹ️ Add your Start doing, Stop doing, and Keep doing items to the table below. We'll use these to talk about how we can improve our process going forward.

Start doing	Stop doing	Keep doing
•	•	•

✅ Action items

- [ ]

Project

Project Organization and responsibilities

The group consists of 6 members.

Scrum Master: Anmol Arora - Main responsibility of the project, takes care of Jira.

Responsibility for Confluence Pages: Jennifer Hensley, Sudikshya Bhattarai - Responsible for Confluence Pages

Developers: Everyone, tasks will be mostly self-assigned according to who has time.

Testing : Toan

The Technical Documentation page will be updated by each developer.

Project Practices

Here:

- Meetings when and how
- Scrum events: daily, review, planning, retrospective
- Documentation and tools: Jira, Confluence, Git, Teams… provide links, add explanations/descriptions

Sprint duration: 2 weeks

Meetings:

Wednesdays at 11:00 in Teams / Plan for the next sprint every other week before the demo meeting

Click here to join the meeting

Fridays on Campus during the lesson

Scrum events:

- Sprint Planning
- Sprint Review
- Sprint Retrospective
- The Sprint

Documentation and Tools:

- Jira: P5 - Software Projects 2023
- Confluence: P5-Software-project
- Gitlab: https://gitlab.tamk.cloud/groups/sw-project-group-5
- Communication: Teams, WhatsApp group

GitLab

Merge requests ?

We name branch names with the convention:

```
PSP2-<Issue Number>-<issue description here>
```

So for instance if your issue number number is 1234 and task is to fix db issue the branch will be:

```
PSP2-1234-fixing-db-issue
```

Make sure to add the description of what you've done and include any special things the tester will need to do to make the code work.

Non-functional Requirements

A non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. It defines how a system is supposed to be. Here are some of the main non-functional requirements that are crucial to the success and trustworthiness of the app, especially related to data protection, performance, user experience, and system reliability.

Non-functional Requirements	Description
GDPR Compliance	Ensure that the app fully complies with the General Data Protection Regulation (GDPR) to protect user data and privacy.
Data Security	Implement encryption and security measures to protect user data, including personal information and search history from unauthorized access, breaches, and data leaks. Regularly conduct security audits and vulnerability assessments to protect user data from potential threats.
Performance:	The app should provide quick and relevant response times for searches and alerts.
Scalability:	Design the system to handle a growing number of users and increased job listings, ensuring that performance remains consistent even as usage scales up. Load testing can be conducted to ensure the app can handle a large number of concurrent users and search requests.
Availability	Maintain high availability to ensure that the app is accessible to users 24/7, minimizing downtime and service interruptions.
Reliability	Guarantee that the app operates reliably and consistently, with minimal errors or disruptions in job searches, alerts, and user interactions.
Compatibility	Ensure the app is compatible with a range of devices and platforms, including mobile devices (iOS and Android), web browsers and various versions of Telegram.
User-Friendly Interface	Prioritize a user-friendly interface to make it easy for users to navigate, search for jobs, and interact with the bot effectively.
Support and Troubleshooting	Implement a robust help and support system, including troubleshooting assistance and user guides, to aid users in resolving issues and getting the most out of the bot.
Future Maintenance	Plan for ongoing maintenance and updates to keep the app current and functional, ensuring it remains a

	valuable resource for job seekers.
Performance Monitoring	Set up monitoring and analytics tools to continuously track system performance, user engagement, and job matching accuracy, allowing for data-driven improvements.

Specifications

- **User needs:**

Job seekers expect a quick and easy procedure to locate suitable job ads based on their criteria, such as location, industry, job title, and keywords.

Personalization and Customization: Users value the option to customise their job search process, including the opportunity to save preferences and receive specialised job recommendations or notifications.

Clear and Useful Information: In order to make educated judgements, users look for clear and useful information about each job posting, such as the job title, firm name, location, and a brief description.

Friendly Interaction and Support: Users appreciate a bot interface that is simple to use, has clear instructions, handles errors, and gives them the option to ask for assistance or support if they run into problems.

- **Users Roles**

Job seekers who use the bot to find jobs should provide detailed information about a specific job they want to find including requirements, responsibilities, and application instructions.

Users can set up alerts for new matching jobs.

User should be implement secure methods to protect sensitive info.

- **Scope / Goals/ future plans:**

Create a Telegram bot from Job Teaser for job searching.
Allow users to look for jobs by location and keywords.
Display job descriptions and titles for open positions.
Upon user selection, provide thorough job information.

Allow users to create alerts for new jobs that match their qualifications.
In bot operations, ensure data security and privacy.
An easy-to-use interface should be the main priority.
Troubleshooting and basic user help should be implemented.
Consider future maintenance and scaling.

- **Technology :**

High level architecture : Bot Frontend, Bot Logic, Integration with Job Teaser,

Implementation technologies : Telegram Bot API, API Integration, Notification Service, Programming Languages, Security Tools, Version Control Documentation

- **Business Case**

Purpose for the project : To create a user-friendly Telegram bot that streamlines the job search process, providing job seekers with a convenient and efficient way to find relevant job listings from Job Teaser.

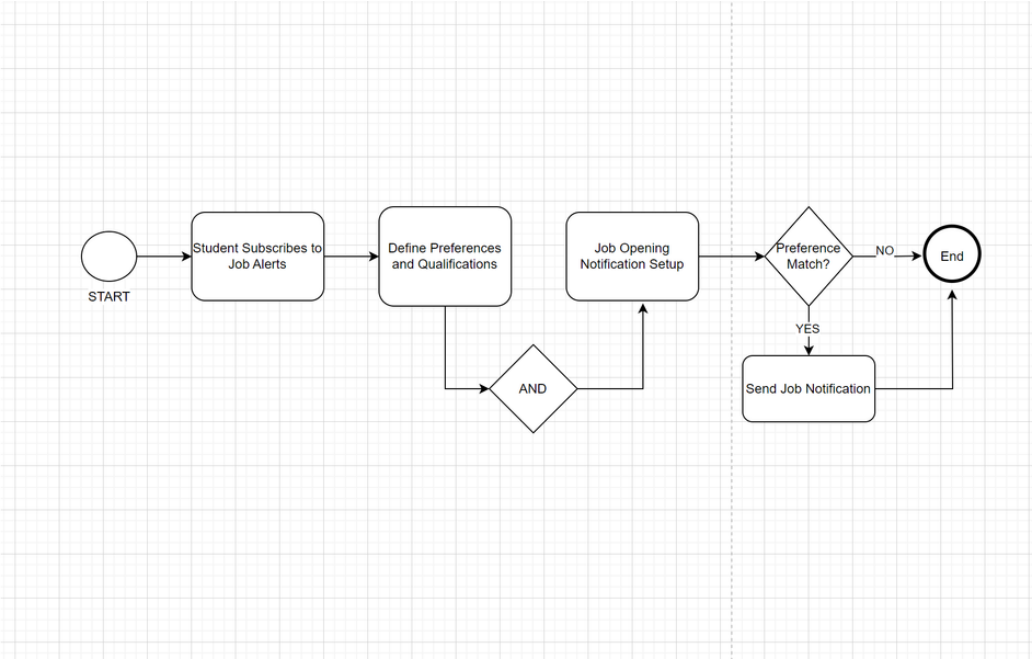
Benefits: Enhanced User Experience, Time-Saving, Personalized Alerts, Efficient Communication, Data Security, .Scalability

User Needs with concrete examples

User Needs:

User Need	Description	Scenario
Regular Job Alerts	Students want to regularly receive notifications about new jobs that match their preferences and qualifications.	Linus, a last-year student at Tampere University, wants to be notified immediately when a job opening in his field of computer science becomes available. He expects to get real-time alerts, so he can stay ahead in his search for a job.
Relevance	Students expect the job alerts to be relevant to their field of study, career goals, and location preferences.	Maija is studying for a finance degree and would like to work in an investment banking role. She needs the job search bot to filter and provide job listings that match this industry, to make sure the alerts are relevant to her.
Ease of Use	Students want a user-friendly interface that is easy to navigate without a need to spend much time or effort to set it up.	Enni, a student with limited technical skills, wants a bot that is straightforward to use. She expects clear and intuitive prompts when setting her job preferences.
Personalization	Users want the possibility to customize their job preferences, for example industry, job type and location	Matti is looking for an internship during the summer break. He wants the bot to allow him to change his job type preference, so it only shows internship offers.
Trustworthiness	Users need to trust that the bot will provide correct and up-to-date links to job listings.	Laura is worried about receiving spam or irrelevant job alerts. She relies on the bot to provide reliable and real job listings.

BPMN Diagram for User:



Explanation of the diagram:

- The process starts when Linus subscribes to job alerts.
- Linus defines his job preferences and qualifications.
- At the same time he sets up job opening notifications.
- The system checks if his preferences match available job openings.
- If the preferences do not match any job offers, the process ends immediately.
- If the preferences do match, the system sends a job notification.

Use Case

Job Search Telegram Bot

How It Works:

- User Interaction:
  Job seekers, as users, interact with Telegram bot through the Telegram messaging app.
- Preferences Setup:
  The user starts by setting their job search preferences within the bot.They specify their industry, location, job type, and other relevant criteria.
- Integration with Job Teaser:
  Telegram bot is integrated with Job Teaser, a reputable job listing platform.
- Real-Time Updates:
  The bot continuously monitors Job Teaser for new job listings that match the user's preferences.
- Job Recommendations:
  When a relevant job opening is found, the bot sends the user a real-time notification with a brief job description and a link to view the full listing on Job Teaser.
- Filtering and Personalization:
  The bot filters job listings based on the user's criteria, ensuring that they only see opportunities that match their skills and preferences.
- User-Friendly Interface:
  The bot provides a simple and user-friendly interface, allowing the user to easily browse job listings, receive updates, and even apply to jobs directly through the Telegram app.
- Remembering Preferences:
  Over time, the bot remembers the user's past preferences and adapts its recommendations as their job search needs evolve.
- Efficient Job Search:
  Users benefit from a streamlined and efficient job search process. They can conveniently access relevant job listings without spending excessive time on multiple job websites.

Features

Job Search

Job Listings & Details

User Interaction

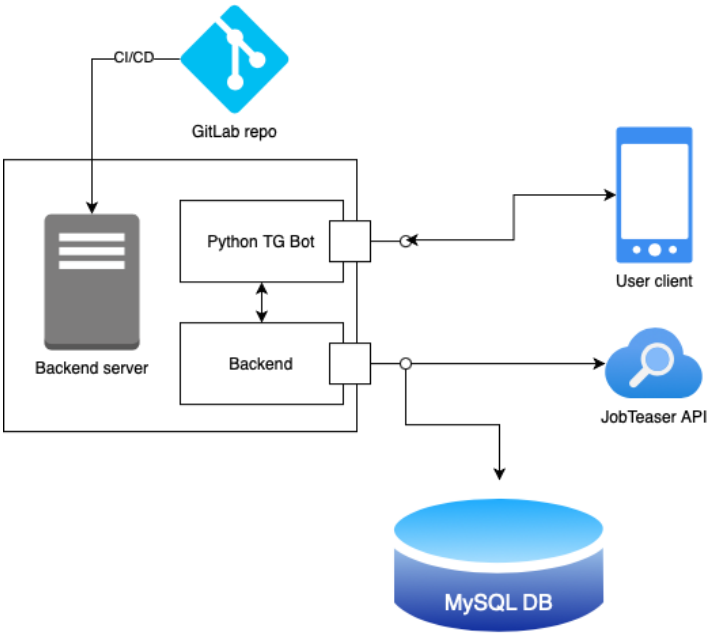
Job Alerts and Notifications

Scalability and Maintenance

Technical Documentation

The technical documentation pages are updated by each team member after working on a task.

SW Architecture



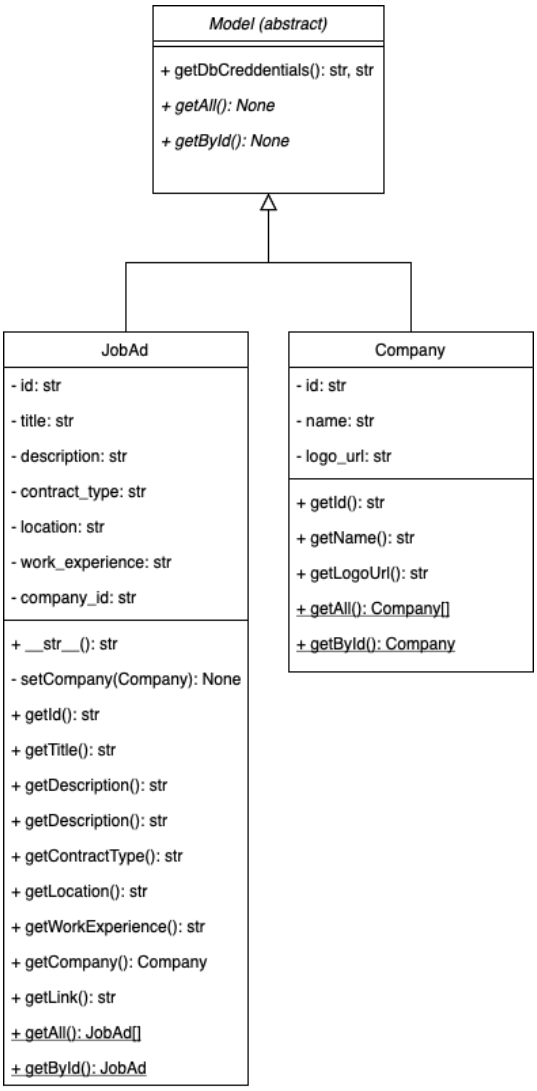
All code pushed to the `master` branch is automatically tested and deployed on the server.

The server runs two separate processes:

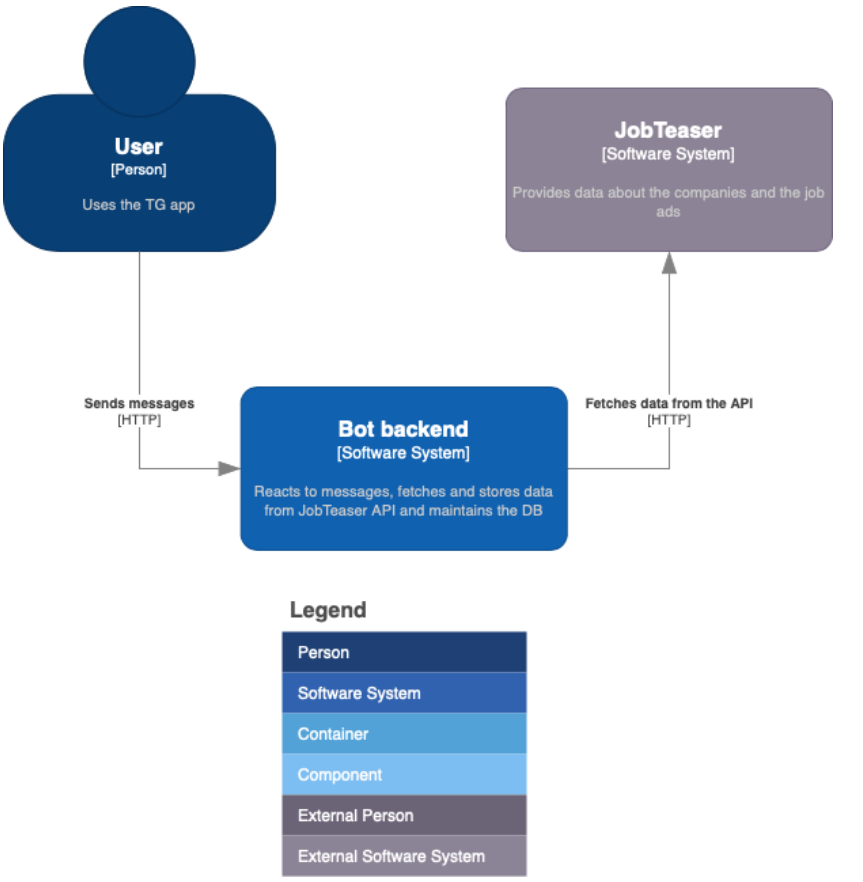
- Python TG Bot
- API Backend

User client communicates only with Python TG Bot, and it can't access directly backend API or the database.

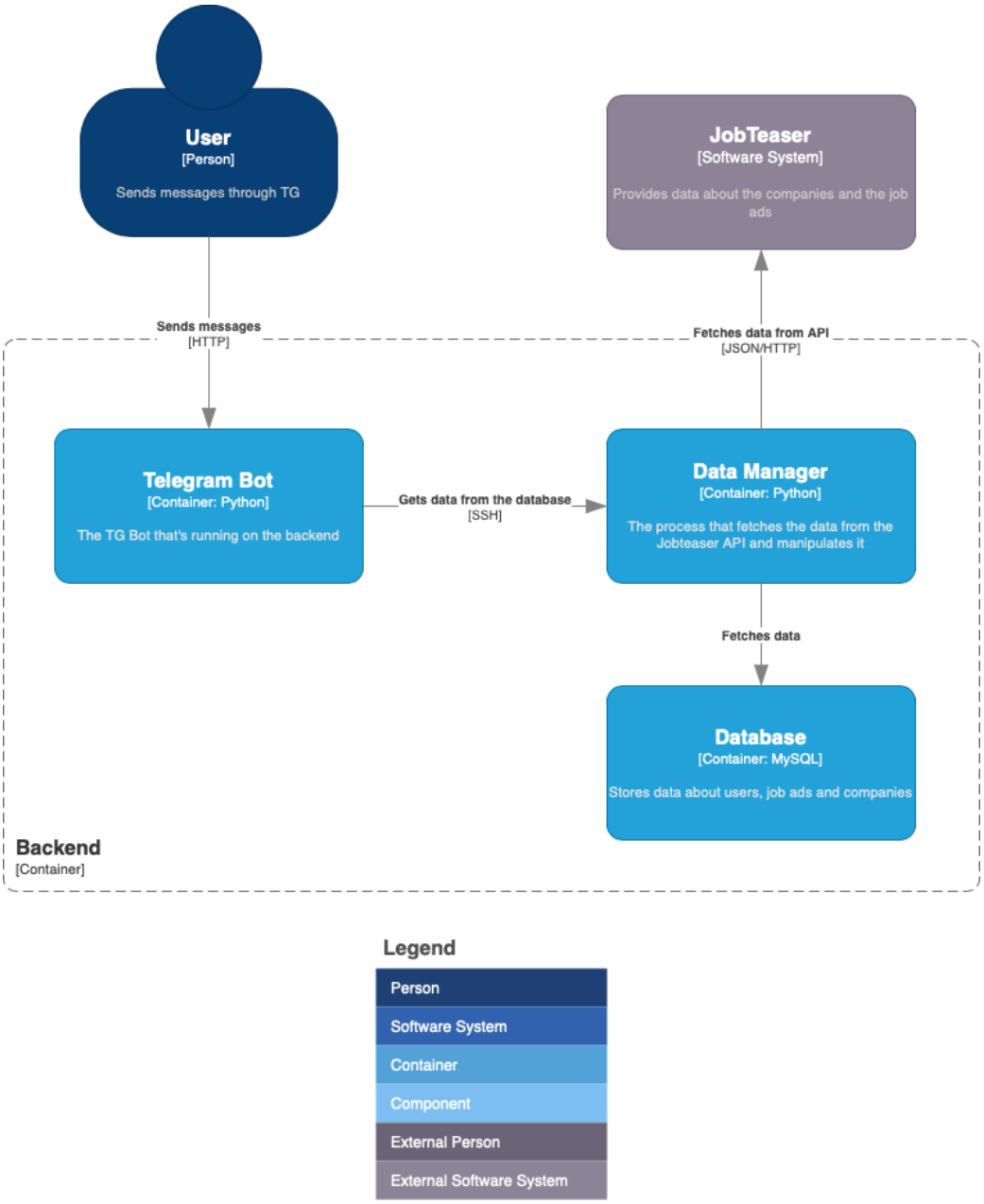
Class diagram



C4 Model - system level



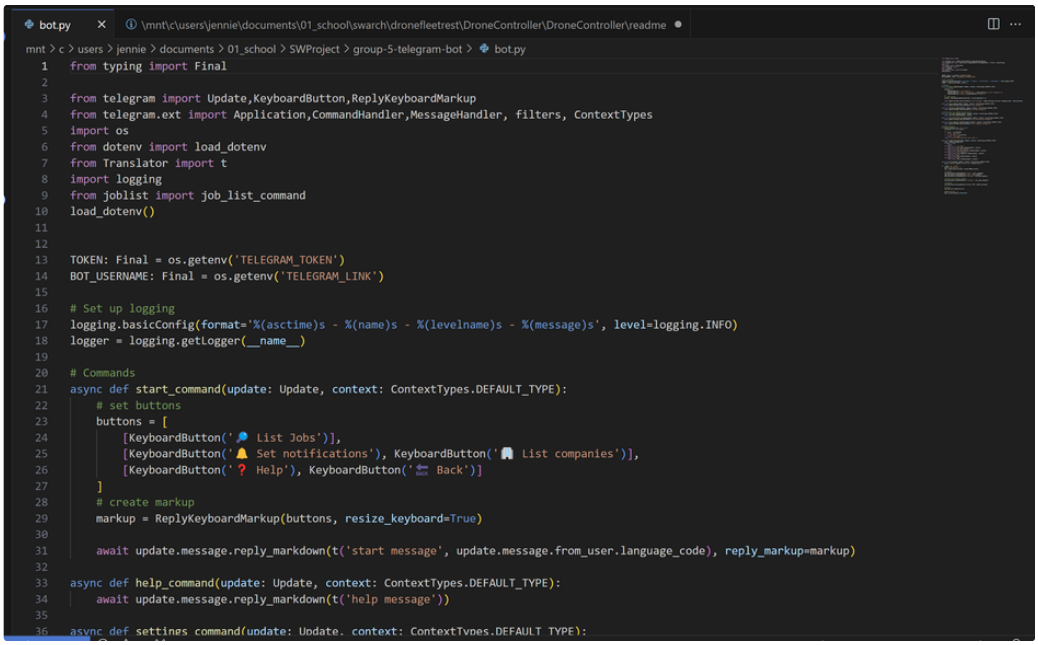
C4 diagram - container level


User
[Person]
Sends messages through TG
JobTeaser
[Software System]
Provides data about the companies and the job ads
Sends messages
[HTTP]
Fetches data from API
[JSON/HTTP]
Telegram Bot
[Container: Python]
The TG Bot that's running on the backend
Gets data from the database
[SSH]
Data Manager
[Container: Python]
The process that fetches the data from the Jobteaser API and manipulates it
Fetches data
Database
[Container: MySQL]
Stores data about users, job ads and companies
Backend
[Container]

Legend
Person
Software System
Container
Component
External Person
External Software System

Code

bot.py (main file)



```python
from typing import Final

from telegram import Update,KeyboardButton,ReplyKeyboardMarkup
from telegram.ext import Application,CommandHandler,MessageHandler, filters, ContextTypes
import os
from dotenv import load_dotenv
from Translator import t
import logging
from joblist import job_list_command
load_dotenv()


TOKEN: Final = os.getenv('TELEGRAM_TOKEN')
BOT_USERNAME: Final = os.getenv('TELEGRAM_LINK')

# Set up logging
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)
logger = logging.getLogger(__name__)

# Commands
async def start_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    # set buttons
    buttons = [
        [KeyboardButton('🔎 List Jobs')],
        [KeyboardButton('🔔 Set notifications'), KeyboardButton('🦷 List companies')],
        [KeyboardButton('❓ Help'), KeyboardButton('⬅️ Back')]
    ]
    # create markup
    markup = ReplyKeyboardMarkup(buttons, resize_keyboard=True)

    await update.message.reply_markdown(t('start message', update.message.from_user.language_code), reply_markup=markup)

async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_markdown(t('help message'))

async def settings_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_markdown(t('settings message'))

# Connect job_list_command from joblist.py
async def list_jobs_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await job_list_command(update, context)

async def set_notifications_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_markdown(t('set notifications message'))

async def list_companies_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_markdown(t('list companies message'))

# Handle Responses
def handle_response(txt: str) -> str:
    processed: str = txt.lower()

    if 'hello' in processed:
        return 'Hey there!'
    if '🔎 list jobs' in processed:
        return 'I am good!'
    return 'I do not understand what your wrote :('

async def handle_message(update: Update, context: ContextTypes.DEFAULT_TYPE):
    text = update.message.text
    if text is None:
        return
    elif text == '🔎 List Jobs':
        return await list_jobs_command(update, context)
    elif text == '🔔 Set notifications':
        return await set_notifications_command(update, context)
    elif text == '🦷 List companies':
        return await list_companies_command(update, context)
    elif text == '❓ Help':
        return await help_command(update, context)
```

```python
    elif text == '⬅ Back':
        return await start_command(update, context)

async def error(update: Update, context: ContextTypes.DEFAULT_TYPE):
    print(f'Update {update} caused error {context.error}')

if __name__ == '__main__':
    print('Starting bot...')
    app = Application.builder().token(TOKEN).build()

    # Commands
    app.add_handler(CommandHandler('start', start_command))
    app.add_handler(CommandHandler('help', help_command))
    app.add_handler(CommandHandler('settings', settings_command))

    # Handle the /listjobs command
    app.add_handler(CommandHandler('listjobs', list_jobs_command))

    # Messages
    app.add_handler(MessageHandler(filters.TEXT, handle_message))

    # Errors
    app.add_error_handler(error)

    print('Polling...')
    app.run_polling(poll_interval=3)

```


joblist.py

```python
from telegram.ext import ContextTypes
from telegram import Update
from Translator import t  # Assuming this is your translation module
import mysql.connector

# get job data from the database
def get_job_data_from_database():
    try:
        # db connection
        mydb = mysql.connector.connect(
            host='[redacted]',
            user='[redacted]',
            password='[redacted]',
            database='user_infodb'
        )
        mycursor = mydb.cursor()

        # Fetch job data from the database
        mycursor.execute("""
            SELECT job_ads.id, job_ads.title, companies.id, companies.name, companies.logo_url
            FROM job_ads
            INNER JOIN companies ON job_ads.company_id = companies.id
        """)
        result = mycursor.fetchall()

        job_list = []
        for row in result:
            job_list.append({
                'title': row[1],
                'link': f'https://www.jobteaser.com/en/job-offers/{row[0]}',
                'company_name': row[3],
                'company_logo_url': row[4],
            })

        # Close the database connection
        mydb.close()
```

```python
        return job_list

    except mysql.connector.Error as err:
        print(err)
        raise

# job listing handler
async def job_list_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    try:
        language = "en"
        list_jobs_message = t("list jobs message", language) + "\n"

        job_list = get_job_data_from_database()

        if job_list:
            message = list_jobs_message
            for job in job_list:
                message += f"*{job['title']}* at {job['company_name']} - {job['link']}\n"

            await update.message.reply_markdown(message)
        else:

            await update.message.reply_markdown(list_jobs_message + t("no jobs found"))

    except Exception as e:
        print(e)
        error_message = t("error", language)
    await update.message.reply_markdown(error_message)

```

jobteaser_data.py

```python
import requests
import json
import mysql.connector

# Database connection
mydb = mysql.connector.connect(
    host=[redacted],
    user=[redacted],
    password=[redacted],
    database='user_infodb'
)
mycursor = mydb.cursor()

# Fetching the data from the given API link
url = 'https://apigw.jobteaser.com/jobteaser.job_ad.v2alpha/JobAdService/SearchJobAds'
header = {
    "query": "",
    "page_size": 20,
    "page_token": "0",
    "sponsored_results_max": 2,
    "career_center_study_levels": [
        1,
        2,
        3,
        4,
        5
    ],
    "locations": [
        "Finland::Pirkanmaa::Tampere::Tampere"
    ],
    "locales": [
        "en",
        "fi"
    ],
    "contract_types": [
        "job",
```

```
        "cdi",
        "part_time",
        "stage",
        "company_inside",
        "internship",
        "thesis",
        "thesis"
    ],
    "curriculum_ids": [
        "f5d66eac-700a-41c8-9943-6604e9b103d8"
    ],
    "radius_km": 200
}
json_data = requests.post(url, json = header).content
parsed_data = json.loads(json_data)
job_ad_id = []
title = []
description = []
contract_type = []
location = []
company_id = []
company_name = []
company_logo_url = []
work_experience = []
position_category = []


# Logic
for job in parsed_data["job_ads"]:
    job_ad_id.append(job["id"])
    title.append(job["title"])
    description.append(job["description"])
    contract_type.append(job["contract"]["type"])
    location.append(job["location"]["country"])
    company_id.append(job["company"]["id"])
    company_name.append(job["company"]["name"])
```

```
    company_logo_url.append(job["company"]["logo_url"])
    work_experience.append(job["work_experience"])
    position_category.append(job["position_category"])

for i in range(len(parsed_data["job_ads"])):
    sql = f"""
    INSERT INTO companies(id, name, logo_url)
    VALUES (%s, %s, %s);
    """
    values = (company_id[i], company_name[i], company_logo_url[i])
    mycursor.execute(sql, values)

for j in range(len(parsed_data["job_ads"])):
    sql = f"""
    INSERT INTO job_ads(id, title, description, contract_type, location, company_id, work_experience)
    VALUES (%s, %s, %s, %s, %s, %s, %s);
    """
    values = (job_ad_id[j], title[j], description[j], contract_type[j], location[j], company_id[j], work_experience[j])
    mycursor.execute(sql, values)

for k in range(len(parsed_data["job_ads"])):
    sql = f"""
    INSERT INTO categories(job_ad_id, category)
    VALUES (%s, %s);
    """
    values = (job_ad_id[k], position_category[k])
    mycursor.execute(sql, values)


# Commiting the queries and closing the database connection
mydb.commit()
mydb.close()
```

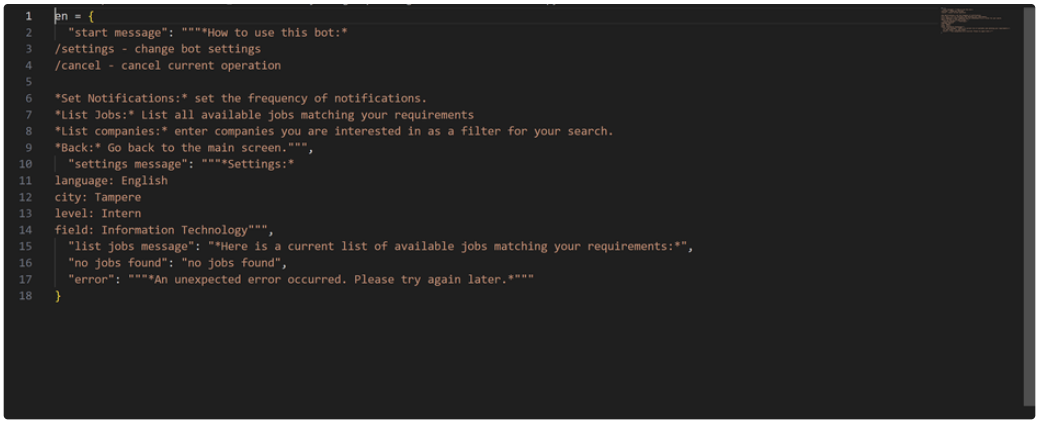
Translator.py

```
import translations;

def t(key, language="en"):
  if language == "en":
    dictionary = translations.en
#  else if language == "fi":
#    dictionary = translations.fi
  else:
    return key

  if key in dictionary.keys():
    return dictionary[key]
  else:
    return key
```

translations.py



```
en = {
  "start message": """*How to use this bot:*
/settings - change bot settings
/cancel - cancel current operation

*Set Notifications:* set the frequency of notifications.
*List Jobs:* List all available jobs matching your requirements
*List companies:* enter companies you are interested in as a filter for your search.
*Back:* Go back to the main screen.""",
  "settings message": """*Settings:*
language: English
city: Tampere
level: Intern
field: Information Technology""",
  "list jobs message": "*Here is a current list of available jobs matching your requirements:*",
  "no jobs found": "no jobs found",
  "error": """*An unexpected error occurred. Please try again later.*"""
}
```

Installation and Operating Instructions

This part has following child pages:

Installing, setting up and starting the containers

Demonstration instructions

Deployment Diagram C4 Model

Installing, setting up and starting the containers

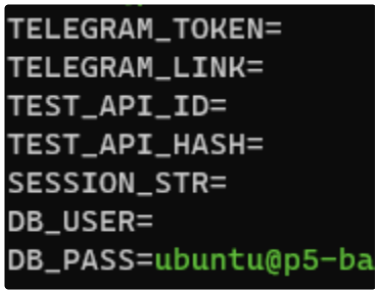
To install the Telegram bot you first need to clone the repo or extract the code from the zip file provided to the customers.

1. https://gitlab.tamk.cloud/sw-project-group-5/group-5-telegram-bot

   git clone ssh://git@gitlab.tamk.cloud:1022/sw-project-group-5/group-5-telegram-bot.git customer-side-installation
2. unzip customer-side-installtion.zip
3. cd into the directory
   cd customer-side-installation

4. Run pip install to get all the dependencies
   pip3 install –r requirements.txt

5. Here is an example of .env file for our system, so you need to copy these environments variable name in to file name. env and then add all your credential there in your all .env file. Since .env file is in .gitignore so it will not be pushed to remote repository.



   TELEGRAM_TOKEN: when you create a telegram Bot with ”BotFather”
   TELEGRAM_LINK: telegram bot name when we create with ”BotFather”, when enter this to broswer or telegram chat search, the bot should show up
   TEST_API_ID and TEST_API_HASH are tokens that create from your own telegram account (for testing you need this so that the application can send message to our telegram bot by using your account)
   SESSION_STR: option of Telethon to format your session data into a not-so-long string that can be easily fit into an environment variable
   To get SESSION_STR you can run this code or visit this website tutorial for better understanding

```python
import pytest
import os
from telethon import TelegramClient
from telethon.sessions import StringSession
import pytest_asyncio
from dotenv import load_dotenv
load_dotenv()
# Your API ID and hash here
api_id = os.getenv('TEST_API_ID')
api_hash = os.getenv('TEST_API_HASH')
session_str = os.getenv('SESSION_STR')


with TelegramClient(StringSession(), api_id, api_hash) as client:
    print("Your session string is:", client.session.save())
```

5. Then we need to get the database back from the backup file, check that mysql is installed with a user sql with its password set to ‘sqlpass’ (these settings can be changed but then requires more changes to other scripts) and then use the command
   mysql –u sql –p sqlpass **user_infodb** < backup.sql

6. Next you can run the service
   python3 jobteaser_data.py

   Then run
   python3 bot.py

Demonstration Instructions

Demonstration instructions

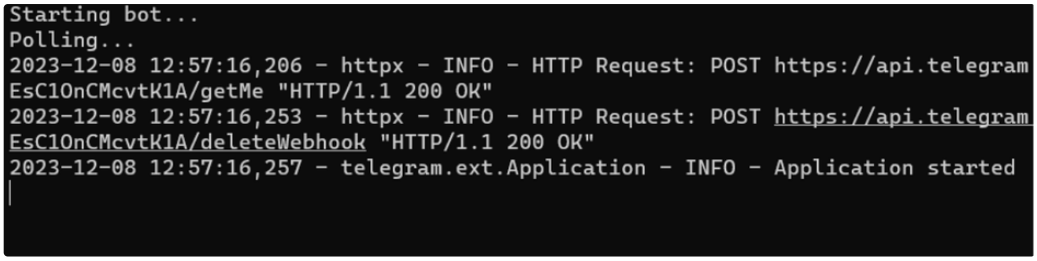
The front end of our project is available on telegram 24/7. The bot's name is “Job Application Bot” and can be tested from any mobile device or PC that has Telegram downloaded.

After the code has been setup and the credentials have been added to the .env file

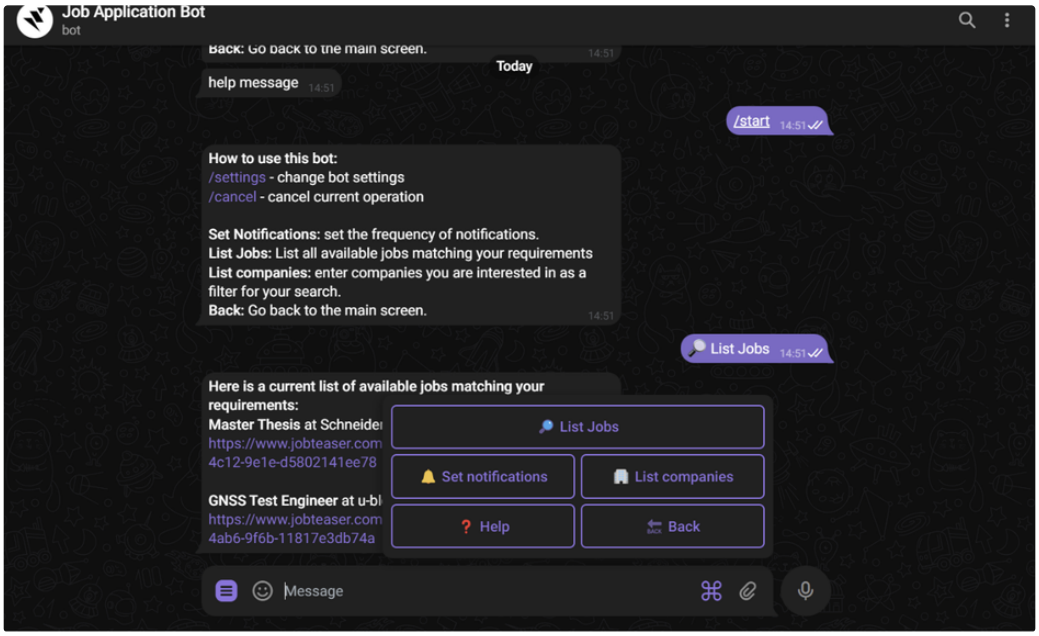
python3 bot.py can be used to run the bot

Then you can go to telegram and add the bot by searching for nickname – “@JobTeaser_bot”

Once all done the bot.py output should look like this

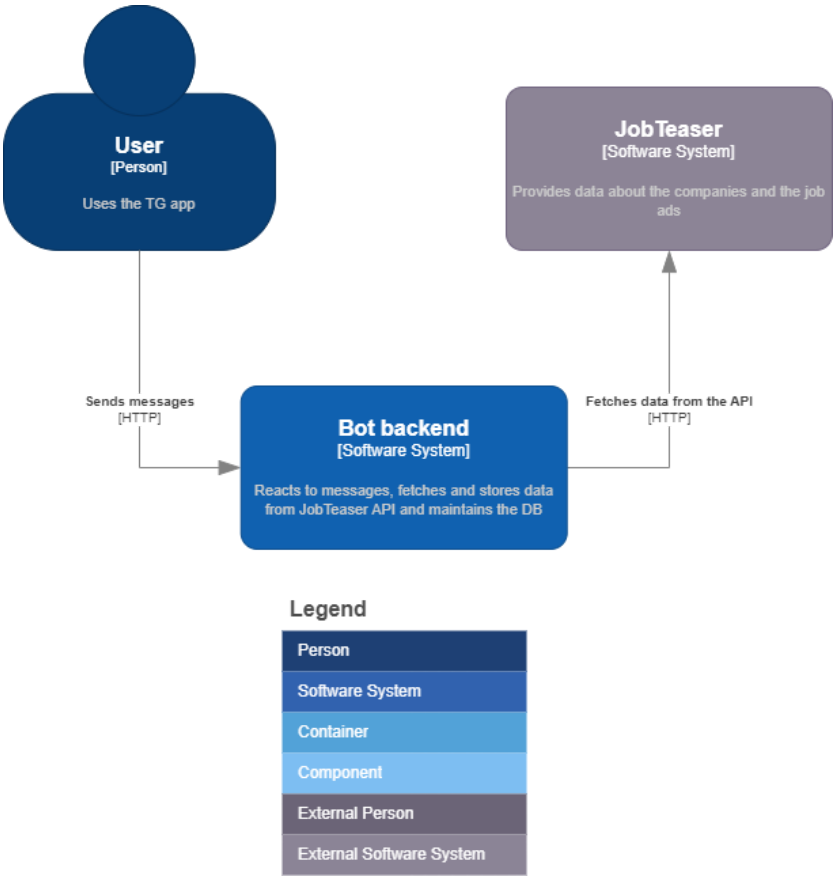


And then you can use the bot.

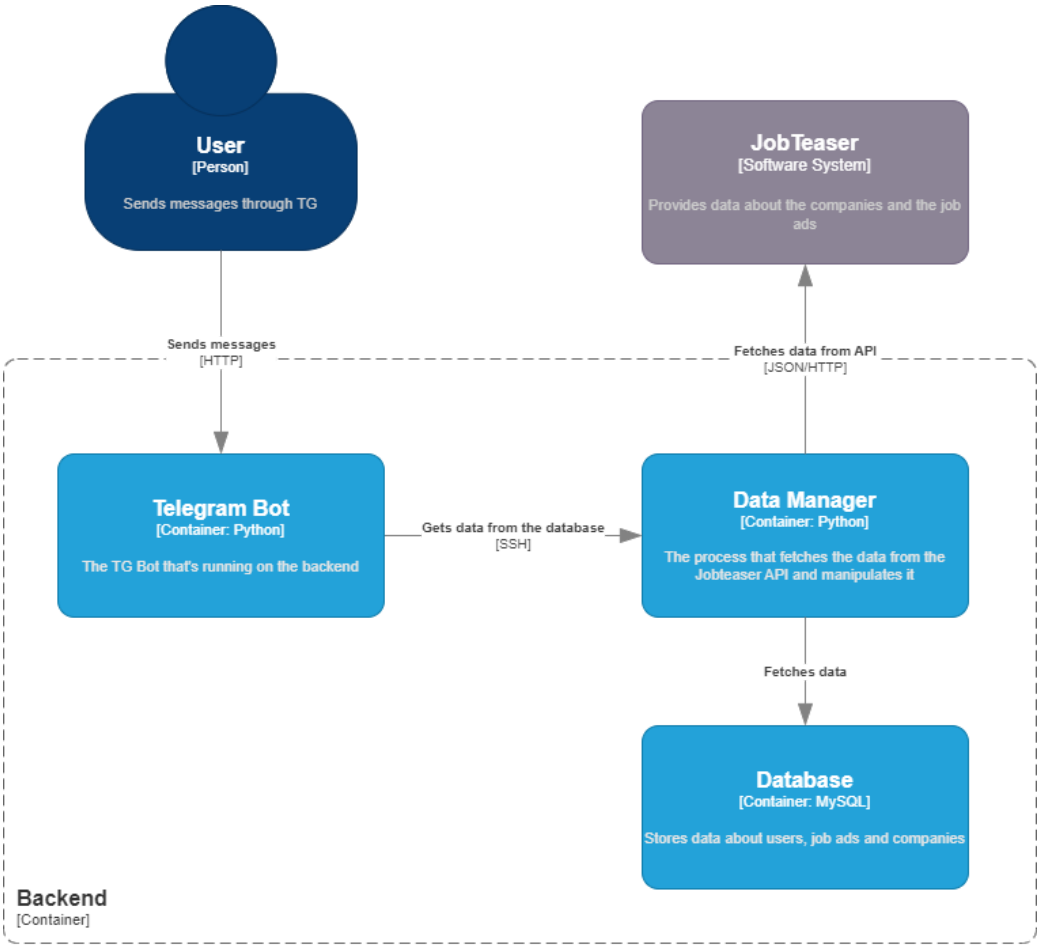


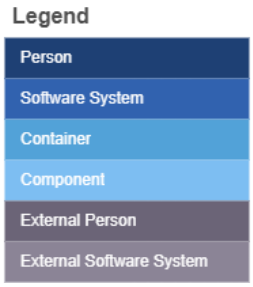
Deployment Diagram C4 Model

C4 Model - system level



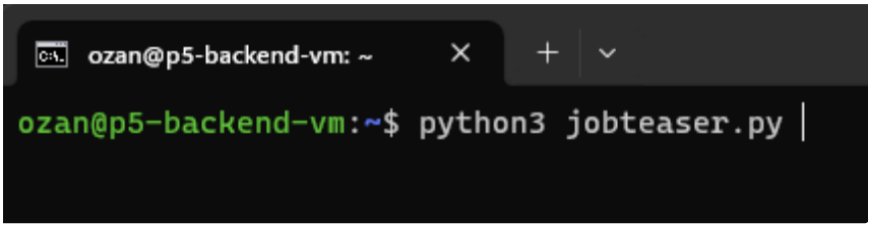
C4 diagram - container level


User
[Person]
Sends messages through TG
JobTeaser
[Software System]
Provides data about the companies and the job ads
Sends messages
[HTTP]
Fetches data from API
[JSON/HTTP]
Telegram Bot
[Container: Python]
The TG Bot that's running on the backend
Gets data from the database
[SSH]
Data Manager
[Container: Python]
The process that fetches the data from the Jobteaser API and manipulates it
Fetches data
Database
[Container: MySQL]
Stores data about users, job ads and companies
Backend
[Container]

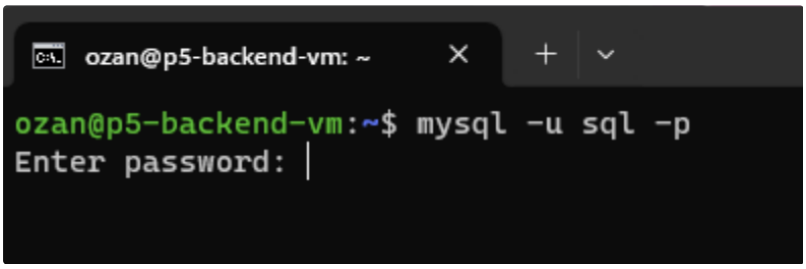

Legend
Person
Software System
Container
Component
External Person
External Software System

Developer Guide

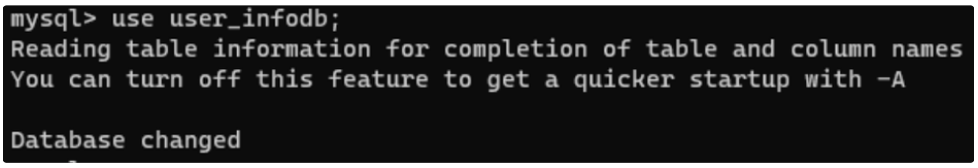
For the backend tester, access to the VM must be granted to the tester since it is usually not needed as an end-user. When the cloud administrator or in our case project manager provides access, the demonstration can be done by running the jobteaser.py with the “python3 jobteaser.py” command. This does not return any output to the console; however, the results can be visualized from the database.



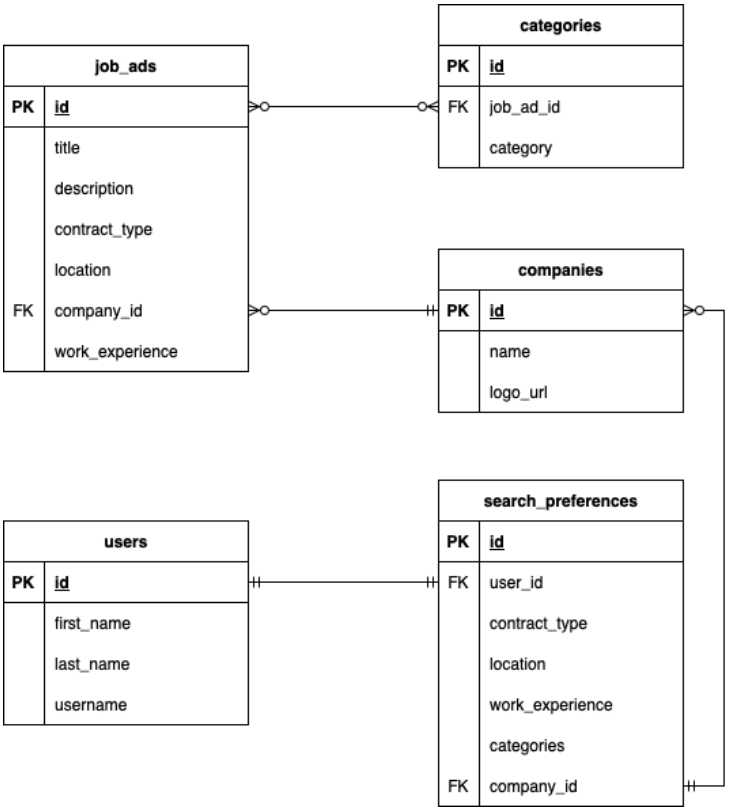
For the database, the access should be done from a specific user called "sql”,



The database is the database called “user_infodb” and it should be invoked with “use user_infodb;”



Data



- Each `User` that sends a message to the bot will be saved in a *users* table.
- `User` can select `SearchPreference`s that will be saved in *search_preferences* table.
- Each `User` can have only one `SearchPreference`.
- A JobAd can have many categories attached to it. To make the data aggregation easier, we store categories in a separate table.

JobTeaser API

Send a POST request to https://apigw.jobteaser.com/jobteaser.job_ad.v2alpha/JobAdService/SearchJobAds
with the following content:

```
{
    "query": "",
    "page_size": 20,
    "page_token": "0",
    "sponsored_results_max": 2,
    "career_center_study_levels": [
        1,
        2,
        3,
        4,
        5
    ],
    "locations": [
        "Finland::Pirkanmaa::Tampere::Tampere"
    ],
    "locales": [
        "en",
        "fi"
    ],
```

```
    "contract_types": [
        "job",
        "cdd",
        "cdi",
        "part_time",
        "stage",
        "company_inside",
        "internship",
        "thesis",
        "thesis"
    ],
    "curriculum_ids": [
        "f5d66eac-700a-41c8-9943-6604e9b103d8"
    ],
    "radius_km": 200
}
```

References

Telegram types - Telegram Bot API

JobTeaser has a public API but it doesn't return the same result as the auth API. We will use the public API for the first version of the product. Later (in theory) we need to request developer access to the API from JobTeaser.

Sample job ad response from JobTeaser API:


```
{
    "id": "20b6032c-6109-4242-a083-ee496b2c1fe3",
    "title": "Talent Acquisition Consultant ",
    "description": "Are you an experienced talent acquisition or HR professional looking for a new challenge in
    "start_period": "0",
    "contract": {
        "type": "cdi"
    },
    "location": {
        "country": "Finland",
        "state": "Uusimaa",
        "sub_state": "Helsinki",
        "city": "Helsinki"
    },
    "company": {
        "id": "473e9136-0160-47a2-b901-1e761c302bbc",
        "name": "Dream Broker Ltd",
        "slug": "dream-broker",
        "logo_url": "https://d1guu6n8gz71j.cloudfront.net/system/asset/logos/4228336/logo.png?1686315631"
    },
    "activation_time": {
        "seconds": 1696324003
    },
    "locations": [
        {
            "country": "Finland",
            "state": "Uusimaa",
            "sub_state": "Helsinki",
            "city": "Helsinki"
        }
    ],
    "legacy_school_ids": [
```

```
        0
    ],
    "legacy_partner_school_ids": [
        6618,
        6946
    ],
    "remote_type": "remote_partial_allowed",
    "work_experience": "three_to_five_years",
    "position_category": "746c77de-5ea9-4999-946c-f4c5a44eee83",
    "company_business_type": "sme",
    "apply_format": 1,
    "candidacy_type": 1,
    "slug": "11731497-dream-broker-ltd-talent-acquisition-consultant",
    "complementary_tracking_infos": {
        "page_size": 20,
        "search_id": "37a74be4-07d3-4365-91ed-4539c55f07f4"
    }
}
```

UI

When a user first opens the chat with a Telegram bot, it shows the description of what the bot can do. The user has to press “Start“ button to start the chat.

After that the user is presented with possible options:

- List Jobs
- Set notifications
- List companies
- Help
- Back

Each button incorporates an emoji to make the navigation easier for a user.

The final prototype in one picture:

Link to the Prototype (press the play button on the top right to try it out!)

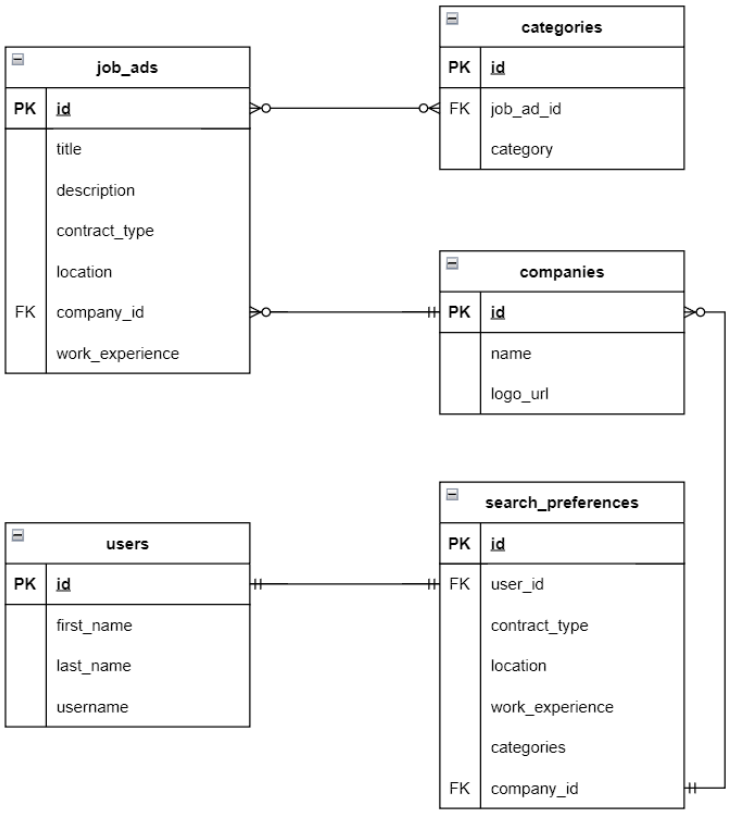
Database

Chosen Database

MySql was chosen for this project for its well-suited nature for structured data with predefined schemas. Our job data can be neatly organized into tables with relationships between them, and the support of MySQL for these needs makes it a viable option.

Database Schema

As mentioned on the Data page in Confluence the data schema is as follows:



Data Retrieval and Updates

The data will be fetched through a javascript program by sending HTTPS requests to JobTeaser's official API; this program will be run using a cron job scheduler to provide periodic updates to the database.

Note: It is important to know that the program that updates the database and the telegram bot will work asynchronously and will have no connection whatsoever, the database is used as a main storage where any request through telegram will be sent over.

Setting Up the Database

These are the commands that are used to create the table:

```
-- Create the companies table
CREATE TABLE companies (
    id VARCHAR(100) PRIMARY KEY,
    name VARCHAR(100),
```

```
    logo_url VARCHAR(255)
);

-- Create the job_ads table with a foreign key constraint
CREATE TABLE job_ads (
    id VARCHAR(100) PRIMARY KEY,
    title VARCHAR(100),
    description TEXT,
    location VARCHAR(100),
    contract_type VARCHAR(100),
    work_experience VARCHAR(100),
    company_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (company_id) REFERENCES companies(id)
);

-- Create the job_ads table with a foreign key constraint
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    category TEXT,
    job_ad_id VARCHAR(100),
    FOREIGN KEY (job_ad_id) REFERENCES job_ads(id)
);

-- Create the users table
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    username VARCHAR(255)
);

-- Create the search_preferences table with a foreign key constraint
CREATE TABLE search_preferences (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    contract_type TEXT,
    location VARCHAR(100),
    work_experience INT,
    categories TEXT,
    company_id VARCHAR(100),
    FOREIGN KEY (company_id) REFERENCES companies(id),
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Enabling SSH on New Pouta VM.

- SSH key is added to the machine.

IP	Admin	Users
128.214.252.173	anmol	jenni
		ozan
		sudi
		sviat
		toan
		sql

Telegram bot testing

We can discuss to use among three testing libraries for our telegram bot, choices are:
Using Pytest
Using pyTelegramBotAPI build in testing
Using Unittest mock
Using Unittest mock and Pytest

1. Tools and libraries
   - Pytest
   - Build in testing of pyTelegramBotAPI library with flask
   - Unittest.mock
2. File structure
   project_root/
   ├── my_telegram_bot.py (main program)
   ├── tests/
   │ ├── test_my_telegram_bot.py (is to write test cases.)
   │ ├── conftest.py(is a configuration file for pytest.)
   │ └── fixtures/ (is a directory store fixture files, like a bot instance)
   │ └── bot_instance.py
3. Run test command

   ```
   pytest
   ```
4. CI/CD
   - Integrate our test into gitlab CI/CD
   - Objective: run test before deploy state

5. Different between pytest and Unitest mock
   - unittest.mock (part of Python's unittest library): unittest.mock is primarily focused on creating mock objects and controlling their behavior during unit testing. It is a module within the broader unittest framework, which provides a full testing framework including test discovery, test fixtures, and test runners.

   - pytest: pytest is a testing framework that provides a more concise and expressive way to write and organize tests. While pytest does not include built-in mocking capabilities like unittest.mock, it allows you to use other mocking libraries like unittest.mock, pytest-mock, or pytest-monkeypatch for creating mock objects when needed.
6. Reference
   - pytest: helps you write better programs — pytest documentation
   - Building Telegram bot Using pyTelegramBotAPI and Flask : Forums : PythonAnywhere (www.pythonanywhere.com)
   - pyTelegramBotAPI/tests/test_handler_backends.py at master · eternnoir/pyTelegramBotAPI (github.com) this test system using pytest

Acceptance Tests

1. **Job Search Feature Acceptance Tests:**

Description	Test	Expectation
Search by Job Title or Skills	Enter a specific job title or a skill in the search.	The system should return job postings that match the entered job title or skills.
Filter by Location	Filter job postings by specifying a city, state, or nation.	The system should display job listings in the selected location.
Refine Results by Fields of Interest	Choose one or more fields of interest.	The system should narrow down job listings to match the selected fields of interest.
Choose Experience Level	Select an experience level (entry-level, mid-level, senior).	The system should show job listings that align with the selected experience level.
Saved Searches	Saved Searches	The system should store the saved search.
Real-Time Results	Search, and then check if the results are current and match the search parameters.	The system should show real-time job posts that are relevant to the search criteria.

2. **Job Alert Feature Acceptance Tests:**

Description	Test	Expectation
Customized Job Alerts Setup	Create a personalized job alert with keywords, location, industry, job type, and frequency (daily, weekly).	The system should save the job alert with the defined parameters.
Alert Notifications	Receive a job alert notification via Telegram including a short job description, company name, and a link to the full job listing.	The system should send timely and informative job alert notifications.
Job Alert Management	Edit an existing job alert to change the criteria or frequency of the job alerts. Temporarily disable a job alert. Delete a job alert.	The system should allow users to manage their job alerts effectively.
Recommendation Engine	Explore the recommendation engine to see if it suggests job alert criteria based on the user's profile and search history.	The system should provide relevant job alert recommendations.

Job Alert History	Access the job alert history to check past job listings sent through alerts.	The system should display a history of job listings received via alerts.

Testing setup local development

1. Introduction

   This document will show basic information about our telegram bot testing system for local development

2. Python library

   telethon

   pytest

   anyio

3. Setup for local development

   - Install libraries
   - Follow this instruction to get your api_id, api_hash, session_str
   - Open 2 terminals, one for running telegram bot, the point is to keep it running so our test file can send and receive message from out bot for testing

     The second terminal we run testing file for example “telegramTestOne.py“

4. Command to run the test at local machine

   - First run the bot: python bot.py
   - Get in to test file and run the test: pytest

5. Basic block of code

```
# Your API ID and hash here
api_id = YOUR_API_ID
api_hash = YOUR_API_HASH

session_str = YOUR_SESSION_STR
```

4.1 Add variables

```
@pytest.fixture(scope="session")
async def conv():
    client = TelegramClient(
        StringSession(session_str), api_id, api_hash,
        sequential_updates=True
    )
    await client.connect()
    async with client.conversation("https://t.me/JobTeaser_bot") as conv:
        yield conv
```

4.3 Connect to telegram bot.

```
@pytest.mark.anyio
async def test_start(conv):
    # Send a message to the bot
    await conv.send_message("/help")

    # Wait for and get the response message
    response = await conv.get_response()

    # Verify if the response message contains the expected text
    assert "I am here to help you with listing job applications ....." in response.text
```

4.4 Write unit test.

Retrospective: PSP2 Sprint 5

📋 Overview

Reflect on past work and identify opportunities for improvement by following the instructions for the Retrospective Play.

Date	
Team	
Participants	

Retrospective

ℹ️ Add your Start doing, Stop doing, and Keep doing items to the table below. We'll use these to talk about how we can improve our process going forward.

Start doing	Stop doing	Keep doing
•	•	•

✅ Action items

- [ ]