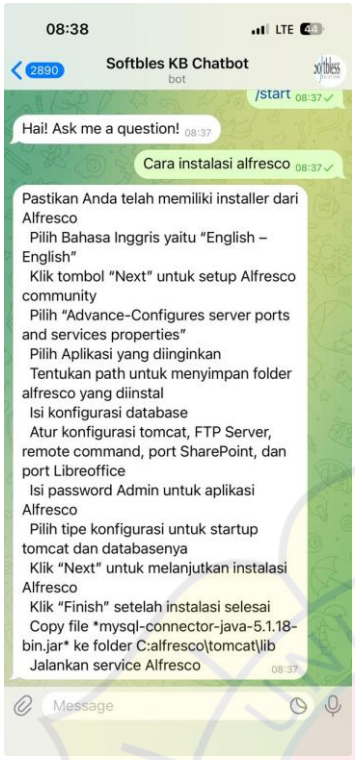
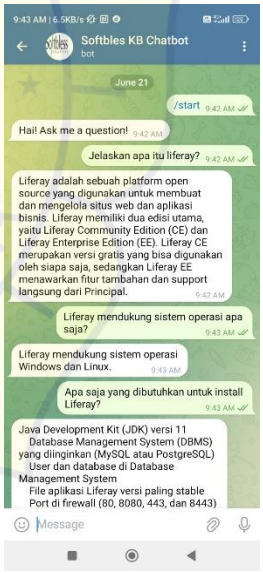
Lampiran 3. Screenshot Chatbot dari beberapa pegawai





Lampiran 4. Source Code Program

```
1.  from django.shortcuts import render, redirect
2.  from django.contrib.auth import authenticate, login, logout
3.  from django.contrib import messages
4.  from .models import Records, Article, ChromaArticle
5.  from .forms import AddRecordForm, AddKBForm
6.  import website.chromadb_util
7.  from website.ArticleChroma import ArticleChroma
8.  import uuid
9.  from django.http import JsonResponse, HttpResponse
10. from django.core import serializers
11. from django.conf import settings
12. from django.contrib.auth.hashers import make_password
13. import mysql.connector
14. import datetime
15. import json
16. import requests
17. from django.views.decorators.csrf import csrf_exempt
18. from telegram import Update
19. from telegram.ext import Application, CommandHandler,
    MessageHandler, filters, ContextTypes
20. import html
21. import re
22. CLEANR = re.compile('<.*?>')
23. from django.contrib.auth.models import User
24. TOKEN = '7468893296:AAEoB_jrkvugkvw7nspkoXlw5DOx9nNXhZI'
25. TELEGRAM_API_URL = f'https://api.telegram.org/bot{TOKEN}/'
26. RECORD_OPEN_API_KEY = 'OPEN_API_KEY'
27. RECORD_TELEGRAM_WEBHOOK_URL = 'TELEGRAM_WEBHOOK_URL'
28. RECORD_TELEGRAM_TOKEN = 'TELEGRAM_TOKEN'
29. def home(request):
30. # Check to see if user logging in
31. if request.method == "POST":
32. username = request.POST['username']
33. password = request.POST['password']
34. # Authenticate
35. user = authenticate(request, username=username,
    password=password)
36. if user is not None:
37. login(request, user)
38. messages.success(request, "Yoh have been logged in")
39. return redirect('home')
40. else:
41. messages.success(request, "There was an error loggin in,
    please try again")
42. return redirect('home')
43. else:
44. return render(request, 'home.html', {})
45. def profile(request):
46. if request.user.is_authenticated:
```

```
47. print(make_password("Softbless3##"))
48. if request.method == 'POST':
49. user = User.objects.get(id=request.user.id)
50. username = request.POST['username']
51. firstname = request.POST['firstname']
52. lastname = request.POST['lastname']
53. email = request.POST['email']
54. newPassword = request.POST['new_password']
55. retypePassword = request.POST['retype_password']
56. if request.POST['new_password'] != '':
57. if request.POST['retype_password'] == '':
58. return JsonResponse({'status': 'error', 'message': "Retype
    Password required"})
59. elif request.POST['new_password'] !=
    request.POST['retype_password']:
60. return JsonResponse({'status': 'error', 'message': "Retype
    Password required"})
61. else:
62. newPassword = make_password(newPassword)
63. else:
64. newPassword = request.user.password
65. if(user.username != username):
66. users = User.objects.filter(username__exact=username)
67. if(users.count() > 0):
68. return JsonResponse({'status': 'error', 'message':
    "Username already exist"})
69. user.username = username;
70. user.firstname = firstname;
71. user.lastname = lastname
72. user.email = email
73. user.password = newPassword
74. user.save()
75. return JsonResponse({'status': 'success', 'message':
    "Updated successfully"})
76. user = request.user
77. return render(request, 'profile/index.html', {'user':
    user})
78. else:
79. messages.success(request, "There was an error loggin in,
    please try again")
80. return redirect('home')
81. def login_user(request):
82. return render()
83. def logout_user(request):
84. logout(request)
85. messages.success(request, "You have been logged out")
86. return redirect('home')
87. def users(request):
88. if request.user.is_authenticated:
```

```
89. users = User.objects.all()
90. return render(request, 'users/index.html', {'users':users})
91. else:
92. messages.success(request, "There was an error loggin in,
    please try again")
93. return redirect('home')
94. def add_user(request):
95. if request.user.is_authenticated:
96. users = User.objects.all()
97. return render(request, 'users/index.html', {'users':users})
98. else:
99. messages.success(request, "There was an error loggin in,
    please try again")
100.  return redirect('home')
101.  def delete_user(request, pk):
102.  if request.user.is_authenticated:
103.  users = User.objects.all()
104.  return render(request, 'users/index.html',
    {'users':users})
105.  else:
106.  messages.success(request, "There was an error loggin in,
    please try again")
107.  return redirect('home')
108.  def update_user(request, pk):
109.  if request.user.is_authenticated:
110.  users = User.objects.all()
111.  return render(request, 'users/index.html',
    {'users':users})
112.  else:
113.  messages.success(request, "There was an error loggin in,
    please try again")
114.  return redirect('home')
115.  def records(request):
116.  if request.user.is_authenticated:
117.  records = Records.objects.all()
118.  return render(request, 'records/records.html',
    {'records':records})
119.  else:
120.  messages.success(request, "There was an error loggin in,
    please try again")
121.  return redirect('home')
122.  def add_record(request):
123.  form = AddRecordForm(request.POST or None)
124.  if request.user.is_authenticated:
125.  if request.method == "POST":
126.  if form.is_valid():
127.  add_record = form.save()
128.  messages.success(request, "Add success")
129.  return redirect('records')
```

```
130.  return render(request, 'records/add_record.html',
   {'form': form})
131.  else:
132.  messages.success(request, "There was an error loggin in,
   please try again")
133.  return redirect('home')
134.  def update_record(request, pk):
135.  if request.user.is_authenticated:
136.  current_record = Records.objects.get(id=pk)
137.  form = AddRecordForm(request.POST or None,
   instance=current_record)
138.  if form.is_valid():
139.  form.save()
140.  messages.success(request, "Update success")
141.  return redirect('records')
142.  else:
143.  return render(request, 'records/update_record.html',
   {'form': form})
144.  else:
145.  messages.success(request, "There was an error loggin in,
   please try again")
146.  return redirect('home')
147.  def delete_record(request, pk):
148.  if request.user.is_authenticated:
149.  record = Records.objects.get(id=pk)
150.  record.delete()
151.  messages.success(request, "Delete Success")
152.  return redirect('records')
153.  else:
154.  messages.success(request, "There was an error loggin in,
   please try again")
155.  return redirect('home')
156.  def configuration(request):
157.  if request.user.is_authenticated:
158.  open_api_key_value = ""
159.  telegram_webhook_url_value = ""
160.  telegram_token_value = ""
161.  if request.method == "POST":
162.  if request.POST['configType'] == 'open_api_key':
163.  try:
164.  open_api_key =
   Records.objects.get(key=RECORD_OPEN_API_KEY)
165.  open_api_key.value = request.POST['open_api_key']
166.  open_api_key.save()
167.  open_api_key_value = open_api_key.value;
168.  except Records.DoesNotExist:
169.  open_api_key =
   Records.objects.create(key=RECORD_OPEN_API_KEY,
   value=request.POST['open_api_key'] )
```

```
170.  open_api_key.save()
171.  open_api_key_value = open_api_key.value;
172.  if request.POST['configType'] == 'telegram_webhook_url':
173.  try:
174.  telegram_webhook_url =
    Records.objects.get(key__exact=RECORD_TELEGRAM_WEBHOOK_URL)
175.  telegram_webhook_url.value =
    request.POST['telegram_webhook_url']
176.  telegram_webhook_url.save()
177.  telegram_webhook_url_value = telegram_webhook_url.value;
178.  except Records.DoesNotExist:
179.  telegram_webhook_url =
    Records.objects.create(key=RECORD_TELEGRAM_WEBHOOK_URL,
    value=request.POST['telegram_webhook_url'] )
180.  telegram_webhook_url.save()
181.  telegram_webhook_url_value = telegram_webhook_url.value;
182.  if request.POST['configType'] == 'telegram_token':
183.  try:
184.  telegram_token =
    Records.objects.get(key__exact=RECORD_TELEGRAM_TOKEN)
185.  telegram_token.value = request.POST['telegram_token']
186.  telegram_token.save()
187.  telegram_token_value = telegram_token.value;
188.  except Records.DoesNotExist:
189.  telegram_token =
    Records.objects.create(key=RECORD_TELEGRAM_TOKEN,
    value=request.POST['telegram_token'] )
190.  telegram_token.save()
191.  telegram_token_value = telegram_token.value;
192.  return JsonResponse({'status': 'success', 'message':
    'Success'})
193.  else:
194.  try:
195.  open_api_key =
    Records.objects.get(key__exact=RECORD_OPEN_API_KEY)
196.  open_api_key_value = open_api_key.value;
197.  except Records.DoesNotExist:
198.  open_api_key_value = "";
199.  try:
200.  telegram_webhook_url =
    Records.objects.get(key__exact=RECORD_TELEGRAM_WEBHOOK_URL)
201.  telegram_webhook_url_value = telegram_webhook_url.value;
202.  except Records.DoesNotExist:
203.  telegram_webhook_url_value = "";
204.  try:
205.  telegram_token =
    Records.objects.get(key__exact=RECORD_TELEGRAM_TOKEN)
206.  telegram_token_value = telegram_token.value;
207.  except Records.DoesNotExist:
```

```
208.  telegram_token_value = "";
209.  return render(request, 'configuration/index.html',
    {'open_api_key': open_api_key_value,
    'telegram_webhook_url': telegram_webhook_url_value,
    'telegram_token' : telegram_token_value})
210.  else:
211.  messages.success(request, "There was an error loggin in,
    please try again")
212.  return redirect('home')
213.  def get_all_knowledge_base(request):
214.  if request.user.is_authenticated:
215.  articles = Article.objects.all()
216.  qs_json = serializers.serialize('json', articles)
217.  return HttpResponse(qs_json,
    content_type='application/json')
218.  else:
219.  messages.success(request, "There was an error loggin in,
    please try again")
220.  return redirect('home')
221.  def knowledge_base(request):
222.  if request.user.is_authenticated:
223.  articles = Article.objects.all().order_by('-
    updated_at').values()
224.  return render(request, 'knowledge_base/index.html',
    {'articles':articles})
225.  else:
226.  messages.success(request, "There was an error loggin in,
    please try again")
227.  return redirect('home')
228.  def add_knowledge_base(request):
229.  form = AddKBForm(request.POST or None)
230.  if request.user.is_authenticated:
231.  if request.method == "POST":
232.  if request.POST['content'] == '':
233.  messages.error(request, 'content is required')
234.  return render(request,
    'knowledge_base/add_knowledge_base.html', {'form': form})
235.  result =
    Article.objects.filter(title__icontains=request.POST['title
    '])
236.  if result.count() <= 0:
237.  kb = form.save(commit=False)
238.  kb.content = request.POST['content']
239.  kb.isFromRedmine = False
240.  if request.POST['enableEmbedding'] == 'on':
241.  kb.isSyncToChatbot = True
242.  kb.rowStatus = True
243.  kb.created_by_user_id = request.user.id
244.  kb = form.save()
```

```
245.  print("kb id: ")
246.  print(kb.id)
247.  # Save colelctions
248.  add_collection_document(kb.id, kb.content)
249.  if request.POST['showProcess'] == 'on':
250.  return redirect('step1', pk=kb.id)
251.  else:
252.  messages.success(request, "Add success")
253.  return redirect('knowledge_base')
254.  else:
255.  messages.error(request, 'Title is exist')
256.  return render(request,
    'knowledge_base/add_knowledge_base.html', {'form': form})
257.  return render(request,
    'knowledge_base/add_knowledge_base.html', {'form': form})
258.  else:
259.  messages.success(request, "There was an error loggin in,
    please try again")
260.  return redirect('home')
261.  def step1(request, pk):
262.  if request.user.is_authenticated:
263.  article = Article.objects.get(id=pk)
264.  cleaned_text =
    website.chromadb_util.clean_text(article.content)
265.  return render(request, 'knowledge_base/step1.html',
    {'article': article, 'cleaned_text': cleaned_text})
266.  else:
267.  messages.success(request, "There was an error loggin in,
    please try again")
268.  return redirect('home')
269.  def step2(request, pk):
270.  if request.user.is_authenticated:
271.  article = Article.objects.get(id=pk)
272.  cleaned_text =
    website.chromadb_util.clean_text(article.content)
273.  doc =
    website.chromadb_util.create_document(website.chromadb_util
    .clean_text(cleaned_text), pk)
274.  docs = []
275.  docs.append(doc)
276.  chunks = website.chromadb_util.split_text(docs)
277.  splitted_contents = []
278.  for documented in chunks:
279.  splitted_contents.append(documented.page_content)
280.  return render(request, 'knowledge_base/step2.html',
    {'article': article, 'cleaned_text': cleaned_text,
    'splitted_contents' : splitted_contents})
281.  else:
```

```
282.  messages.success(request, "There was an error loggin in,
    please try again")
283.  return redirect('home')
284.  def step3(request, pk):
285.  if request.user.is_authenticated:
286.  article = Article.objects.get(id=pk)
287.  cleaned_text =
    website.chromadb_util.clean_text(article.content)
288.  doc =
    website.chromadb_util.create_document(website.chromadb_util
    .clean_text(cleaned_text), pk)
289.  docs = []
290.  docs.append(doc)
291.  chunks = website.chromadb_util.split_text(docs)
292.  splitted_contents = []
293.  embedded_contents = []
294.  open_ai_ef =
    website.chromadb_util.getEmbeddingFunction(open_ai_key=getR
    ecordValue(RECORD_OPEN_API_KEY));
295.  for documented in chunks:
296.  splitted_contents.append(documented.page_content)
297.  embedded =
    open_ai_ef._embed_documents(chunks[0].page_content)
298.  embedded_contents.append(embedded[:1])
299.  return render(request, 'knowledge_base/step3.html',
    {'article': article, 'cleaned_text': cleaned_text,
    'splitted_contents' : splitted_contents,
    'embedded_contents' : embedded_contents})
300.  else:
301.  messages.success(request, "There was an error loggin in,
    please try again")
302.  return redirect('home')
303.  def add_collection_document(id_kb, content):
304.  doc =
    website.chromadb_util.create_document(website.chromadb_util
    .clean_text(content), id_kb)
305.  docs = []
306.  docs.append(doc)
307.  chunks = website.chromadb_util.split_text(docs)
308.  articleChromas = []
309.  my_documents = []
310.  uuids = []
311.  for documented in chunks:
312.  uid = uuid.uuid4()
313.  uuids.append(str(uid))
314.  articleChromas.append(ArticleChroma(uid,
    documented.page_content, metadata={"id_article":
    documented.metadata}))
315.  my_documents.append(documented.page_content)
```

```
316.  collection =
    website.chromadb_util.add_collection(ids=uuids,
    documents=my_documents,
    open_ai_key=getRecordValue(key=RECORD_OPEN_API_KEY))
317.  for articleChroma in articleChromas:
318.  chromaArticle =
    ChromaArticle.objects.create(id_article=id_kb,
    chroma_doc_id=articleChroma.id)
319.  def getRecordValue(key):
320.  value = ""
321.  try:
322.  record = Records.objects.get(key=key)
323.  value = record.value
324.  except Records.DoesNotExist:
325.  value = ""
326.  return value
327.  def view_knowledge_base(request, pk):
328.  if request.user.is_authenticated:
329.  article = Article.objects.get(id=pk)
330.  return render(request,
    'knowledge_base/view_knowledge_base.html', {'article':
    article})
331.  else:
332.  messages.success(request, "There was an error loggin in,
    please try again")
333.  return redirect('home')
334.  def import_kb(request):
335.  if request.user.is_authenticated:
336.  print("on import kb")
337.  print(settings.DATABASES['default']['HOST'])
338.  connection = mysql.connector.connect(
339.  host=settings.DATABASES['default']['HOST'],     # e.g.,
    'localhost'
340.  user=settings.DATABASES['default']['USER'], # e.g.,
    'root'
341.  password=settings.DATABASES['default']['PASSWORD'], #
    e.g., 'password'
342.  database=settings.DATABASES['default']['NAME']  # e.g.,
    'test_db'
343.  )
344.  cursor = connection.cursor()
345.  sql = "SELECT title, content, category_id FROM
    kb_articles"
346.  cursor.execute(sql)
347.  data = cursor.fetchall()
348.  cursor.close()
349.  connection.close()
350.  created = 0
351.  skipped = 0
```

```
352.  for item in data:
353.  print(item[0])
354.  if item[1] == '' or item[1] is None:
355.  skipped +=1
356.  else:
357.  result = Article.objects.filter(title__icontains=item[0])
358.  if result.count() <= 0:
359.  kb = Article.objects.create(title=item[0],
    content=item[1], isFromRedmine=1, isSyncToChatbot=1,
    rowStatus=1, created_by_user_id=request.user.id)
360.  add_collection_document(kb.id, kb.content)
361.  created +=1
362.  else:
363.  skipped += 1
364.  return JsonResponse({'status': 'success', 'created':
    created, 'skipped': skipped})
365.  else:
366.  return JsonResponse({'status': 'error', 'message' : 'you
    are not logged in'})
367.  def delete_knowledge_base(request, pk):
368.  if request.user.is_authenticated:
369.  article = Article.objects.get(id=pk)
370.  chromaArticles =
    ChromaArticle.objects.filter(id_article=pk)
371.  docIds = []
372.  for chromaArticle in chromaArticles:
373.  docIds.append(chromaArticle.chroma_doc_id)
374.  print(chromaArticle.chroma_doc_id)
375.  #Delete from collection
376.  website.chromadb_util.delete_collection(docIds,
    getRecordValue(RECORD_OPEN_API_KEY))
377.  delete_chroma_article(pk)
378.  article.delete()
379.  return JsonResponse({'status': 'success', 'message':
    'Data has been successfully deleted'})
380.  else:
381.  messages.success(request, "There was an error loggin in,
    please try again")
382.  return redirect('home')
383.  def delete_chroma_article(pk):
384.  chromaArticles =
    ChromaArticle.objects.filter(id_article=pk)
385.  deleted_chroma_ids = []
386.  if chromaArticles.count() > 0:
387.  for chromaArticle in chromaArticles:
388.  deleted_chroma_ids.append(chromaArticle.chroma_doc_id)
389.  collection =
    website.chromadb_util.get_collection(getRecordValue(RECORD_
    OPEN_API_KEY))
```

```
390.  collection.delete(ids=deleted_chroma_ids)
391.  chromaArticles.delete()
392.  def ask_bot(request):
393.  print("on ask_bot")
394.  if request.user.is_authenticated:
395.  if request.method == "POST":
396.  if request.POST['question'] != '':
397.  try:
398.  response =
    website.chromadb_util.getFormattedResponse(request.POST['qu
    estion'], getRecordValue(RECORD_OPEN_API_KEY))
399.  return JsonResponse({'status': 'success', 'message':
    response})
400.  except ConnectionError as conn_err:
401.  return JsonResponse({'status': 'error', 'message' :
    conn_err})
402.  except Exception as e:
403.  return JsonResponse({'status': 'error', 'message' : e})
404.  else:
405.  return JsonResponse({'status': 'error'})
406.  return JsonResponse({'status': 'error'})
407.  else:
408.  messages.success(request, "There was an error loggin in,
    please try again")
409.  return render(request, 'home.html', {})
410.  def update_knowledge_base(request, pk):
411.  if request.user.is_authenticated:
412.  current_record = Article.objects.get(id=pk)
413.  form = AddKBForm(request.POST or None,
    instance=current_record)
414.  if form.is_valid():
415.  newRecord = form.save(commit=False)
416.  if current_record.title != request.POST['title']:
417.  result =
    Article.objects.filter(title__icontains=request.POST['title
    '])
418.  if result.count() > 0:
419.  messages.error(request, 'Title is exist')
420.  return render(request,
    'knowledge_base/update_knowledge_base.html', {'form':
    form})
421.  else:
422.  newRecord.title = request.POST['title']
423.  newRecord.content = request.POST['content']
424.  newRecord.updated_at = datetime.datetime.now()
425.  newRecord.save()
426.  delete_chroma_article(pk)
427.  add_collection_document(pk, request.POST['content'])
428.  messages.success(request, "Update success")
```

```
429.  return redirect('knowledge_base')
430.  else:
431.  return render(request,
    'knowledge_base/update_knowledge_base.html', {'form':
    form})
432.  else:
433.  messages.success(request, "There was an error loggin in,
    please try again")
434.  return redirect('home')
435.  def cleanhtml(raw_html):
436.  cleantext = re.sub(CLEANR, '', raw_html)
437.  return cleantext
438.  def send_message(chat_id, text):
439.  print(chat_id)
440.  print(text)
441.  url =
    f'https://api.telegram.org/bot{getRecordValue(RECORD_TELEGR
    AM_TOKEN)}/sendMessage'
442.  payload = {'chat_id': chat_id, 'text': cleanhtml(text),
    'parse_mode': 'HTML'}
443.  response = requests.post(url, json=payload)
444.  print(response.content)
445.  @csrf_exempt
446.  def telegram_bot(request):
447.  print("on telegram bot")
448.  print(getRecordValue(RECORD_OPEN_API_KEY))
449.  if request.method == 'POST':
450.  data = json.loads(request.body)
451.  chat_id = data['message']['chat']['id']
452.  text = data['message']['text']
453.  # Handle the incoming message
454.  if text == '/start':
455.  send_message(chat_id, "Hai! Ask me a question!")
456.  else:
457.  response =
    website.chromadb_util.getFormattedResponse(text,
    getRecordValue(RECORD_OPEN_API_KEY))
458.  send_message(chat_id, response)
459.  return JsonResponse({"status": "ok"})
460.  else:
461.  return JsonResponse({"status": "not ok"}, status=400)
462.  def setwebhook(request):
463.  URL = 'https://softbless-kb-chatbot.my.id/getpost/'
464.  response =
    requests.get(f'https://api.telegram.org/bot{getRecordValue(
    RECORD_TELEGRAM_TOKEN)}/setWebhook?url={getRecordValue(RECO
    RD_TELEGRAM_WEBHOOK_URL)}')
465.  return HttpResponse(response)
```