```python
# Import section
from flask import Flask, request, render_template, make_response, jsonify
import pandas as pd
from tabulate import tabulate
from random import randrange
import datetime
import dateutil.parser
# Import section

# Startup routine
print("### start up ###")
app = Flask(__name__)

# Init Process: System Variables #
intentName = "No intent"
traceResponse = "Response from Flask. Why do you see me?"
# Init Process: System Variables #

# Init Process: Reservation Values #
# Explitcit use of global variables for architectural rapid prototyping
setConfirm = False
reservationID = 666
startDate = datetime.datetime.now()
reservationDays = 0
customerName = "Ninalina"
startDateShortStr = startDate.strftime("%A, %d of %B")
df_init = pd.DataFrame(
    {
        'Start Date Long': startDate,
        'Start Date': startDateShortStr,
        'Days': reservationDays,
        'Customer': customerName
    },
    index=[reservationID])
getReservationID = 0
getDeletePerson = "Not a name"

# Display the initial pandas customer table: df_init
print(tabulate(
    df_init,
    headers="keys",
    tablefmt="psql",
))
# Init Process: Reservation Values #

# Start App, open ports
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=8080, debug=True)


# HTML entry point. This is needed to indicate that the server is running
@app.route("/")
def index():
    # Static Homepage, Indicator
    return render_template("index.html")


# Dialogflow-API
@app.route("/webhook", methods=["GET", "POST"])
def respond():
    # Write all requests from Dialogflow to CSV for debugging
    # Convert the JSON from Dialogflow to a flat-pandas dataframe
    flattenResult()
    # Call main routine for intent specific handling
    intentDetector()
    # The following JSON string is transmitted back to Dialogflow
    return make_response(jsonify(results()))


```

```
70  # Finalize the response string from flask to Dialogflow
71  def results():
72      return {"fulfillmentText": traceResponse}
73
74
75  # Converting from JSON to Pandas table format
76  def flattenResult():
77      # Table is global for debugging
78      global pd_norm
79
80      print("### flatten the json ###")
81      # Get the request from Dialogflow and translate to table format
82      pd_norm = pd.json_normalize(request.json)
83
84      # Show the column names for debugging reasons
85      print(pd_norm.columns)
86
87      # Append the normalized result from Dialogflow to a CSV table for trace/debugging
88      pd_norm.to_csv("fromDialogflow_NORM.csv", mode="a", header=True)
89
90      return True
91
92
93  # The response from Dialogflow is based on a defined intent. The intent is detected here
94  def intentDetector():
95      # The content of the received intent is
96      # 1. Normalized into pd_norm
97      # 2. Stored in global variables for better debugging
98      global setConfirm
99      global reservationID
100     global startDate
101     global reservationDays
102     global customerName
103     global df_init
104     global getReservationID
105     global getDeletePerson
106
107     # The name of the intent is extracted from the corresponding column
108     intentName = pd_norm["queryResult.intent.displayName"].values[0]
109     print("### Detected intent is " + intentName)
110
111     # When receiving the intent: CustomerHelp, there's nothing to do for the flask
        backend. It will be handled by Dialogflow internally. No database operation needed
112     if intentName == "CustomerHelp":
113         # Forward the suggested intent from Dialogflow back
114         passOnly(intentName)
115
116     # When receiving the intent: ReservationCancel, the corresponding line in the
        database will be removed/dropped.
117     if intentName == "ReservationCancel":
118         getDropID = int(pd_norm["queryResult.parameters.number"].values[0])
119         df_init.drop(getDropID, inplace=True)
120         readPandas()
121
122         passOnly(intentName)
123
124     # When receiving the intent: ReservationGet, extract the reservation number
        queryResult.parameters.number and send back the underlying reservation
125     if intentName == "ReservationGet":
126         # Extract the reservation number
127         getReservationID = int(
128             pd_norm["queryResult.parameters.number"].values[0])
129
130         # Call routine for reservation retrievement
131         passGetReservationID(intentName)
132
133     # When receiving the intent: ReservationBook, extract the day for check-in and the
        number of days for the customer stay
134     if intentName == "ReservationBook":
```

```
135             setConfirm = False
136             reservationID = randrange(999)
137             startDate = pd_norm["queryResult.parameters.date-time"].values[0]
138             startDate = startDate[0]
139             # The date can come in the form of a list or a dictionary. When Dialogflow
                sends a dict, this workaround is used
140             if isinstance(startDate, dict):
141                 startDate = startDate.get('date_time')
142 
143             # Check the raw date format
144             print("Date Debug: " + startDate)
145             startDate = dateutil.parser.parse(startDate)
146             reservationDays = int(
147                 pd_norm["queryResult.parameters.number"].values[0])
148 
149             # Check the translated date format
150             print("The Start is " + startDate.strftime("%m/%d/%Y") + " and days " +
151                   str(reservationDays))
152             passOnly(intentName)
153 
154         # When receiving the intent: "ReservationBook - yes", no database operation is
            made. A flag gets set when the customer confirms the reservation
155         if intentName == "ReservationBook - yes":
156             passOnly(intentName)
157             setConfirm = True
158 
159         # When receiving the intent: "ReservationBook - cancel", no database operation is
            made. The customer has decided to not confirm the reservation
160         if intentName == "ReservationBook - cancel":
161             setConfirm = False
162             passOnly(intentName)
163 
164         # When receiving the intent: CustomerDelete -> (1.) The name of the customer is
            retrieved
165         if intentName == "CustomerDelete":
166             getDeletePerson = pd_norm["queryResult.parameters.person.name"].values[
167                 0]
168             print("### Deletion of customerbase: " + getDeletePerson)
169 
170             passOnly(intentName)
171 
172         # When receiving the intent: "CustomerDelete - no", the customer stays in the
            pandas database. No further action
173         if intentName == "CustomerDelete - no":
174             passOnly(intentName)
175 
176         # When receiving the intent: "CustomerDelete - yes", all rows which contain this
            name will be dropped
177         if intentName == "CustomerDelete - yes":
178             # The .drop won't work with a non index call. Instead subsetting is used
179             df_init = df_init[df_init.Customer != getDeletePerson]
180             # Check if the customer is really deleted
181             readPandas()
182             passOnly(intentName)
183 
184         # Dialogflow default intent
185         if intentName == "Default Fallback Intent":
186             passOnly(intentName)
187 
188         # Dialogflow default intent
189         if intentName == "Default Welcome Intent":
190             passOnly(intentName)
191 
192         # When receiving the intent: "ReservationBook - yes - custom", the customer has
            confirmed the reservation and gave his/her name. In this case, the entry is made
            to the database
193         if intentName == "ReservationBook - yes - custom":
194             passReservationID(intentName)
195             # Retrieve the customer name
```

```
196             customerName = pd_norm["queryResult.parameters.person.name"].values[0]
197             # Add an entry with the booking to the database
198             writePandas()
199
200         return True
201
202
203     # Helper-Function: The intent fulfillment message is passed through
204     def passOnly(intentName):
205         global traceResponse
206         print("handling " + intentName)
207         traceResponse = pd_norm["queryResult.fulfillmentText"].values[0]
208
209
210     # Helper-Function: The backend-generated reservation ID is attached to the fulfillment
211     def passReservationID(intentName):
212         global traceResponse
213         print("handling " + intentName)
214         # Add the reservation ID, so the customer can always ask for the status
215         traceResponse = pd_norm["queryResult.fulfillmentText"].values[
216             0] + ": " + str(reservationID)
217
218
219     # Helper-Function: The reservation ID is looked up in the pandas database. The
        customer recieves the information for his/hers reservation
220     def passGetReservationID(intentName):
221         global traceResponse
222         # The reservation is for 'Customer' and starting from 'Start Date' for 'Days' days
223         print("handling " + intentName)
224         res = 'The reservation is for ' + df_init.loc[
225             getReservationID, 'Customer'] + ' and starting from ' + df_init.loc[
226                 getReservationID, 'Start Date'] + ' for ' + str(
227                     df_init.loc[getReservationID, 'Days']) + ' days'
228         traceResponse = pd_norm["queryResult.fulfillmentText"].values[0] + res
229
230
231     # Helper-Function: Add the booking information as a row to the database (df_init)
232     def writePandas():
233         global setConfirm
234         global reservationID
235         global startDate
236         global reservationDays
237         global customerName
238         global df_init
239         print("Routine writePandas")
240         startDateShortStr = startDate.strftime("%A, %d of %B")
241         df = pd.DataFrame(
242             {
243                 'Start Date Long': startDate,
244                 'Start Date': startDateShortStr,
245                 'Days': reservationDays,
246                 'Customer': customerName
247             },
248             index=[reservationID])
249         df_init = df_init.append(df)
250
251         readPandas()
252
253         return True
254
255
256     # Helper-Function: Print the complete database with all entries to the console in a
        human readable format
257     def readPandas():
258         print(tabulate(
259             df_init,
260             headers="keys",
261             tablefmt="psql",
262         ))
```

```
263         # Print out the structure of the database
264         print("### Structural integrity")
265         print(df_init.dtypes)
266         return True
267 
```