
ORACLE®

Oracle Intelligent Bots Advanced Training

Design Practices, Hints & Tips

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Topic Agenda

1. Bot Scopes
2. Custom Component Design
3. Custom Component Response
5. Custom Component Error Handling
6. Managing User Profiles
7. Strategies for unlocking users

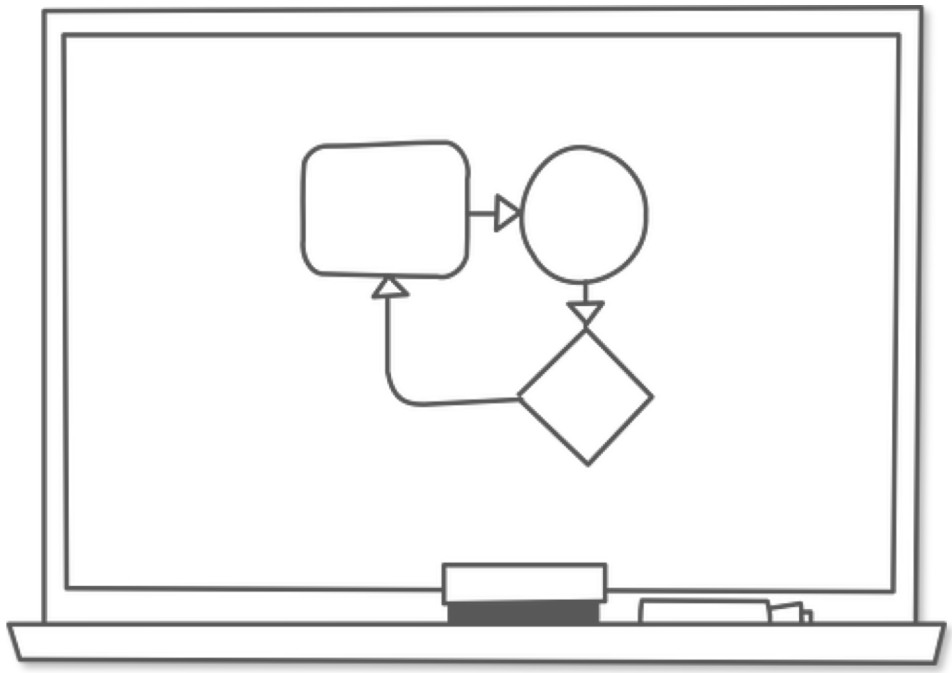
Design Practices, Hints & Tips

Bot Scopes

Bots with a **narrow scope** enjoy a **reduced error surface**

Design Recommendation

- Start small & get it working
- Develop the bot for an objective
  - Avoid building monolithic bots
  - Focus on a task
- Before you start, be clear on
  - Conversation flows
  - Intents
  - Entities



Think of Custom Component Services as the equivalent of **libraries** in Java

Design Custom Component Services as Libraries

- Build with resuse in mind
- Share helper classes and infrastructure
  - E.g. share implementation to read/write to Oracle Mobile Storage or DB
- Defines the scope and size of component service

Custom Component Naming Conventions

- Choose a descriptive component name
  - A good component name describes what the component does
  - Do the same for input-parameter names and action strings
- Use a prefix to ensure the component name is unique
  - JavaScript doesn't know of name spaces
  - Expect your component to be used with 3rd party components

Align Custom Component With System Components

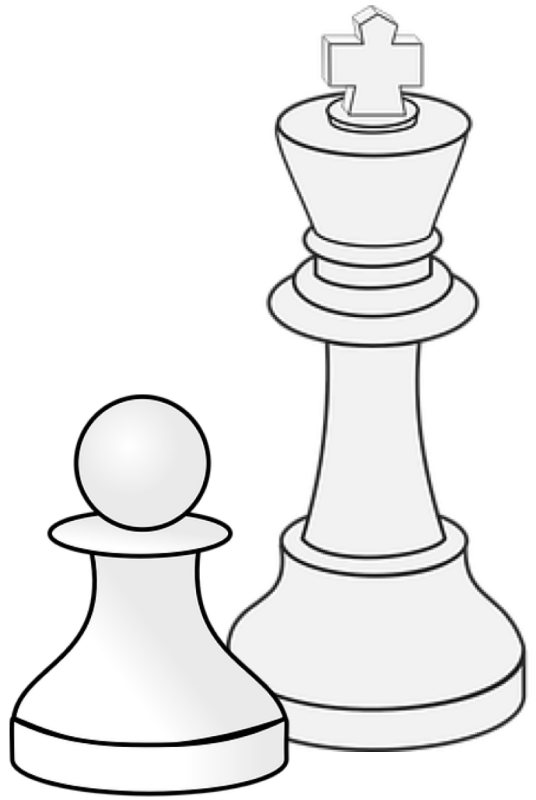
- Add keepTurn property if component is read-only
- Use "variable" property If there is a single context variable to update
- Use actions
  - Supported action strings returned with "transition"
  - Name action sensibly

Design Practices, Hints & Tips

Custom Component Responses

Custom Component Response Strategies

- Two Broad Strategies
  - Render Response to Messenger Client
    - Use Conversation Message Model
    - Handle complete task
  - Save Data in Context Variable
    - Render data using CRC component



Writing Data to a Context Variable

- **PRO**
  - Smaller response payload
  - Data saved in context variable until variable is reset or dialog flow is exit
  - Data can be auto-translated when displayed by system components
  - Data can be used with message bundles

- **CON**
  - No option available to describe the data structure to bot designers
    - Makes it hard to share the component
  - Requires bot designer to know how to use the component
    - E.g. need to create context variable

Writing Response to Message Channel

- **PRO**
  - No need to document response data structure
  - Custom component developers have full control over the rendered UI
  - Component can be designed as "black box" for easy reuse
- **CON**
  - Component response is more verbose compared to when just sending data
  - Data objects cannot be cached
    - Information needs to be re-queried if needed more often
  - Multi-language translation must be handled by the component

Be Mindful when saving data in context variables**. A bot is not a database**

Image courtesy of Farbentek at FreeDigitalPhotos.net

Design Practices, Hints & Tips

Custom Component Error Handling Practices

As a minimum, **custom components should handle** internal **errors** and return 'error' action

Error Handling Practice: A Suggestion

- Define custom set of error actions for custom components to use
  - authenticationError
  - applicationError
    - Action string returned for all kinds of http-4** errors
    - Action string returned for errors in the custom component logic
  - serverError
- Enforce use of custom error message variable
  - conversation.variable('_errorMesssage', <the message>);

If you need to **display error messages** to the user, display them **in the user language**

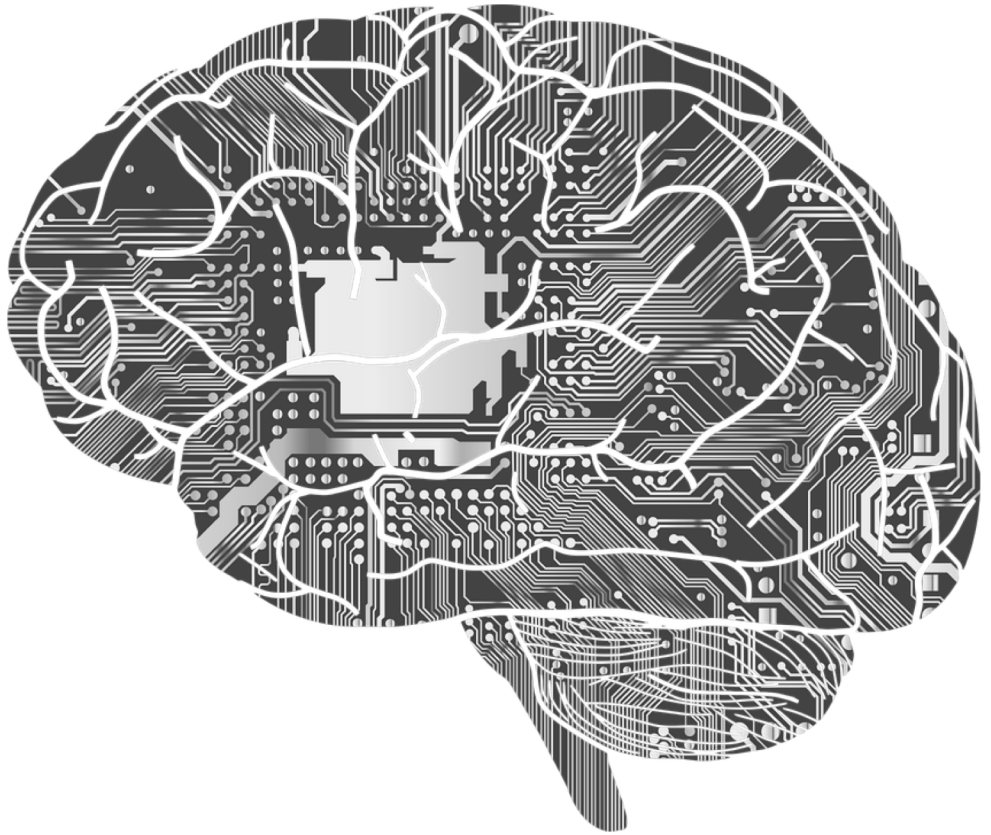
Design Practices, Hints & Tips

Managing User Profiles

A **user profile** presents the **memory** an application has about a user

Where to Save User Information ?

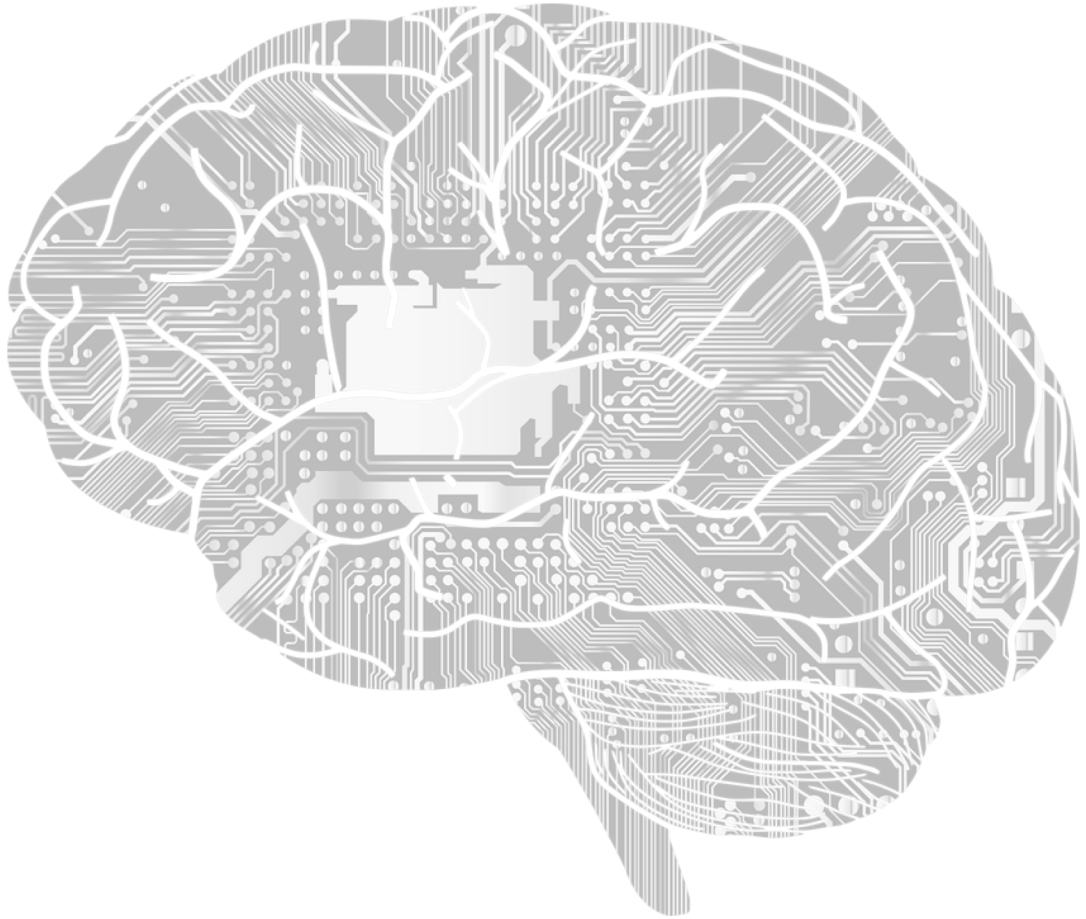
Long-Term Memory



Short-Term Memory

Options

- Bot Options
  - Context variable
  - User scope variable
- Oracle Mobile Cloud Options
  - Oracle Mobile Database
  - Oracle Mobile Storage
  - Remote Service



Bot User Scope Variable

- Information persisted in Cloud Database
- Information saved for specific bot and channel
  - Accessible across user – bot conversations and sessions
- Cannot be shared with other applications or bots

Creating & Reading User Scope Variables

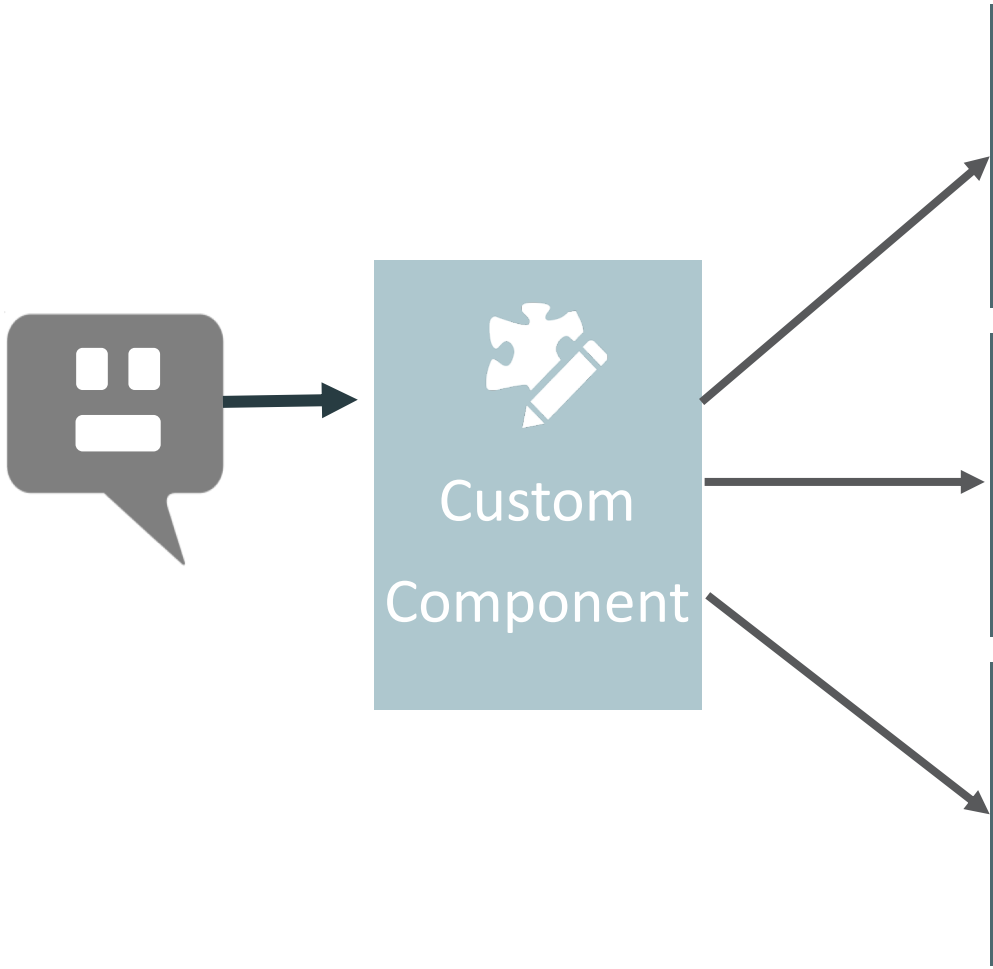
- **Write**

```
setUserInformation:
  component: "System.SetVariable"
  properties:
    variable: "user.preferredLocale"
    value: "de"
```

- **Read**

```
readUserInformation:
  component: "System.SetVariable"
  properties:
    variable: "profile.locale"
    value: "${user.preferredLocale}"
```

Oracle Mobile Cloud Options

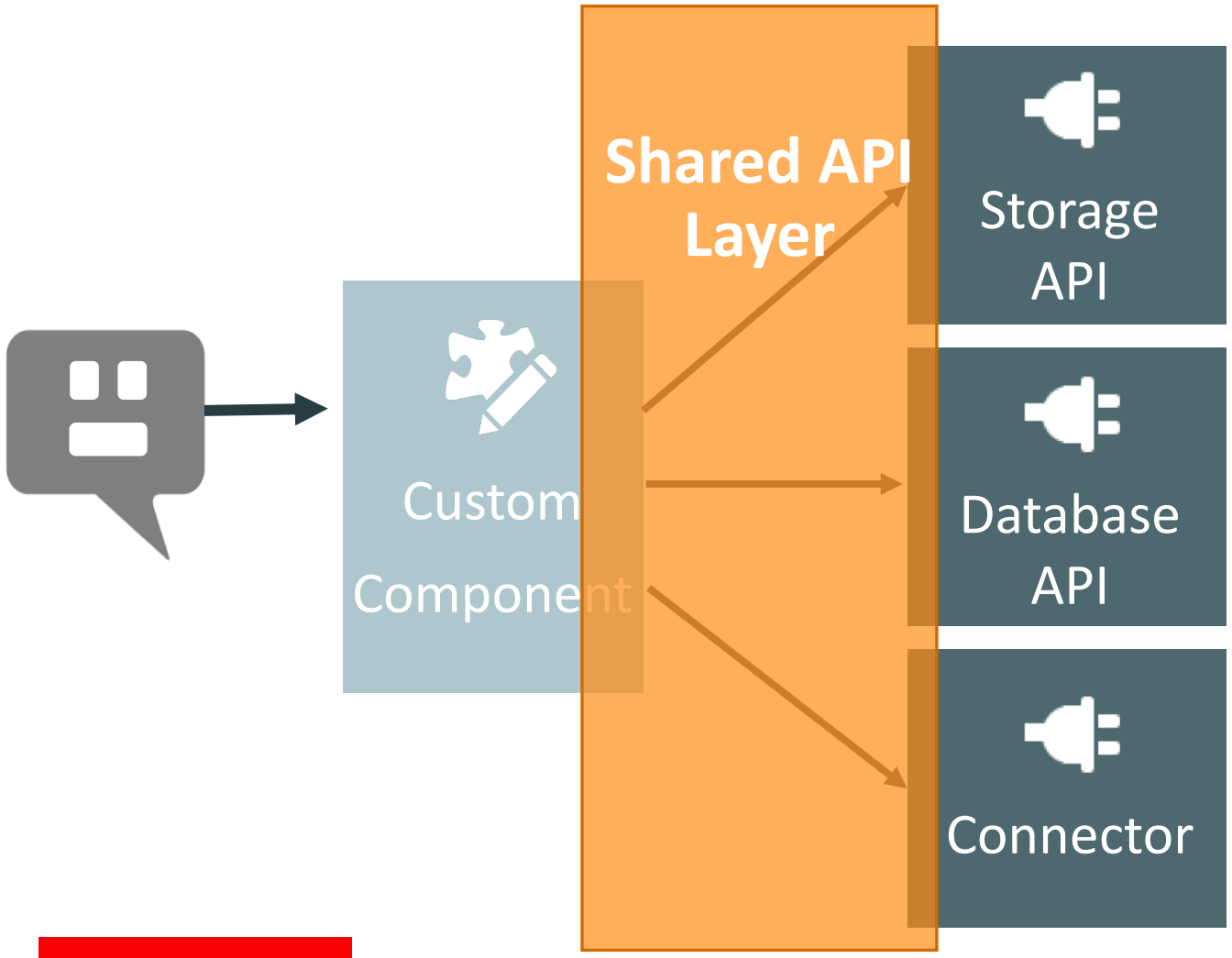


- Write content as JSON object to storage
- Object saved in Mobile Cloud database
- JSON object can be set to context variable
- Content can be encrypted before saving
- External edit possible if no encryption

- Write content into database table
- Create, update, delete option
- Profile information shared across apps using Oracle Mobile Cloud

- Access to remote REST or SOAP service
- Integrates on premise solutions
- Allows user profiles to be shared across apps and environments

Oracle Mobile Cloud Options



- Write content as JSON object to storage
- Object saved in Mobile Cloud database
- JSON object can be set to context variable
- Content can be encrypted before saving
- External edit possible if no encryption

- Write content into database table
- Create, update, delete option
- Profile information shared across apps using Oracle Mobile Cloud

- Access to remote REST or SOAP service
- Integrates on premise solutions
- Allows user profiles to be shared across apps and environments

Design Practices, Hints & Tips

Strategies for unlocking users

A bot is no prison. Allow users to escape.

Strategies

- Use list or button choices for questions with a limited set of answers
  - Clear guidance is better than user's guessing
- "maxPrompts" property
  - Uses cancel action transition after max. number of failed input attempts
- Cancel button on list and card UI
  - Allows users to exit a conversation
- Use Instant Apps to simplify user input
- Configure and detect "exit" words in user input
  - Custom component or Apache FreeMarker expressions required

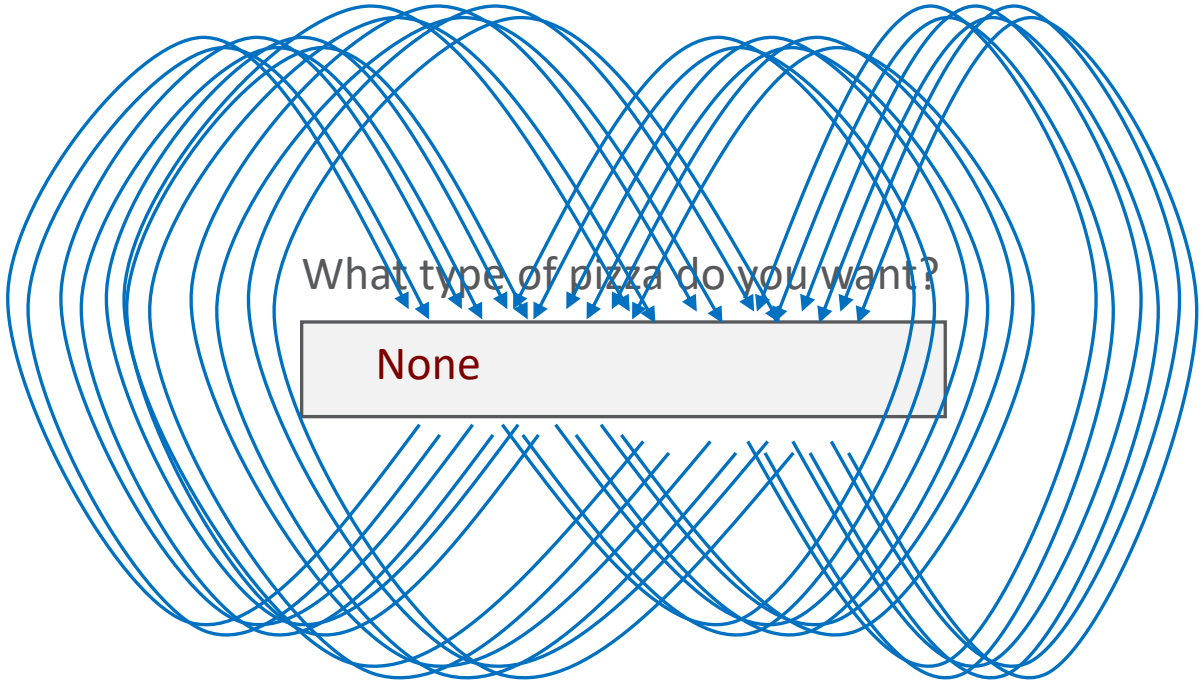
maxPrompts

- Default input component behavior is to keep a user in a loop until she provides a valid input
  - For variables of entity type
- maxPrompts property allows bot designers to set a maximum count of failed input attempts
  - Component triggers ***cancel*** action if max count is exceeded
  - Bot designer to handles ***cancel*** action in flow
  - User is *unlocked* when **cancel** action is triggered

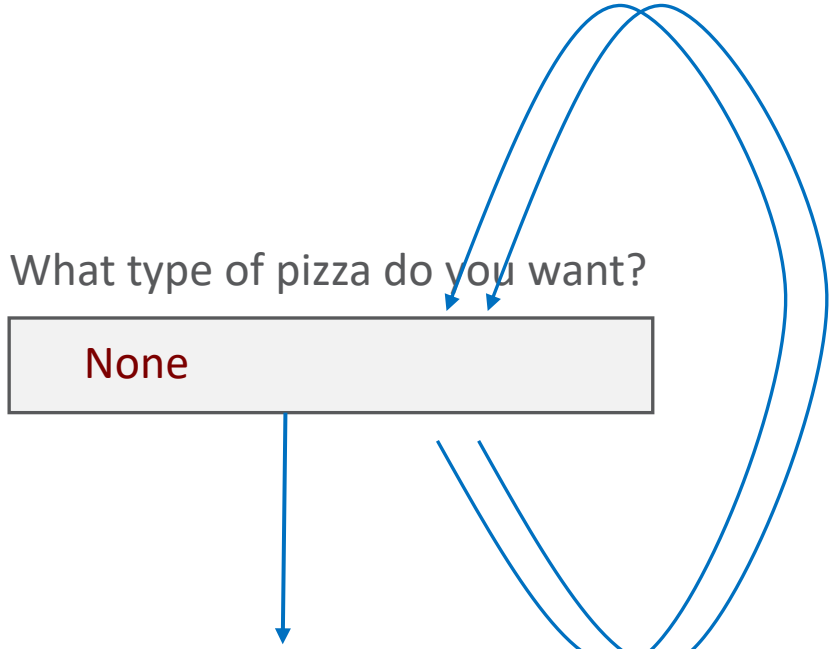
Input Validation

With and Without maxPrompts property set

Component: System.Input
properties:
variable: "entity_based_variable"



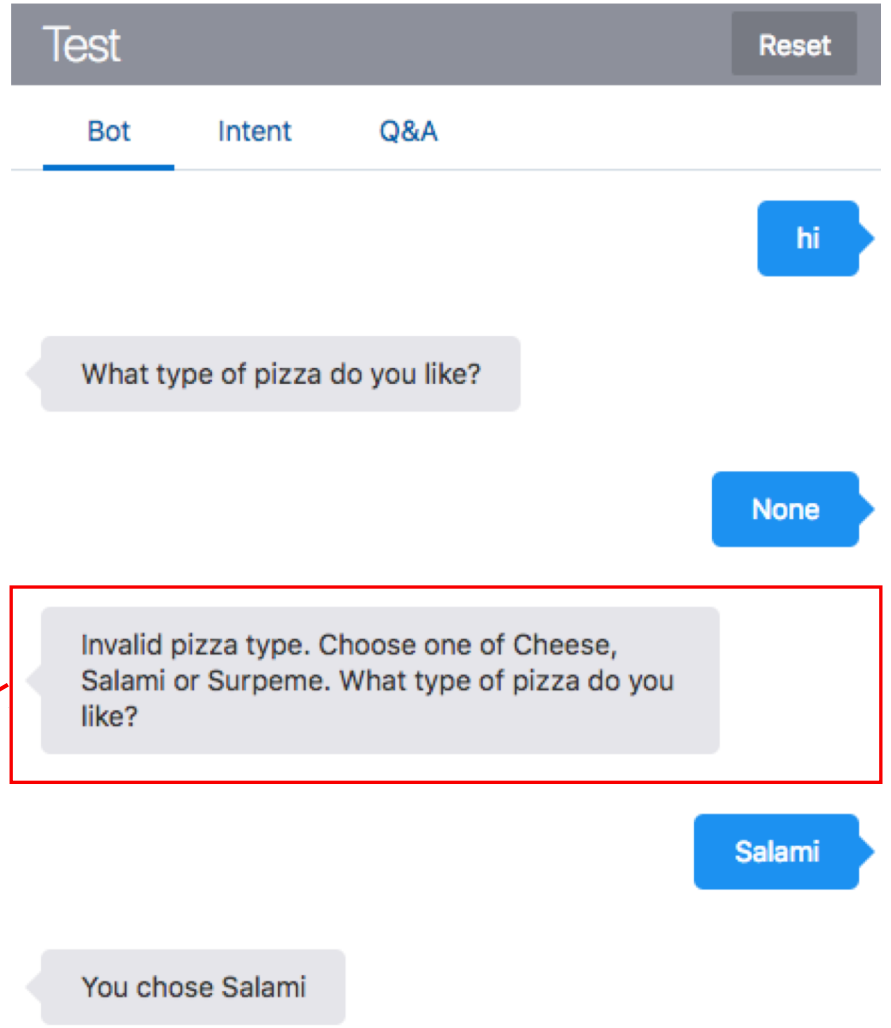
Component: System.Input
properties:
variable: "entity_based_variable"
maxPrompts: 3



Show Different Prompts

- Show different prompt after failed user input
- *system.invalidUserInput* variable set to true after failed input attempt
- Use Apache FreeMarker <#if > </#if> expression to conditionally show prompts

```
askPizzaType:
  component: "System.Text"
  properties:
   prompt: "<#if system.invalidUserInput == 'true'>Invalid pizza type.
                Choose one of Cheese, Salami or Surpeme. </#if>
               What type of pizza do you like?"
   variable: "pizza"
   transitions:
     actions:
       cancel: "provideHelp"
```

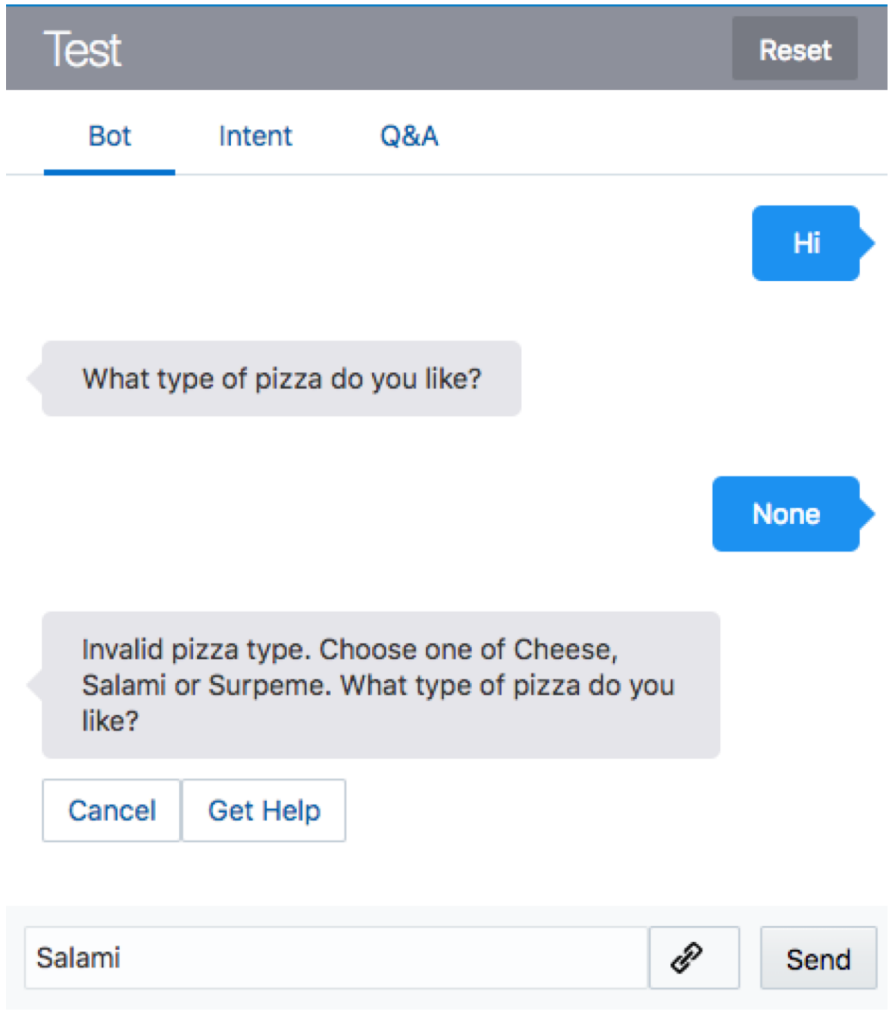


A Better User Experience

Show A Way Out

- Show choice of possible actions after failed input attempt
- Use CR component instead of System.Text
  - Don't set maxPrompts value
- Use postback action to allow users to escape validation loop
  - Global actions
  - Use CR *onInvalidUserInput* property to display actions after first failed input attempt

A Better User Experience



Unlock user …

or take order …

A Better User Experience

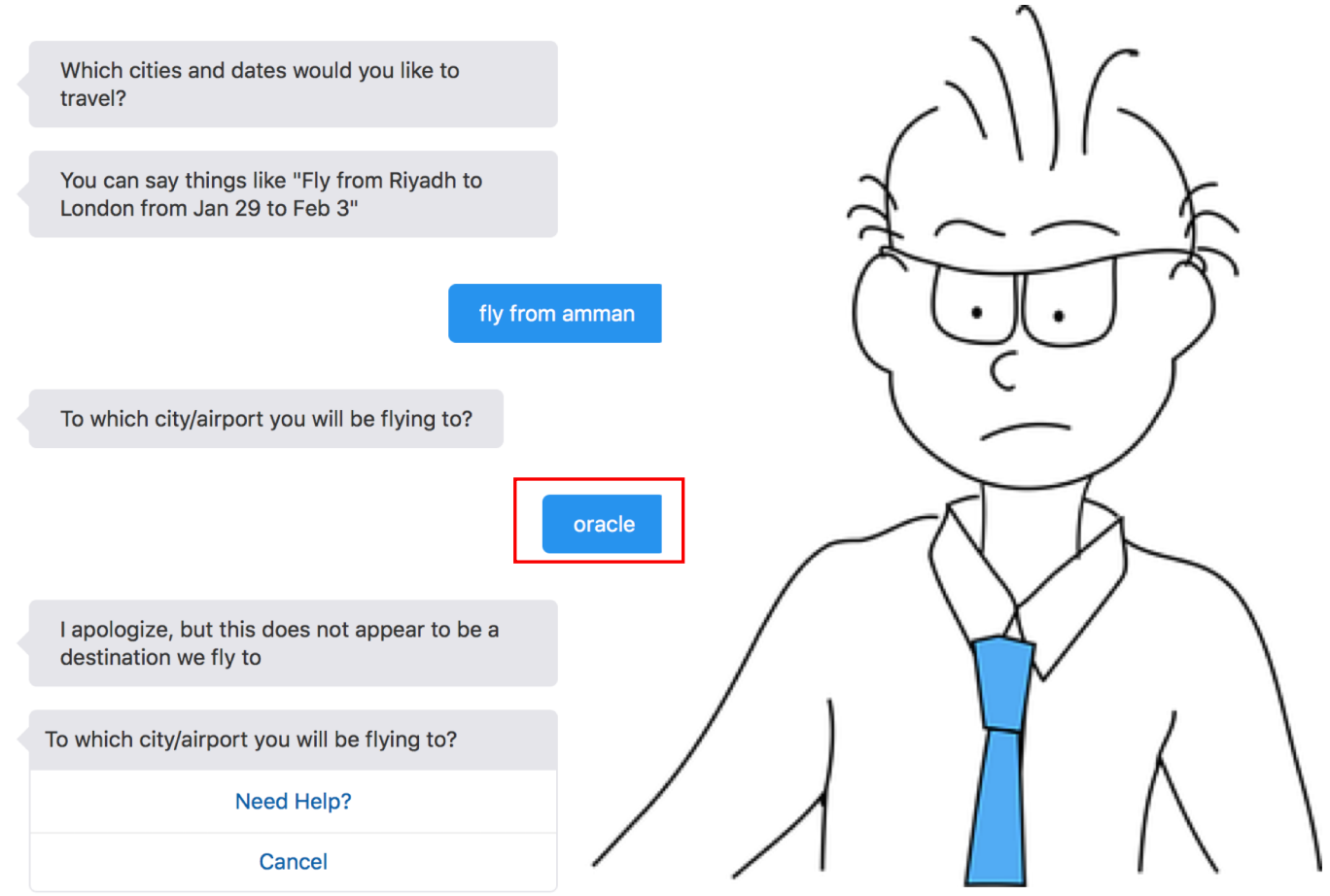
```
askPizzaType:
  component: "System.CommonResponse"
  properties:
    processUserMessage: true
    variable: "pizza"
    metadata:
      responseItems:
      - type: "text"
        text: "<#if system.invalidUserInput == 'true'>Invalid pizza type.
               Choose one of Cheese, Salami or Surpeme. </#if>
               What type of pizza do you like?"
    globalActions:
      - label: "Cancel"
        type: "postback"
        visible:
          onInvalidUserInput: true
        payload:
          action: "doCancel"

      - label: "Get Help"
        type: "postback"
        visible:
          onInvalidUserInput: true
        payload:
          action: "getHelp"
  transitions:
    actions:
      doCancel: "handleCancel"
      getHelp: "provideHelp"
```

Take order …

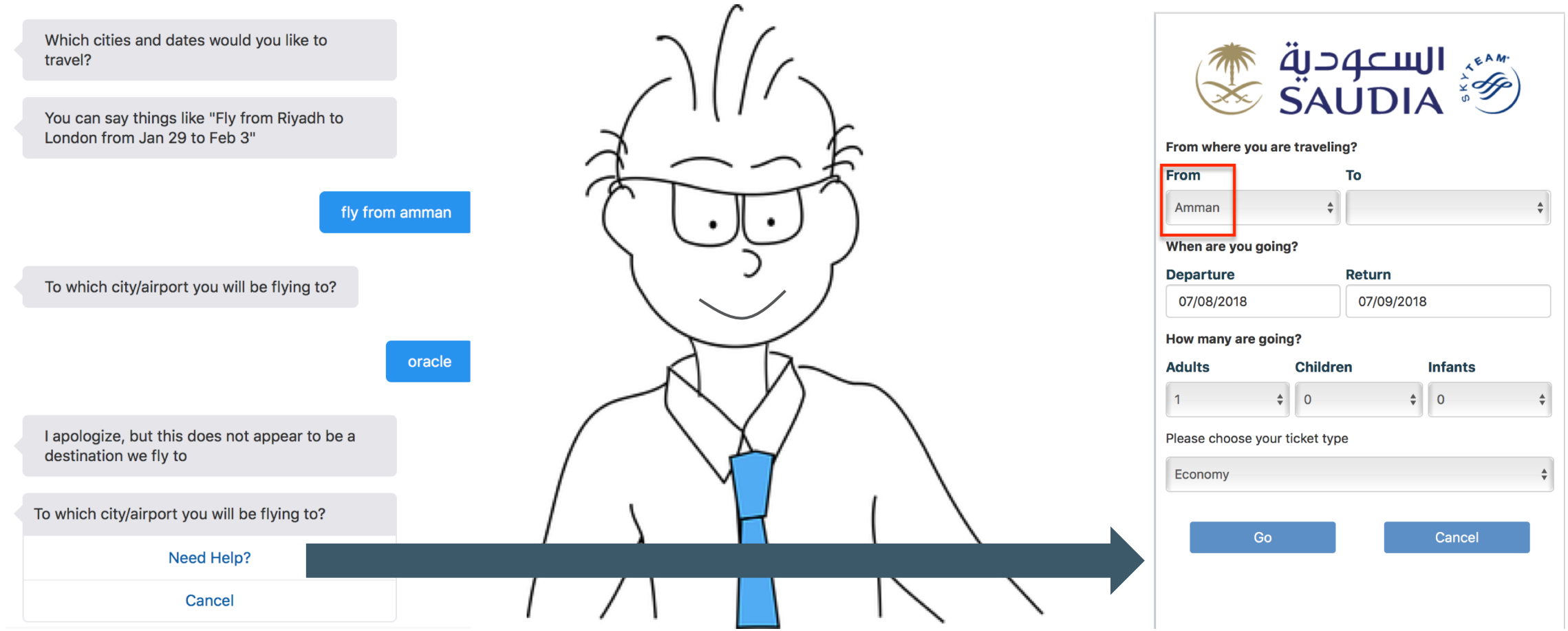
or unlock user …

Handle user choice

Use Instant Apps to Help Users

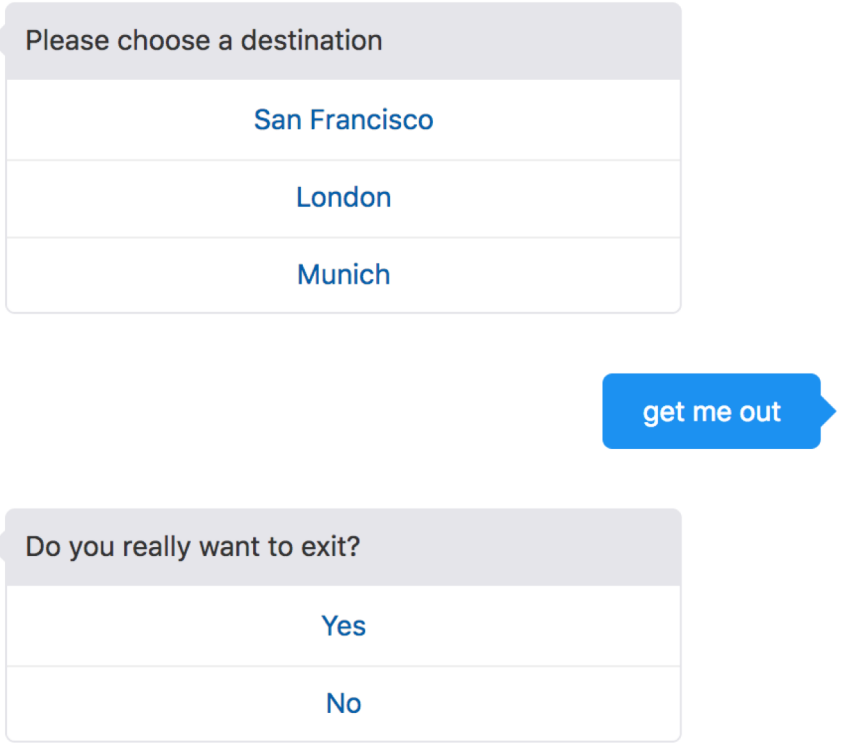


Use Instant Apps to Help Users

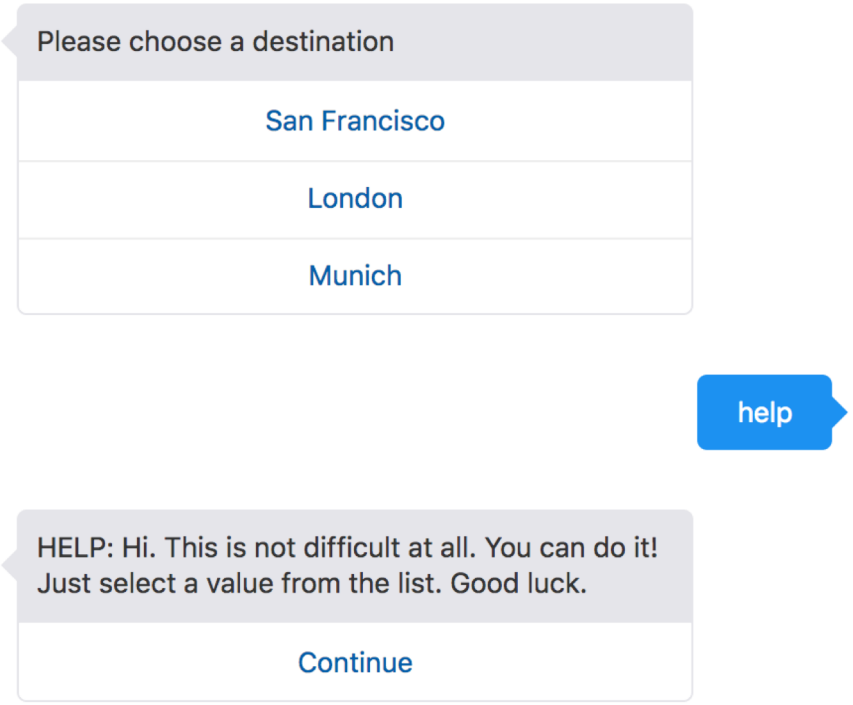


Custom Component to Handle Stop and Help Keywords

Stop Keywords



Help Keywords

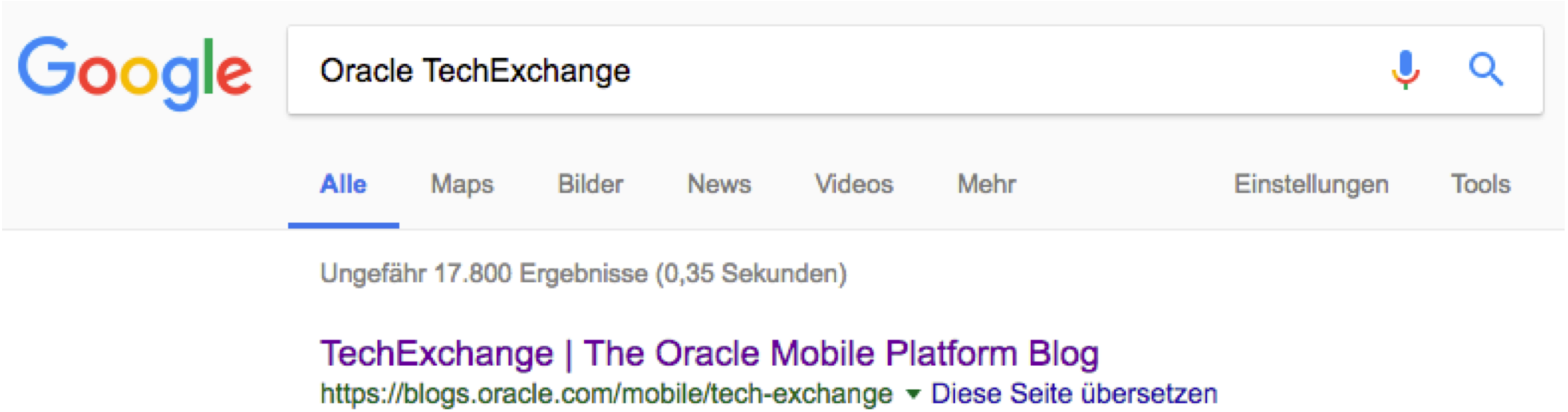


Design Practices, Hints & Tips

Find More of These

More Technical Help

https://blogs.oracle.com/mobile/tech-exchange



Advanced Bot Training Hands-On

Lab 9: Unlocking users in a bot conversation

Integrated Cloud

Applications & Platform Services


ORACLE®