TANDEM-APP

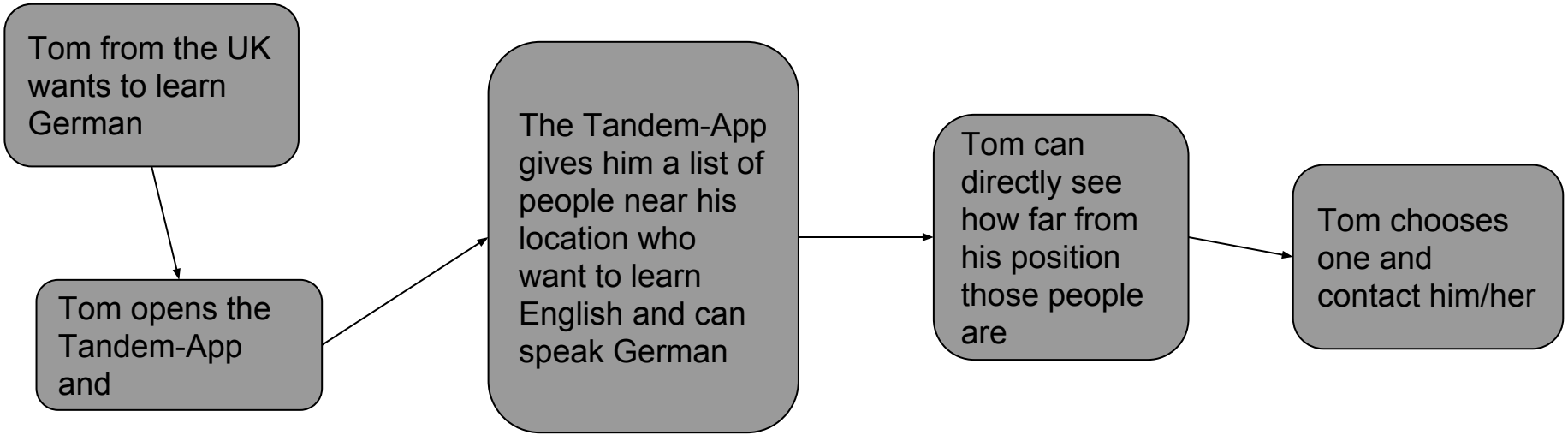
Application Development for Mobile and Ubiquitous Computing

Salohy Miarisoa, Ljupka Titizova
Dresden, December 2017

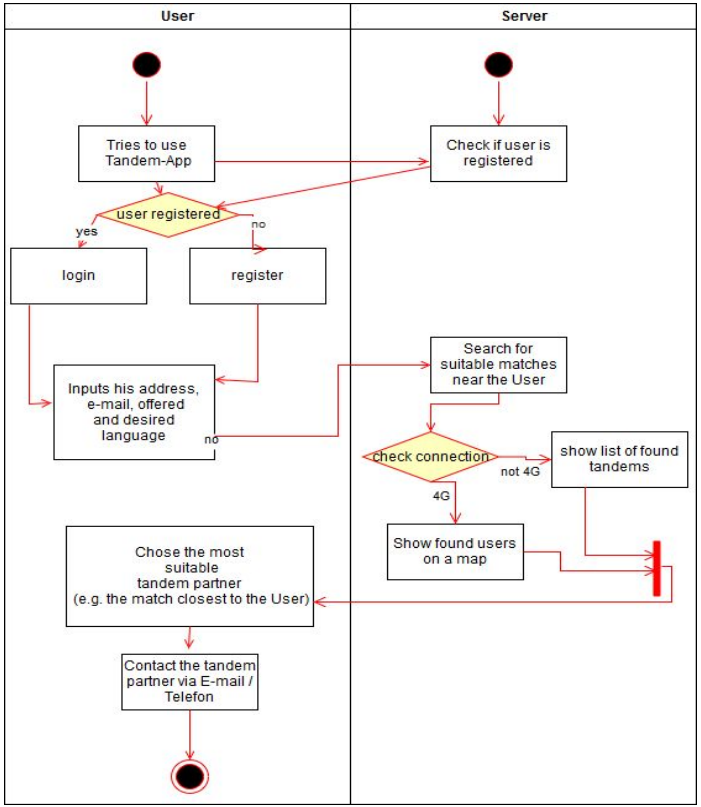
AGENDA

- Scenario
- Technologies
- Adaptation and Context
- Strategies
- Current state
- What’s next

Scenario


Tom from the UK wants to learn German
Tom opens the Tandem-App and
The Tandem-App gives him a list of people near his location who want to learn English and can speak German
Tom can directly see how far from his position those people are
Tom chooses one and contact him/her

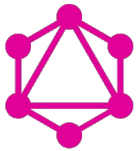
Scenario through Activity Diagram

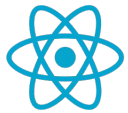


Technologies

Demand-Driven Architecture

Multiplatform





Adaptation: Application Data

Context: Network, Backend

Adaptation:

- As far as possible, getting minimum information from server to minimize payload (Demand Driven Architecture).
- Load x nearest matches while good connection, Load y nearest matches while bad connection (y < x). Get more when the user scroll down.

Adaptation: Application Data

The type of connection is checked on the client side using “NetInfo” from react-native

```
NetInfo.getConnectionInfo()
```

Depending on the connection, the client requests more or less data to the server.

Adaptation: Application Data

While checking the connection, if null is returned, the application is in offline mode. The client will not send requests directly to the server but to the cache.

Apollo offers a cache-feature called “InMemmoryCache”, which saves all previous requested data.

Adaptation: Map

Context: Network, Location, nearby Person

Adaptation:

- Only display Map if the connection is good (eg: 4g) or if the user wants it explicitly, otherwise a list of nearby Tandem is given as Text.
- User Location is detected on the client-side (latitude, longitude)

  ```
  navigator.geolocation.getCurrentPosition()
  ```

- Server searches entries near the client position

Adaptation: Map

```
Tandem.find({
        "languages.offer": offer,
        "languages.search": search,
        "location": {
            $near: [latitude, longitude],
            $maxDistance: 6
        }
})
```

MongoDb uses $near and $maxDistance to find entries near the user.

Sources of Context

1. Database
2. GPS - Google
3. User Input

Summary of Strategies

Challenges

1. Connectivity
2. Offline
3. Usability

Strategies

1. On Demand Data, check of connection
2. Use of cache on the client side
3. Material Design

Current state

Server

1. Set up
2. Geo-Location
3. Accept request from client
4. Return data desired by client

Client

1. Set up
2. Connect to server
3. Display Map with nearby Tandem
4. Display Login
5. Get nearby searched location from server
6. check connection and display the corresponding view

What's next?

Server

1. User Authentication
2. User (Login, register)

Client

1. combination of Geolocation and user
2. Offline challenge
3. Design
4. Lazy evaluation

Screenshot of current states