Mixidea UI構築に必要なディベートの基礎知識

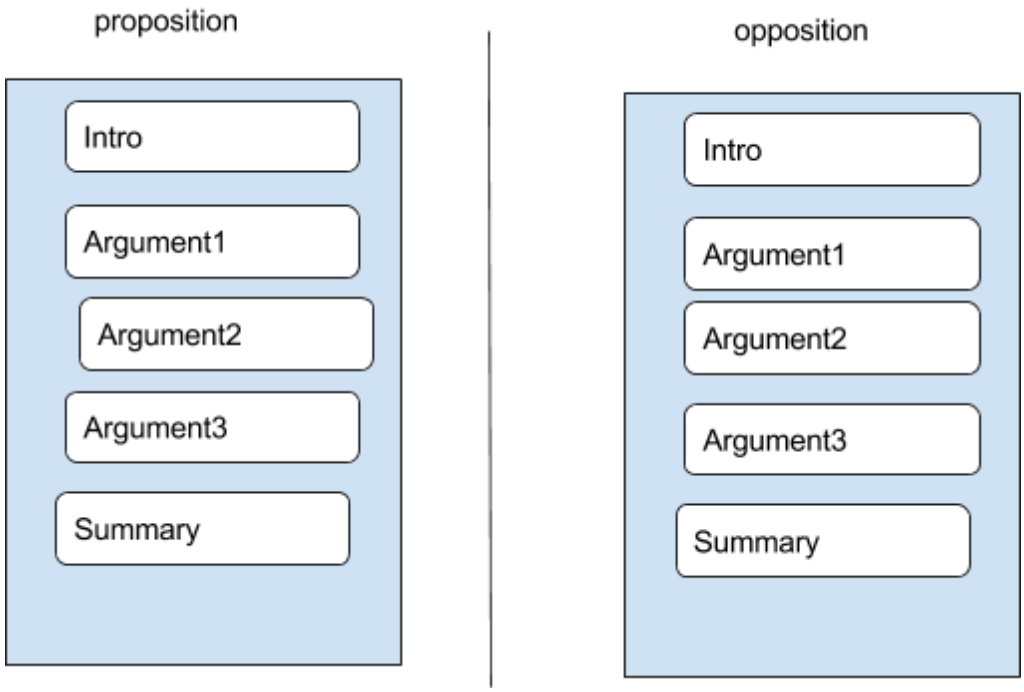
＜目的および構成＞

このドキュメントでは、

＜ディベートの論理構成＞

ディベートには様々なスタイルがあるが、この論理構成は全てのスタイルにおいて共通であり、WrittenDebate(書き込み型のディベート)では、この形式でディベーターがドキュメントそ皆でつくりあげていくような形ですすめられる。

ディベートの理論構成は論文の構成に似ている。ディベートの用語を論文形式にすると次のようになる。



Intro:

proposition側は、議論の定義や社会背景などを説明する(definitionという)

例：「タバコ禁止」という論題の場合だと、「日本でのことか？」「禁止するのは公共領域で？自宅でも禁止？　会社では？」といった、論題だけでは明確でない点を明確化する。

また、チームとしてどういうことを明確にしたいか(team stance)も説明する。

opposition 側もチームの全体の方向性(team stance)を話す。

Argument

argumentは次の順番で話す。

principle : Argumentの本質を一言で表す。 sign postともいう。

reason : 理由

example : 例

principle 再度確認のために、本質を説明

summary

両者のディスカッションを全てふまえて、どういう観点が重要で、何を基準に勝敗を決めるべきか説明する。

反論(refute)のしかたについて

反論は、相手のArgumentのReasonが間違っていることをいい、それに関連するexampleをだす。反論は必ずArgumentのprincipleにそった内容にしなければならない。

相手のArgumentが、「タバコは健康の被害が大きい」というArgumentを出してきた場合

「反論となる意見」

→健康の被害は限定的で、実際に肺がんになったひとのうち、タバコが原因となっているものより、排気ガスなどによる環境のほうが大きい原因である。

「反論ではない意見」

→健康よりも個人の自由のほうが重要である。

→「個人の自由」と「健康」は別の観点であり、「健康被害」というArgumentの反論にはならない。

反論の反論(refute back)について

反論にたいして、反論の反論をしないと、「Argumentは反論されて成り立っていなくて無効」ととらえられるので、必ず反論に対する反論はされる。

Summary

それぞれのArgumentで何が重要なポイントだったのか、お互いのチームが何をいっていたのかをサマリーし、どれがもっとも重要な争点になったのか(impactが大きいものは何か？)を説明し、ジャッジへの判断ポイントを提供する。

コメント、ジャッジのArgumentにたいするフィードバックについて

Argumentが、principle, reason, exampleの全てがそろっていて妥当かどうかを精査する。

どれかがかけていたりすると、そのArgumentはなりたっていないと判断する。

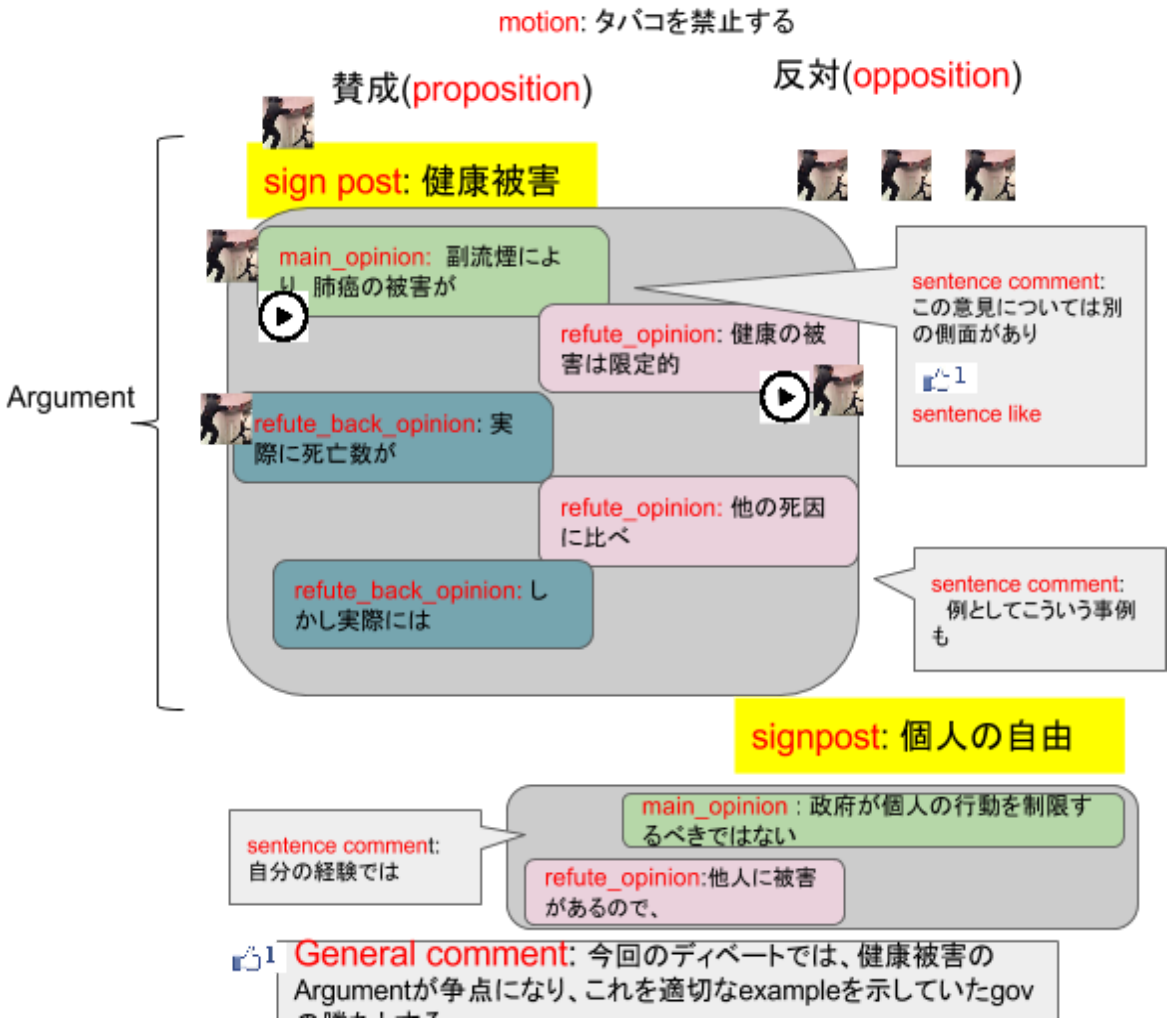
なのでArgumentにたいして、細かく、一行一行、精査していく作業になる。

→よって、sentence commentが必要になる。

勝敗の決定

argumentで成り立たったものがどれかを決める。反論されなかったものや、反論に対する反論で反論意見をつぶしていたものは成り立ったものととらえる。
相手の意見に反論をしているだけではなく、自分のArgumentがしっかりないと、自分の加点ポイントにはならない。
そのうえで、どのArgumentがより重要だったか(impact)を判断して勝敗を決める。

ディベートをUIで表すとどうなるか？



Motion: ディベートでディスカッションするお題
Opposition: 反対のチーム
proposition: 賛成のチーム
Argument: 議論に対する大きい一つのまとまり。

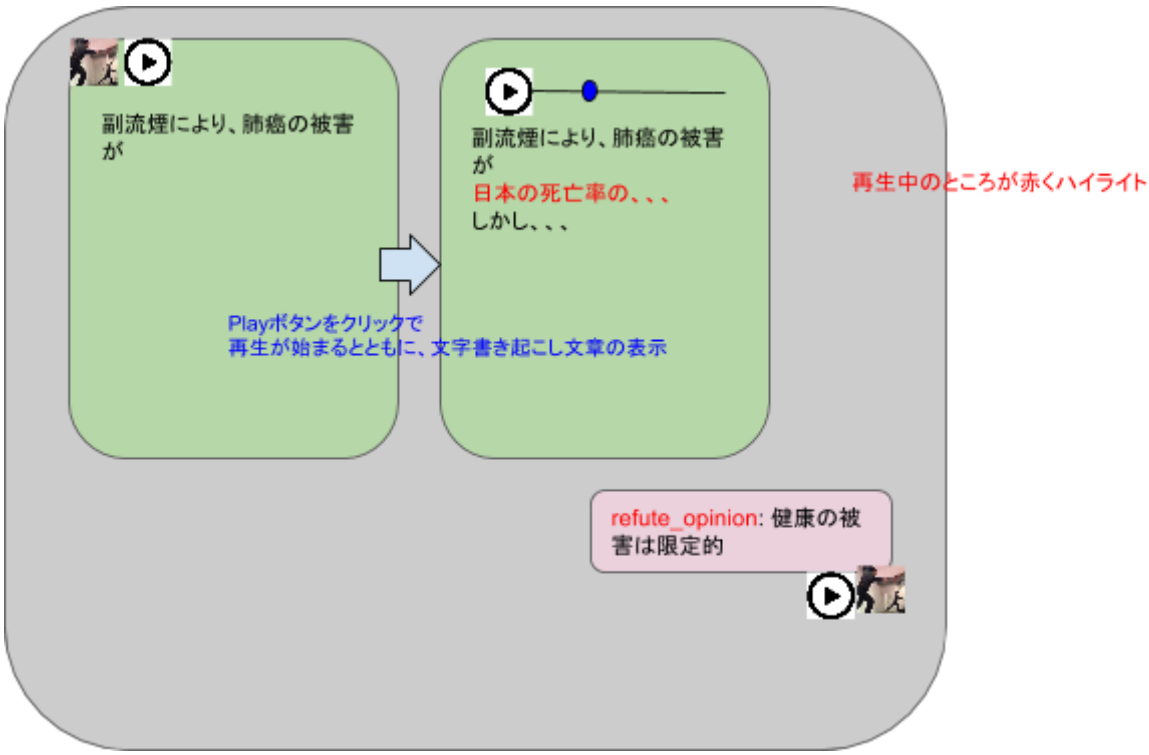
sign post: Argumentのタイトル
opinion: argumentのなかの一つ一つの意見。
　　opinion という用語はディベートにはないので
　　表示はさせない。main, refute, refute-backの三種類がある。
 > main: argumentのなかのはじめのopinion
 > refute: argumentに対する反対意見
 > refute-back: 反論に対する反論

general comment
　ディベートの全体に対するコメントで、勝敗の理由などもここでいう。
 general like
ディスカッションぜんたいに対するLike
sentence comment,
　ーセンテンスづつのコメント。
　これは、本来の用語ではないので、表示のときには、comment for argumentくらいが無難。ただ、実装上は、argumentに対するものではなく１センテンスづつなので区別
 sentence like
ーつ一つのセンテンスにたいするLike。これは、sentence commentにたいするLikeではなく、あくまでセンテンスにたいするLike

Audience Vote:
　勝敗を、一般の人が勝敗を決めるときの、投票のこと
adjucator vote
　正式な審判が決めた投票
Decision: 勝敗の結果

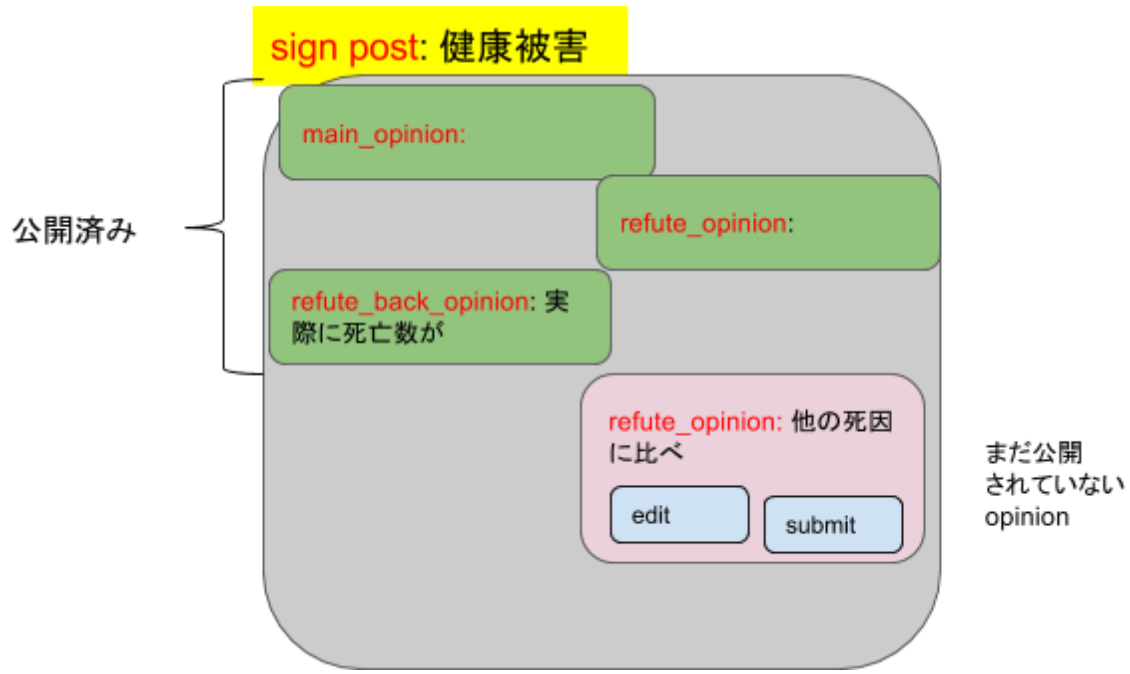
＜録音音声と書き込み文章について＞

それぞれのopinionの入力時に、文章でタイプをするのと、録音の両方を用いることができる。録音のときに自動文字お越しをしているので、それも表示することができる。



＜Writte Debateにおける特有のプロセス＞

今回のプロセスでは、チームメイトがopinionをチームメイトが承認するまで公開しないというプロセスをふむ。



submitボタンをおすことで、公開される。
submitボタンは、書き込んだ人でもだれでもsubmitできるようにしておく。
一度公開されたopinionについては、編集は不可能とする、
　→反論されたからopinionを編集したら、ずるい。

当然だが、相手方の公開されていないopinionは表示されない。

＜データベース構造＞

第二階層において並列の関係になる要素は共通のIDを持ち、同じIDを用いて並列関連のデータを取得可能にする。
event listのIDとそのなかに記載されているevent typeを取得するとでwritten debateのデータをIDとtypeを用いて取得できる。
list化する必要があるデータは、一つの塊の中のデータをできるだけ少なくし、実際のコンテンツを表示するときに、大きいデータを持つtreeからデータを取得する。

-event-related
-- article : article listで検索するのに最低限の情報を取得するデータを格納
-- event : event listで検索するのに最低限の情報を取得するデータを格納
-- written_debate : written debateに設定するデータ
-- livevideo_debaate : live video debateに設定するデータ

- user
-- user_basic: 名前と写真などの一般公開データ
-- event_list: 参加しているイベント
-- notify: 各ユーザにnotificationをするデータを格納, ユーザがこのデータをsubscribeすることでpush通知される。
-- chat_notify: chatのnotification

- url_related
--ogp ogp関連の情報を格納

- follow
--follower
--following

＜公開済みコンテンツと非公開コンテンツの管理＞

公開非公開のセキュリティはあまり強いセキュリティではないので、表示非表示をわけるのみで区分けする。

arg_statusで、componentの配置や表示非表示を管理し、実際のデータをopinionから取得して表示するという形をとる。

-arg_status : それぞれのコンポーネントを誰に表示してよいかの条件を設定
--arg_id : argumentの大きい塊のリスト
---main : argumentのうち、はじめのopinionの設定
----opinion_id: 関連するopinionデータのID
----status: approved, checkingの二つのステータスをもち、
checkingのときには同じチームの人しかopinionをみることができない。
特に、main opinionがcheckingのときには、argumentの全体のcomponentが見えなくなる。
----team_name: prop, oppのどちらかでcheckingのときに見えなくするための、チーム名
---subsequent refute, refute-backなどのmainにつづく複数のopinionのリスト
----opinion_id: 同上
----status: 一つ一つのopinionの表示非表示を管理
----team_name: 同上

-opinion : それぞれのopinionのデータを
--opinion_id : オピニオンデータのリスト

---content : 各セクションのデータを配列でもつ
---audio_url : オーディオファイルのURL
---transcript : 文字お越ししたデータを各センテンスごとに配列で格納
--type : main, refute, refute-back それぞれのタイプにより、表示の色や位置が異なる。
--writer : 書き込み者のユーザID

-comment : コメント
-- general comment ディベートの最後の下に記載するコメント
--- comment_id
---- context
---- user_id
--- section comment 各オピニオンの中の各セクションに記載するコメント
----opinion_id
-----section_num:　セクションの配列の添字
------comment_id
------user_id
------context: コメントの内容
-- transcription section comment　文字お越し文章に対するコメント。構造は同上

　sectionにはデフォルトではコメント数のみを記載するが、コメントのデータをsectionまで伝播させ、コメントの数を計算して表示する。

Like
likeの構造はcommentとほぼ同様
dislike
dislikeの構造はcommentとほぼ同様

＜データ構造とセキュリティ＞

firebseのへのデータ格納アクションにおける、設定データとセキュリティルールを明確化する。
ここで記載のセキュリティ設定は、動作確認をfirebaseの別アカウントsecurity-confirmation、webdemo.dac.co.jp/firebase_security.htmlで確認すること。

＜記載事項＞
利用用途：データの利用用途
パス：firebaseにデータ追加コマンドである、pushやsetなどを実行するパス
データ：作成するデータのオブジェクト
セキュリティ方針：誰にデータ書き込みや読み込みをきょかするか？
セキュリティ設定：セキュリティ方針を満たす、firebaseに設定するセキュリティ

■event_data

- **パス**：
  - event_related/event/$event_id に配列的にpush される。
- **データ**：
  - 作成時
    - {date,type,title,created_by}
  - 参加時
    - {participants:{uid:true}}
  - 変更
    - date, type, title
- **セキュリティ方針および設定**

＜イベントのデータ形式＞
イベントには、title, created_by, dateが存在すること
".validate": "newData.hasChildren(['title','created_by','date'])"
＜イベントの新規作成＞
誰でも作成できること：古いデータがない場合にはＯＫ
(!data.exists())
<イベントの削除>
作成者のみが、まだ参加者がいないときに、削除ができること：
古いデータがあり、新しいデータがない場合には、実行者は作成者と同じで、まだ参加者がいないこと。
((data.exists() && !newData.exists()) &&
(data.child('created_by').val()==auth.uid && !data.child('participants').exists()))

<作成データには、created_by, date, titelが存在すること>
".validate": "newData.hasChildren([‘date’, ‘title', ‘created_by'])"
<イベントの削除は作成者のみでまだ参加者がいないこと＞
古いデータがあり新しいデータがない場合には、古いデータのcreatecbyとauth.uidがマッチしていてparticipantがないこと
((data.exists() && !newData.exists()) && (data.child('created_by').val()==auth.uid && !data.child('participants').exists()))

■イベントの日付変更

- **パス**：event_related/event/$event_id/date
- データ：Date()型を数字にgetDateで変換した数値
- **セキュリティ方針および設定**

<作成は条件なしで可能なこと。>

```
!data.exists()
```

<更新は、作成者のみで、まだ参加者がいないときに変更可能>

```
(
  data.exists() &&
  (data.parent().child('participants').exists() == false) &&
  (data.parent().child('created_by').val() === auth.uid)
)
```

■イベントのタイトル

日付とほぼ同様だが、参加者が決まっていても、変更可能

■opinion

- **利用用途:**
  - ユーザの一つ一つの書き込み単位で、立論、反論などの種類に関わらず、一回のディベータの書き込みにあたり、複数のopinionでArgumentが構成される。
  - Argumentの下位のデータ要素になるが、arg_idがpushで自動で生成されるのにたいし、opinion_idはopinion作成画面が生成するidとなる。
  - art_idとopinion_idと一つのコンテンツを複数のidで分けた理由は、トップのcomponentから下位のcomponentにデータが伝播するときに、各かたまりに階層構造でデータが渡るようにするため
-
- **パス**：event_related/event-id/opinion/art_id/opinion_id
- **データ**：
  - content_arr: 書き込みの文章のセクションごとの配列
  - audio_url
  - transcription_arr: 文字お越しした文章オブジェクトの配列
  - type: main, refute, refute_backの三種類
  - writer: 書き込みをする人
- **セキュリティ方針および設定**

<イベントへの参加者が作成可能なこと>

まだ、データが存在せず、ログインしていて、イベントのparticipantsにuidが設定されていた場合に作成ができる。

<書き込んだ人が更新可能なこと。>

既にデータが存在していて、そのデータのwriterとauth.uidが同じであること。

＜書き込みデータのユーザとwriterは常に一致していること＞

■セクションコメント

- **利用用途:**

    - コンテンツや文字書き起こしの１センテンス１センテンスに書き込みができること。
    - 新規の追加のみで、編集などは認めない。

- **パス**：
  event_related/event-id/comment/section_comment/arg_id/opinion_id/section_numにpush
- **データ**：
    - comment_content: コメントの内容
    - user_id: 書き込み者のユーザID

＜written debate layout の構造とデータ伝播＞

構造設計方針

Componentを形勢する構造として次のようなcomponent interactionをおこなう。
- 上位から下位に大してInputのデータを受け渡す。
-下位から上位に大して、outputのon関数を実行する
上位のSmart Componentが受け取るデータは、FirebaseとSyncするのでリアルタイムで変更されるが、下位の必要ではないcomponentには、データ変更が伝わらないように、可変パラメータが無駄に下位に伝わらないようにきをつける。
可変ではない、IDなどについては下位に全てつたえ、outputで上位にわたるパラメータを全て下位のcomponentが把握できているようにする必要がある。

ディベータとその他の人での表示差分

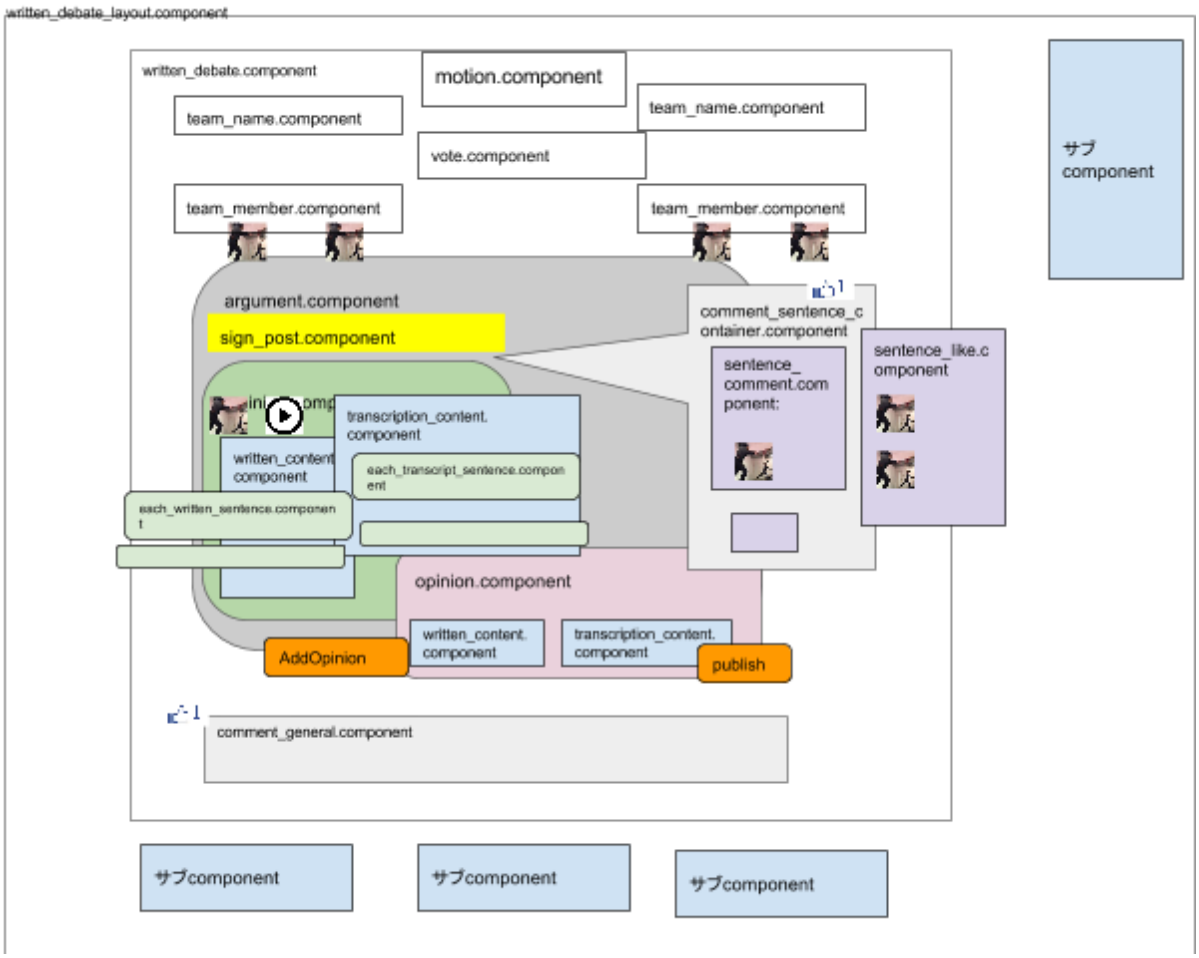
urlは同じだが見え方がディベータとその他の人とタイミングにより異なる。
- Voteをするボタン
    - ディベーターに見えない。
    - イベントが終わるまで見えない。　→このタイミングは要検討
- opinion_checking.component
    - ディベーターで、自分のチームのopinionしか見えない。
- opinion 追加ボタン
    - ディベーターにしか見えない。

通知（Notification）

ユーザのコメントなどの行動は、データをFirebaseに書き込むのと同時に、適時ユーザにNotificationが送られる。Notificationは、各ユーザの固有のデータ領域に書き込みを行うため、AWS Lambdaを通じて、Master鍵を通じてデータの書き込みがおこなわれる。
通知するユーザを取得するために、これらのアクションは、下位のDump componentではなく、Smart Componentで処理が行われる。
- sentence comment追加時の通知
    - 同じargumentにいままでcommentした人全て
    - 両チームのディベーター
- general comment
    - general commentに今までcommentした人全て

    - 両チームのディベーター
- publish opinion
    - 両チームのディベーター：これは、最優先の通知として、通常の通知とは違う、Notificationを行う。chromeのnotificationができた場合にはそれも利用
- sentence Like
    - Likeされたディベーター
- general like
    - ディベーター

UIとComponent



各componentにおいて記載する事項

役割：コンポーネントの主な役割
取得データ：上位のcomponentからのinputやrouterからのデータ取得など、外部からcomponentができた時点で取得できるデータ
生成データ：取得データやサーバとの通信など、あとからの計算などにより生成することができるデータ
処理：なにかユーザの動作にかかわるような事項
クラス関数：component内のイベントの処理など、何らかの関数
ユーザ行動：そのコンポーネントでユーザが行うアクションなど。
利用service: サービスとして利用するものの
child componentとのinteraction：下位のcomponentと受け渡すデータ

＜written_debate_layout.component：スマート＞

■役割：routerに設定されるcomponentであり、サイドパネルの構成などとメインコンテンツのwritten_debate_layoutの配置を行う。スマホのに非表示にするコンテンツなどはここで切ってしまう。

＜written_debate.component：スマート＞

■役割

- written debateの機能を全て包含するsmart component。
- routerからevent_idを受け取り、firebaseからeventに関するデータを受け取る。
- 下位のdump componentからのoutputの関数もここに実装する。

■利用service

- userauth: ユーザのログイン状態とユーザIDを取得
- angularfire: サーバとのデータ送受信に利用

■取得データ

- **event_id:** urlのrouterからrouter paramとしてデータ取得
- **user_id:** userauthサービスから取得

■生成データ

- **event_data:** firebaseからpath:(written_debate/event_id)で取得する、本ページで利用する全てのデータが含まれるJSONデータ
- **own_team**: イベント参加者のユーザ名一覧と、自分のユーザIDを比べ、自分のチームを特定して変数として利用する。
    - prop, opp, audienceの三種類があてはめられる。
    - checking opinionのどちら側のチームのopinionを表示するかに用いられる

■クラス関数

- onCommentSentence(arg_id, opinion_id, section_num, comment_content)
    - path(/written_debate/event_id/comment/section_comment/arg_id/opinion_id/section_num)にたいして次のデータをpushする。
    - {comment_content,user_id}
    - firebase security: write権限をpathに対してauth userにだけ与えるとともに、データのvalidationで、usser_idがマッチしていること。
- onCommentGeneral()
    - path(/written_debate/event_id/comment/general_comment)に対して次のデータをpushする。
    - {comment_content, user_id}
- onLikeSentence(arg_id, opinion_id, section_num)
    - path(/written_debate/event_id/Like_user/section_like/arg_id/opinion_id/section_num)に対して次のデータをpush
    - {user_id}
- onLikeGeneral(user_id)
    - path(/written_debate/event_id/Like_user/general_like)に対して次のデータをpush
    - {user_id:true}
- onPublishOpinion(arg_id,opinion_type,subsequent_id ,opinion_id)
    - opinion_typeがmainの場合

        - path(/written_debate/event_id/arg_status/main/status)をapprovedに変更
    - opinion_typeがsubsequentの場合
        - path(/written_debate/event_id/arg_status/subsequent/subsequent_id/status)をapprovedに変更

■child componentとのinteraction

- motion.component
    - input: モーション
- team_name.component
    - input: チームネーム
- team_member.component
    - input: useridの配列
- vote.component
    - input: 投票結果
- argument.component
    - ループ：arg_status.**arg_id**のリストで呼び出す。
    - 表示の有無：
        - arg_status.arg_id.main.status == approvedの場合
            - 常に表示
        - arg_status.arg_id.main.status == checkingの場合
            - arg_status.arg_id.main.team_name == own_teamのときに表示
            - mainのopinionが表示されていないときには、Argumentじたいができていないはずなので、本来opinionのあるなしの判断をする変数を、argument全体の表示の有無に利用している。
    - input:
        - **arg_id**:ループの情報を渡す。
        - partial_arg_status: arg_status[arg_id]を渡す。
        - partial_opinion: opinion[arg_id]を渡す。
        - partial_section_comment: comment.section_comment[arg_id]
        - partial_like_users: Like_users[arg_id]を渡す。
        - arg_sign_post: sign_post[arg_id]を渡す。
        - **own_team**を渡す。
        -
    - output
        - コメントしたときのアクションは、最上位のスマートコンポーネントまで伝える。最上位のcomponentだけが、ユーザの全情報を持つからである。
        - onCommentSentence
            - arg_id, opinion_id, section_num, comment_content
        - onCommentGeneral
            - comment_content
        - onLikeSentence
            - arg_id, opinion_id, section_num,
        - onLikeGeneral
            - non

- comment_general.component
    - input: comment.general_component

■処理

- **ログインの強制**
    - ディベート期間中で、まだ書き込みが可能な状態だと、イベントの期間から判断される場合には、強制的にログインダイアログを出す。
    - ログイン実施後に、全体が書き換わるように、userauthからのデータは、asyncで利用し、UIがアップデートされるようにしておく必要がある。
- **ユーザIDのuser_data_serviceへの引き渡し**
    - ユーザの名前や写真のデータは、event_dataで取得せず、IDのみを取得しているので、ユーザデータはfirebaseに取得させにいく必要がある。上位のsmartcomponentで、ユーザのIDをuser_data_serviceに引き渡しておく。
    - ディベーターのuser_id: dbater.proposition, debater.oppに格納さえている。
    - ディベーターのuser_idは書き込み者でもある、opinion.arg_id.opinion_id.writerにもあり、重複している可能性もあるがこれも、user_data_serviceに渡す。
    - セクションコメンター：comment.arg_id.opinion_id.section_num.comment_id.user.idに格納
    - transcription_commentor:comment.transcription_section_comment.arg_id.opinion_id.section_num.comment_id.user_idに格納
    - likeuserは、ここではデータ取得は行わない。likeユーザを見るを表示されたときにはじめてデータ取得がはじまる。

- **データを取得してから決まる状態**
    -
    - **type**: イベントの期間でありかつ、team_nameがprop, oppであった場合にはeditに、それ以外の場合には、articleになる。

＜argument.component＞

■役割

- 複数のopinionを包含するArgument
- ディベーターは、新規のopinionを追加することができる。
    - opinionを記載するのは別URLなので、パラメータは渡す必要あり

■取得データ

全てが上位のComponentから引き継がれる**input 情報**

- arg_id:input情報からそのまま利用
- partial_arg_status: input情報からそのまま利用
- arg_sign_post: input情報からそのまま利用
- partial_opinion: input情報からそのまま利用
- partial_section_comment: input情報からそのまま利用
- partial_like_users: input情報からそのまま利用

- **own_team**: input情報からそのまま利用

■生成データ

- **arg_team:** inputデータのpartial_arg_statusより、partial_arg_status.main.team_nameより、argument全体でのチームのサイドを特定し、クラス変数とする。

■表示方法

- arg_teamがpropositionであったときには左寄り
    - sentence commentが右側になる。
- arg_teamがoppositionであったときには右寄り
    - sentence commentが左側になる。

■child component

- sign_post.component
    - input: arg_sign_postを渡す
- opinion.component
    - ループ
        - partial_arg_status.main
        - partial_arg_status.subsequentの下の配列をループで呼び出す。
        - ループからopinion_idを取得する。
    - 表示非表示の制御
        - partial_arg_status.subsequent.statusと partial_arg_status.main.statusが、approvedの場合には表示
        - 上記がcheckingの場合にはpartial_arg_status.subsequent.team_name ==**own_team** の場合には、表示
    - input
        - **opinion_id**: **partial_arg_statusのループのitemからopinion_idの添字で取得したID**
        - arg_id: input情報からそのまま受け渡す
        - **opinion_status**:**partial_arg_statusのitemからstatusで取得した文字列。approvedまたはcheckingが渡される。**
        - **opinion: partial_opinionから、opinion_idを用いて取得したデータ**
        - **section_commen**t_arr:**partial_section_commentからopinion_idを用いて取得したデータ**
        - **transcription_comment_arr**：**partial_transcription_commentからopinion_idを用いて取得したデータ**
        - l**ike_users: partial_like_usersから、opinion_idを用いて取得したデータ**
        - **opinion_type:**
            - **main:** partial_arg_status.mainから呼び出した場合
            - **subsequen**t: partial_arg_status.subsequentから呼び出した場合
        - **subsequent_id**:
            - partial_arg_status.subsequentからループで呼び出した場合の配列の添字を渡す。
    - output

- written.debate.componentと同じ。そのまま上位に受け渡す。

■クラス関数

- addOpinion()
  - 追加ボタンをクリックすることで呼ばれる。
  - 新規のopinionを追加するために別画面にいくためのNavigationのためのぼたん

＜opinion.component：Dump＞

■役割

- それぞれのopinionを表示するためのcomponent

■取得データ

全てが上位のComponentから引き継がれる**input 情報**

- arg_id:input情報からそのまま利用
- opinion_id: input情報からそのまま利用
- **opinion**: input情報からそのまま利用
- opinion_status: input情報からそのまま利用
- section_comment_arr: input情報からそのまま利用
- l**ike_users**: input情報からそのまま利用
- **transcription_comment_arr**: input情報からそのまま利用
- **opinion_type:** input情報からそのまま利用
- **subsequent_id:** input情報からそのまま利用

■生成データ

- writer_user_id: opinion.writerによりopinionを記載したユーザのIDを取得。

■Viewの表示

- 全体の色
  - opinion_statusがapprovedの場合は、通常の記事表示色
  - opinion_statusがcheckingの場合は、警告色
- submitボタンの表示
  - opinion_statusがcheckingの場合にのみ表示
- Audio playボタンの表示
  - opinion.audio_urlがあったときにのみ表示
- コンテンツの表示
  - デフォルトでは、書き込み側の表示
    - 書き込みがなかった場合に、transcriptionの表示

■関数

- publish_opinion
  - submit ボタンがクリックされたときの動作
  - これも上位のcomponentlに通知し、上位のcomponentからfirebaseに通知し、arg_statusを編集
  - publish_opinion.emit(arg_id, ,,,, opinion_type)
- play_audio
  - Audioボタンをクリックされたときの動作。音声の再生がはじまるとともに、コンテンツ表示がtranscriptionに切り替わり、transcription componentにcurrent time のinput にわたしはじめる。

- onPlay_startTime(start_time)
    - transcription.compnentから開始時間をうけとり、Audioを指定時間から再生する
    -

■child component

- written.component
    - input:
        - content_arrをopinion.content_arrで渡す。
        - section_comment_arrをそのまま受け渡す。

- transcription.component
    - input
        - current timeをプレイヤのの再生時間より渡す。
        - transcription_arrをopinion.transcript_arrで渡す。
        - transcription_comment_arrをそのまま受け渡す。
    - output
        - onPlay_StartTime(time):

＜written.component：Dump＞

■データ取得

- content_arr
- section_comment_arr

■データ表示

- XXXXをトリガーに、comment_sentence_container.componentを表示する。
-

■child component

- comment_sentence_container
    - input
    - output:
        - onAddComment

<sentence_written.component：Dump＞

＜transcription.component：Dump＞

■データ取得

- current time
    - audio playerの再生時間
- transcription_arr

<sentence_transcription.component：Dump＞

■取得データ

- start_time
- finish_time

- transcript_sentence
- audio_current_time

■処理

- 入力で受け取る動的な変数であるaudio_current_timeが、
- start_time < audio_current_time < finish_time という関係であったときだけ、センテンスを赤い色で表示

■クラス関数

- onSentencePlay()
    - センテンスの横かどこかに、再生ボタンを付与し、そこがクリックされると、outputのonPlay_StartTime(start_time)を実行して上位に伝える。
- showComment()
    - センテンスの横かどこかに、コメント表示ボタンを付与し、クリックすると、comment_sentence_containerを表示

■child component

- comment_sentence_container
    - input
        -

＜comment_sentence_container.component＞

■利用方法

transcription_container, writtten_content.containerの両方とも、comment sentence containerにアクセスする方法は同じ。

■データ利用

- section_comment_arr: transcription_containerからも、written_content.contaienerの両方からこの名前でデータをうけとる。

■child component

- comment_sentence_eachをループで呼び出す。

■関数

- show_like_users
    -
- add_comment
    - コメントをユーザが追加するときに呼び出す。伝播して上位のスマートコンポーネントで処理。

＜comment_sentence_each.component＞

ユーザの写真とコメントを表示

＜comment_sentence_likeuser.component＞

likeしたユーザの一覧をリストとして表示

<データ取得:user_data_serviceの動作>

behavior subjectを利用した、firebase dataの取得