

# Firebase

---

# Contents

---

- 関数
  - 関数の定義
  - 引数と戻り値
  - 関数の練習(乱数の生成)
- 自作チャットの作成
  - Firebaseの準備
  - チャット作成の準備
  - チャット処理と画面の作成
- 課題発表→P2Pタイム

# rules...

---

- 授業中は常にエディタを起動！
- 考えたことや感じたことはslackのガヤチャンネルでガンガン発信！
- 質問はslackへ！他の人の質問にも目を通そう！(同じ質問があるかも)
- 演習時、できた人はスクショなどslackに貼ってアウトプット！
- まずは打ち間違いを疑おう！
  - {}"';など
- 書いたら保存しよう！(よく忘れる！)
  - command + s
  - ctrl + s

# 本日のゴール

---

- オンラインでデータを扱う！
- リアルタイムでデータ共有する！
- Firebaseの癖を把握する！

# 関数(function)

# 関数とは？？

```
// 関数 (function)
// - 関数とは記述した処理をまとめて名前をつけて使い回せるようにしたもの。
// - 一度処理を定義てしまえば、呼び出すだけで実行可能！

// 例
function test(){           // 関数の定義の仕方は決まっている
  console.log('関数は便利！');
}                           // 「{}」内に実行したい処理を記述
test();                     // 関数の実行 (console.log();が実行される)
```

関数は呼び出さないと実行されない！  
定義するだけではNG！

# 引数と戻り値

---

- 引数(関数に入力する値)
  - 定義した関数に対して、処理に必要な値を入力する。
  - 引数の数は一つでも複数でもOK！
- 戻り値(関数から出力されてくる値)
  - 関数の中で計算などを実行した後、結果を返す処理。
  - 関数内の変数、配列、オブジェクトなどで返せる。

# 引数と戻り値

```
// 関数の定義
function add(a, b){          // aとbが引数
    const total = a + b;
    return total;             // totalが戻り値
}
```

```
// 関数の実行
const sum = add(10, 20);
console.log(sum);           // 30が表示される
```

10と20を入力すると30が返ってくる

# プログラミングの関数 = 数学の関数

---

例:  $f(x) = x^2 + 2x + 1$  の関数を定義すると. . .

$$f(2) = 9,$$

$$f(5) = 36,$$

$$f(10) = 121$$



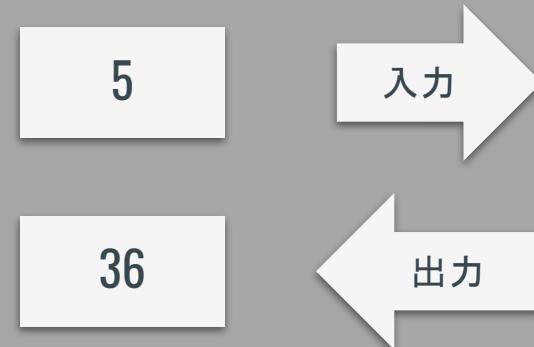
# プログラミングの関数 = 数学の関数

例: JavaScriptで書くと. . .

```
calculate(2);    // 9
```

```
calculate(5);    // 36
```

```
calculate(10);   // 121
```



```
function calculate(x) {  
  const result = x * x + 2 * x + 1;  
  return result;  
}
```

引数と戻り値がない場合もある(ややこしいポイント)

# これまでに登場していた関数

---

```
// 亂数関連も関数（最初から用意されている関数）
```

```
Math.random();           // 0.534714863872  
// 引数：なし  
// 戻り値：0.534714863872
```

```
Math.floor(3.1415926535); // 3  
// 引数：3.1415926535  
// 戻り値：3
```

# 【おまけ】関数の書き方

```
// 関数の記述方法（関数内の処理は同一）  
function add1(a, b){  
    return a + b;  
}  
  
const add2 = function(a, b){  
    return a + b;  
}  
  
const add3 = (a, b) => {  
    return a + b;  
}
```

全部(大体)同じ！  
add(10, 20);で実行！！

# 閾数の利用

# 関数の使いどころ

---

- 関数の利点
  - イベントごとに毎回同じ処理を書くのは面倒！
  - 関数を定義しておけば、ボタン押したら実行するだけ！
- 例
  - 押したボタンに応じて、異なる範囲の乱数を発生させたい！

# 関数の使いどころ

```
// 関数の定義
function generateRandomNumber(min, max){
  const rand = Math.floor(Math.random() * (max - min + 1) + min);
  return rand;
}

// 実行するときはこんな感じ
const result = generateRandomNumber(1, 9);    // 1から9までの乱数
console.log(result);
```

# 関数の使いどころ

```
// ボタンをクリックしたイベントで関数を実行
$('#btn01').on('click', function () {
  const result = generateRandomNumber(1, 10);
  $('#echo').text(result);
});
```

ボタンごとに範囲を設定して実行

```
// ※btn02, btn03も同様
// 【参考】janken.htmlに関数を使用したじゃんけんの例もあります！
```

# 関数の練習

---

- 関数の応用
  - 最小値と最大値を入力してランダムな数を返す関数を定義しよう！
  - 各ボタンのクリック時に関数を実行し、結果をechoに出力しよう！

# 自作チャットの実装



# Webブラウザでチャット

The diagram illustrates a real-time chat application running in two separate web browser windows. Both windows show the same interface with a text input field ('user: 1'), a send button, and a message history area.

**Left Window (User 1):**

- Text input field: user: 1
- Send button: send
- Message history:
  - 1 2019-06-07T18:24:21 11111111
  - 2 2019-06-07T18:24:10 2222
  - 1 2019-06-07T18:23:48 1111

**Right Window (User 2):**

- Text input field: user: 1
- Send button: send
- Message history:
  - 1 2019-06-07T18:24:21 11111111
  - 2 2019-06-07T18:23:48 1111

**Annotations:**

1. 片方で送信すると... (When you send from one side...)
2. 両方で表示される！ (Both sides will display it!)

A large double-headed arrow points between the two windows, indicating that messages sent by either user are immediately reflected in the other user's window.

# Firebase(cloud firestore)とは？？

---

Firebaseは、クライアントからアクセス可能なデータベースとしてFirebase Realtime Database( 以下 Realtime Database)とCloud Firestoreの2つを用意しています。

Realtime Databaseは、リアルタイムでクライアント全体の状態を同期させる必要があるモバイルアプリ向けの効率的で低レイテンシなものです。

Realtime Databaseはクラウド上でホスティングされるNoSQLのデータベースです。データはすべてのクライアントにわたってリアルタイムに同期され、アプリがオフラインになっても利用可能です。クロスプラットフォームアプリを構築した場合でも、すべてのクライアントが1つのRealtime Databaseを共有して、最新のデータへの更新を自動的に行います。またクライアントからも直接アクセスが可能なため自前のサーバなしで使えるデータベースとしても活用できます。

Cloud Firestoreは、直感的な新しいデータモデルで、Realtime Databaseの性能をさらに向上しており、Realtime Databaseよりも豊かで高速なクエリとスケールを備えています。Cloud Firestoreは2017年のGoogle I/Oで発表されたプロダクトであり、2018年5月現在はベータ版リリースです。

引用：WEB+DB PRESS vol.105 第4章（※2019年2月より正式版として運用されています。）

# Firebase(cloud firestore)とは？？

---

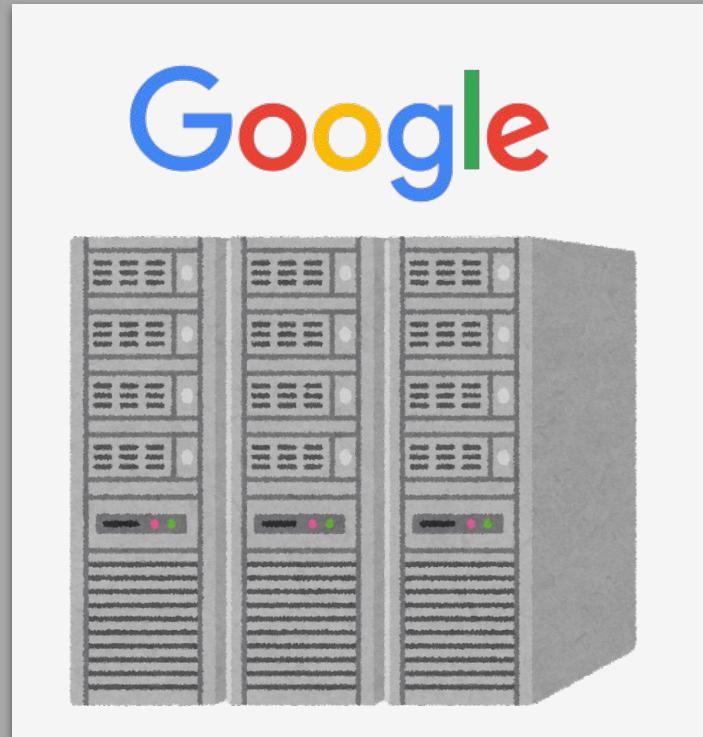
- サーバ上にデータを保存できる！
- 保存したデータをリアルタイムに同期できる！
- 異なるデバイスでもデータを共有可能！
  - PCとスマホでリアルタイムにデータを同期できる.
- JavaScriptのみで実装可能！
  - Swift, Go, Pythonなど他の言語でも使用可！

# サーバにデータを保存する

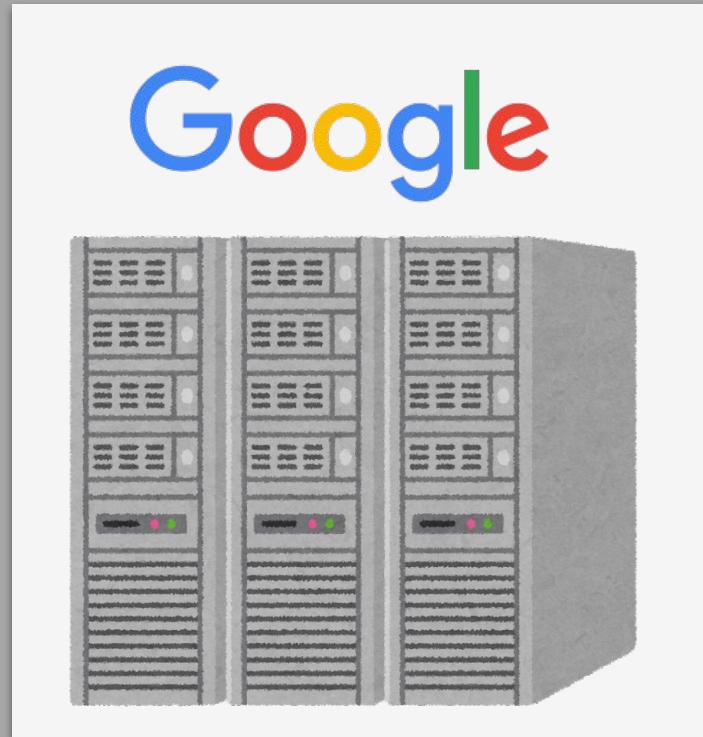
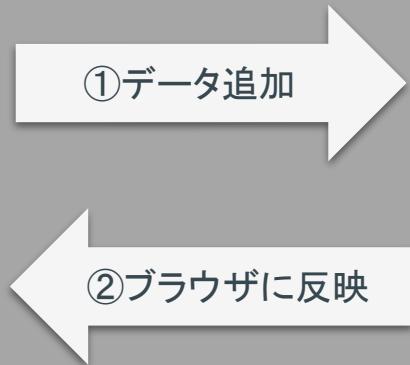
---



データ送信



# サーバにデータを保存する



# サーバにデータを保存する



①データ追加



②全てに反映

②全てに反映

Google



# Firebaseの準備が必要！

# 準備の流れ

---

- ①ログイン
- ②プロジェクトの作成
- ③権限の設定
- ④データベースの準備

<https://firebase.google.com/>にアクセス！

The screenshot shows the official Firebase website homepage. At the top, there is a navigation bar with links for 'Firebase' (with a logo), 'プロダクト', '使用例', '料金', 'ドキュメント', 'サポート', a search icon, a search bar labeled '検索', a link to 'コンソールへ移動', and a 'ログイン' button. Below the navigation bar, there is a banner for 'Firebase Crashlytics' with the subtext 'Prioritize and fix issues with powerful, realtime crash reporting.' Two large, semi-transparent callout boxes are overlaid on the page. The first box, located on the right side of the banner, contains the text '①ログイン' (Step 1: Log in). The second box, located below the banner, contains the text '②コンソール画面へ移動' (Step 2: Move to the console screen). The main content area features a large smartphone icon displaying various mobile application screens, such as a lock screen and a payment screen. Below the smartphone, the text 'Firebase helps mobile app teams succeed.' is displayed. At the bottom left, there are two blue buttons: 'スタートガイド' and '動画を見る'.

①ログイン

②コンソール画面へ移動

Firebase helps mobile app teams succeed.

スタートガイド 動画を見る

# 新規プロジェクトの作成

Firebase ドキュメントに移動  

## Firebase へようこそ

優れたアプリの開発、ユーザーとの交流、モバイル広告からの収益向上に役立つ Google のツールを利用できます。

[詳細](#) [ドキュメント](#) [サポート](#)

最近のプロジェクト

**FireTrip**  
firetrip-813c6

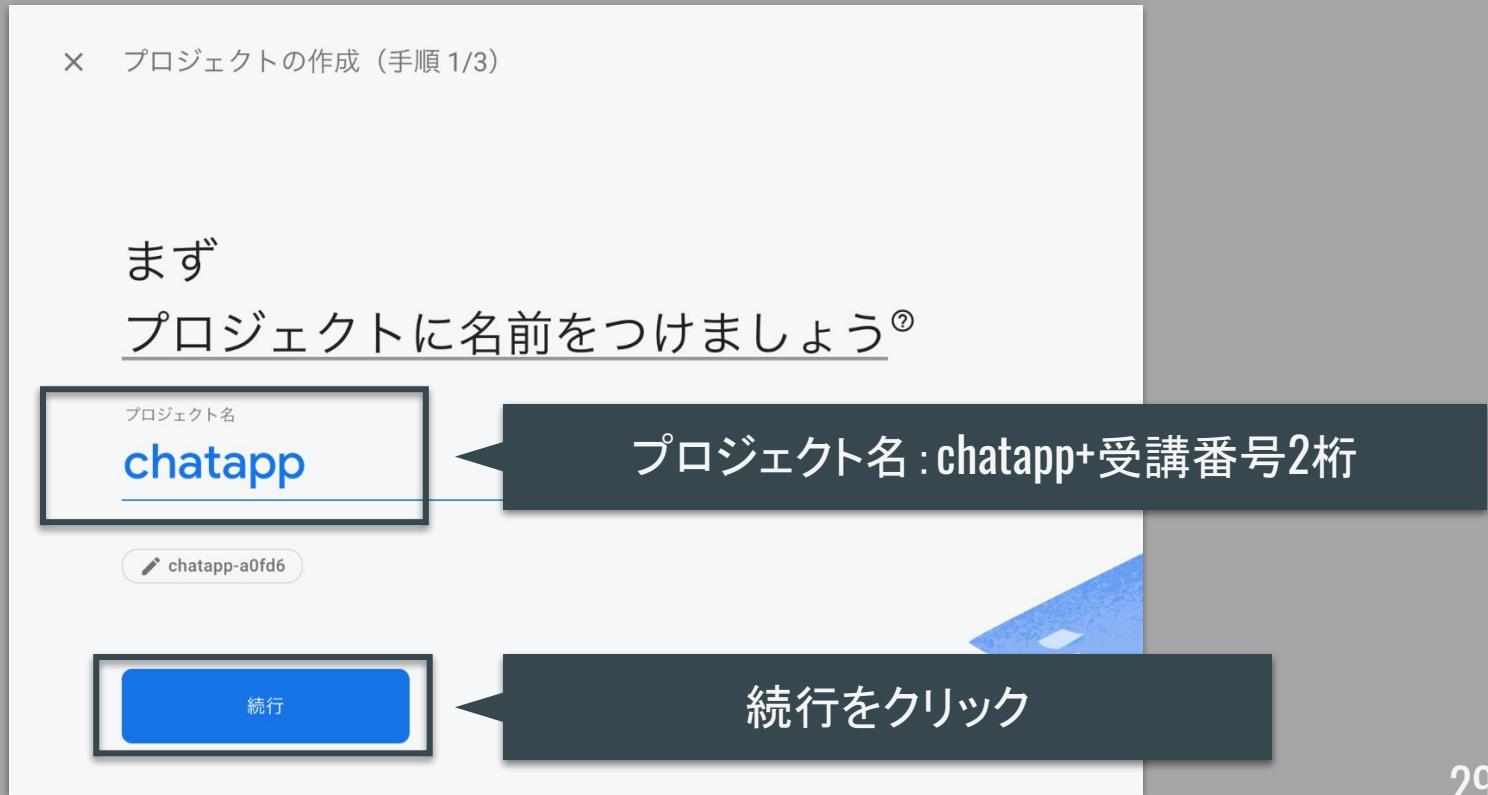
**web-db-test**  
web-db-test-3fd6d

**+**  
プロジェクトを追加

 デモ プロジェクトを見る



# 新規プロジェクトの作成



# 新規プロジェクトの作成

×

プロジェクトの作成 (手順 2/2)

Google アナリティクスは無料かつ無制限のアナリティクス ソリューションで、Firebase Crashlytics、Cloud Messaging、アプリ内メッセージング、Remote Config、A/B Testing、Predictions、Cloud Functions で、ターゲティングやレポートなどが可能になります。

Google アナリティクスにより、以下の機能が有効になります。

- ×
- A/B テスト ②
- ×
- Firebase プロダクト全体でのユーザー サービスとターゲティング ②
- ×
- ユーザー行動の予測 ②
- ×
- タラッシュに遭遇していないユーザー ②
- ×
- イベントベースの Cloud Functions トリガー ②
- ×
- 無料で無制限のレポート ②

このプロジェクトで Google アナリティクスを有効にする  
推奨

どちらでも

クリック !!

プロジェクトを作成

# 新規プロジェクトの作成



# Webアプリを追加

The screenshot shows the Firebase Project Overview interface for a project named 'chatapp'. The sidebar on the left contains links for Project Overview, Authentication, Database, Storage, Hosting, Functions, ML Kit, Crashlytics, Performance, Test Lab, Analytics, Dashboard, Events, Conversions, and Predictions, A/B Testing, Cloud M... . The main content area features a large blue background with white text and illustrations. The text reads: 'アプリに Firebase を追加して利用を開始しましょう' (Add Firebase to your app to start using it) and 'ほとんどの Firebase 機能と、iOS、Android アプリ用のAnalyticsが含まれた Core SDK をインストールしてください' (Install the Core SDK, which includes most Firebase features and the iOS and Android Analytics). Below this, there are icons for iOS, Android, and a square containing '</>' (representing web or server-side code), with the '</>' icon highlighted by a white arrow. At the bottom, it says '開始するにはアプリを追加してください' (Please add your app to get started). The top right of the screen shows standard navigation icons for document movement, notifications, and coffee.

# 適当に設定！！(プロジェクト名と一緒にわかりやすい)

## × ウェブアプリに Firebase を追加

### 1 アプリの登録

アプリのニックネーム ②

chatapp|

このアプリの **Firebase Hosting** も設定します。 [詳細](#) 

Hosting は後で設定することもできます。いつでも無料で始めることができます。

アプリを登録

### 2 Firebase SDK の追加

# 必要なコードが表示されるのでコピー

これらのスクリプトをコピーして <body> タグの下部に貼り付けます。この作業は Firebase サービスを使用する前に行ってください。

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/6.0.0.firebaseio.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
      https://firebase.google.com/docs/web/setup#config-web-app -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyB_qDNjlAe0naKGxr5vBzbcK6kL1tUVY30",
    authDomain: "testapp-279d8.firebaseio.com",
    databaseURL: "https://testapp-279d8.firebaseio.com",
    projectId: "testapp-279d8",
    storageBucket: "testapp-279d8.appspot.com",
    messagingSenderId: "760501105214",
    appId: "1:760501105214:web:af0034bfc6598206"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

ウェブ向け Firebase の詳細については、こちらをご覧ください: [使ってみる](#)、[ウェブ SDK API リファレンス](#)、[サンプル](#)

コンソールに進む

# エディタで貼り付け

```
22 <!-- The core Firebase JS SDK is always required and must be listed first -->
23 <script src="https://www.gstatic.com/firebasejs/6.0.0.firebaseio.js"></script>
24
25 <!-- TODO: Add SDKs for Firebase products that you want to use
26     https://firebase.google.com/docs/web/setup#config-web-app -->
27
28 <script>
29   // Your web app's Firebase configuration
30   var firebaseConfig = {
31     apiKey: "AIzaSyB_",
32     authDomain: "test",
33     databaseURL: "http",
34     projectId: "testap",
35     storageBucket: "te",
36     messagingSenderId:
37     appId: "1:76050110
38   };
39   // Initialize Firebase
40   firebase.initializeApp(firebaseConfig);
41 </script>
```



# 【超重要】ソースコードの修正

```
22 <!-- The core Firebase JS SDK is always required and must be listed first -->
23 <script src="https://www.gstatic.com/firebasejs/6.0.0.firebaseio.js"></script>
24
25 <!-- TODO: Add SDKs for Firebase products that you want to use
26     https://firebase.google.com/docs/web/setup#config-web -->
27
28 <script>
29   // Your web app's Firebase configuration
30   var firebaseConfig = {
31     apiKey: "AIzaSyB_",
32     authDomain: "test",
33     databaseURL: "http
34
35   },
36
37   // Initialize Firebase
38   firebase.initializeApp(firebaseConfig);
39
40 </script>
```

「-app」を削除！！！

コレやらないと一切動かない！！！

# ブラウザに戻ってコンソールに進む

これらのスクリプトをコピーして <body> タグの下部に貼り付けます。この作業は Firebase サービスを使用する前に行ってください。

```
<!-- The core Firebase JS SDK is always required and must be listed first -->
<script src="https://www.gstatic.com/firebasejs/6.0.0.firebaseio.js"></script>

<!-- TODO: Add SDKs for Firebase products that you want to use
      https://firebase.google.com/docs/web/setup#config-web-app -->

<script>
  // Your web app's Firebase configuration
  var firebaseConfig = {
    apiKey: "AIzaSyB_qDNjlAe0naKGxr5vBzbcK6kL1tUVY30",
    authDomain: "testapp-279d8.firebaseio.com",
    databaseURL: "https://testapp-279d8.firebaseio.com",
    projectId: "testapp-279d8",
    storageBucket: "testapp-279d8.appspot.com",
    messagingSenderId: "760501105214",
    appId: "1:760501105214:web:af0034bfc6598206"
  };
  // Initialize Firebase
  firebase.initializeApp(firebaseConfig);
</script>
```

ウェブ向け Firebase の詳細については、こちらをご覧ください: [使ってみる](#)、[ウェブ SDK API リファレンス](#)、[サンプル](#)

コンソールに進む

# データベースの準備



# データベース(Cloud Firestore)の準備

「テストモードで開始」→「有効にする」

The screenshot shows the 'Realtime Database のセキュリティ ルール' (Realtime Database Security Rules) dialog box. It contains two options:

- ロックモードで開始  
読み取りと書き込みをすべて拒否し、データベースを非公開で作成します
- テストモードで開始  
読み取りと書き込みをすべて許可し、設定をすばやく行います

A large orange callout bubble points to the second option, containing the text: 'データベース参照を所有しているユーザーでも、データベースの読み取りや書き込みが自由に行えます' (Even if the user owns the database reference, they can freely read or write to it). A large blue arrow points from this callout to the '有効にする' (Enable) button at the bottom right.

開発

- Authentication
- Database**
- Storage
- Hosting
- Functions
- ML Kit

品質

Crashlytics, Performance, Test Lab

アナリティクス

Dashboard, Events, Conversions, A...

拡大

Predictions, A/B Testing, Cloud M...

Realtime Database のセキュリティ ルール

データ構造の定義後に、データを保護するルールを作成する必要があります。

詳細 [\[リンク\]](#)

ロックモードで開始  
読み取りと書き込みをすべて拒否し、データベースを非公開で作成します

テストモードで開始  
読み取りと書き込みをすべて許可し、設定をすばやく行います

```
{  
  "rules": {  
    ".read": true,  
    ".write": true  
  }  
}
```

データベース参照を所有しているユーザーでも、データベースの読み取りや書き込みが自由に行えます

キャンセル 有効にする

# データベース(Cloud Firestore)の準備

## コレクションの開始

- 1 コレクションに ID を付与する
- 2 最初のドキュメントの追加

親パス

「コレクションを追加」→「chat + 受講番号2桁」を作成

コレクション ID ②

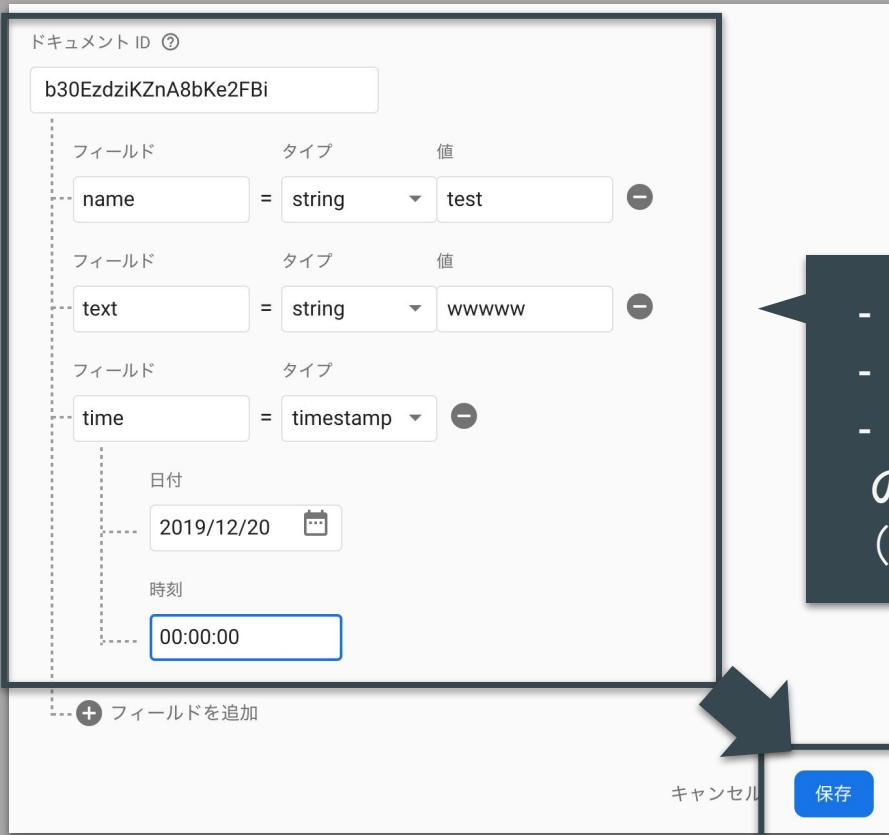
chat

キャンセル

次へ

# データベース(CloudFirestore)の準備

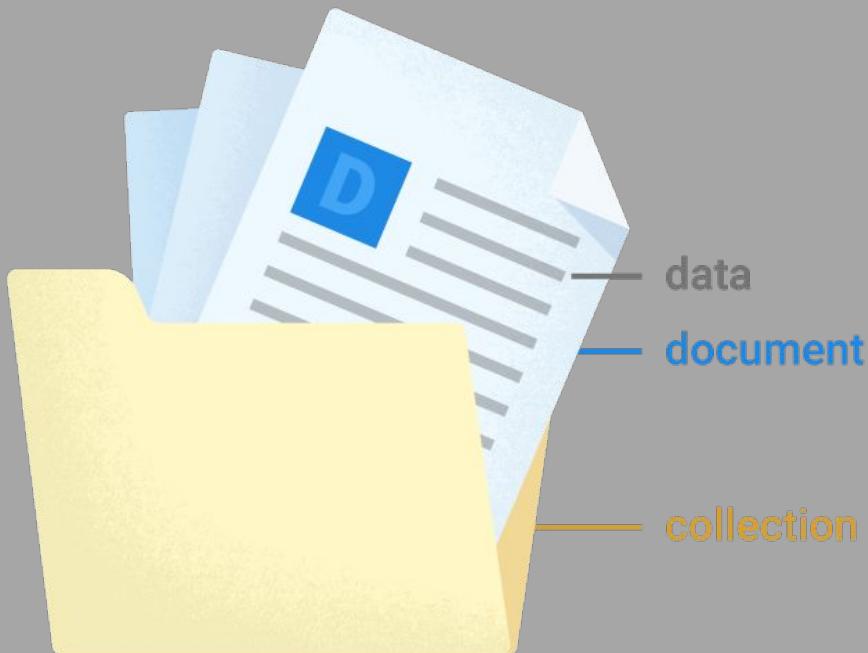
IDは「自動ID」



- name(string)
  - text(string)
  - time(timestamp)
- の3項目を設定！  
(値は適当でOK！)

# データベース(CloudFirestore)のイメージ

<https://firebase.google.com/docs/firestore/data-model?hl=ja>



# チャットの実装

# 必要な処理一覧

---

- チャット画面の作成
  - チャットを入力&表示する画面の作成
- データ送信の処理
  - 入力して送信ボタンを押下したらイベント発火.
  - 入力内容を取得する.
  - firebaseにデータを送信. 送信後に入力欄を空にする.
- データ受信処理
  - データ追加時に自動的にデータ取得する.
  - 受信したデータをブラウザ上に表示する.

# チャット画面の作成

# チャット画面の作成

```
// 入力フォームの作成
<ul>
  <li>
    <label for="name">name</label>
    <input type="text" id="name">
  </li>
  <li>
    <textarea name="" id="text" cols="30" rows="10"></textarea>
  </li>
  <li>
    <button id="send">send</button>
  </li>
</ul>
```

こんな感じ！

The diagram illustrates the rendered HTML structure. It shows a list item (`<li>`) containing a label (`<label>`) with the text "name", an input field (`<input type="text" id="name">`), a text area (`<textarea>`), and a button (`<button>`). A callout bubble points to the text area with the text "こんな感じ！". To the right, a separate box shows a simplified representation of the form fields: a text input labeled "name", a large empty text area, and a button labeled "send".

- name
- 
- send

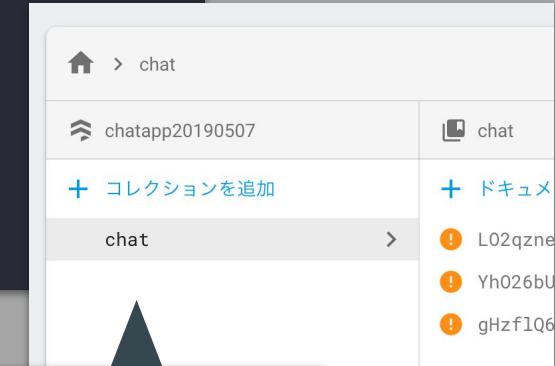
# データ送信処理の実装

# リアルタイム通信の準備

```
messagingSenderId: "445936384974",
appId: "1:445936384974:web:44b5b67ccfd0b39d"
};

firebase.initializeApp(firebaseConfig);

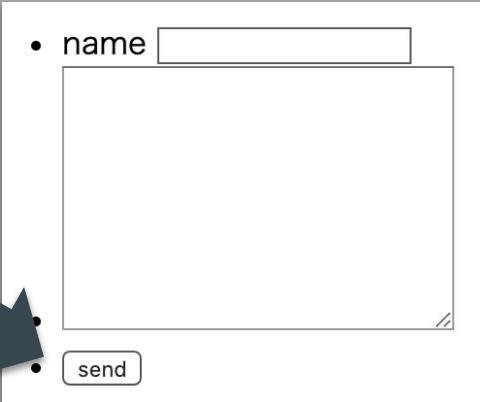
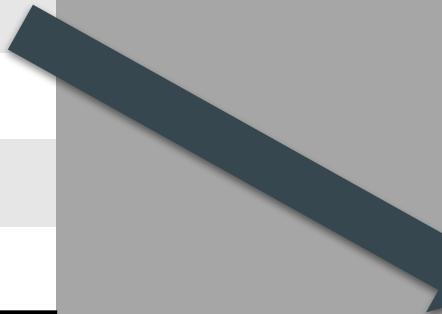
// cloudfirestoreの場所を定義する処理
var db = firebase.firestore().collection('chat')
```



「db」がここに対応

# 送信ボタンクリック時の処理

送信ボタン	#send
テキストボックス	#name
テキストエリア	#text
メッセージ表示領域	#output



The diagram shows a user interface with the following elements:

- A list item with the text "name" followed by an input field.
- A large empty rectangular area representing the "text" field.
- A list item with the text "send" followed by a button.

```
// 送信ボタンクリックでメッセージ送信
$('#send').on('click', function() {
    ...
});
```

# データ送信処理の流れ

---

## やること

- 送信ボタンをクリックしたら. . .
  - (.on('click', function(){})...の形)
- 名前と本文を取得.
  - (.val()とか)
- firebaseにデータ(名前, 時間, 本文)を送信.
  - (db.add()の処理を使う！)
- 本文入力用のtextareaを空にする.
  - (.val("")とか)

# データ送信処理のコード

```
// 送信処理の記述
db.add({
    name: $('#name').val(),          // dbが送信先 送信データはオブジェクトの形
    text: $('#text').val(),           // inputの入力値
    time: firebase.firestore.FieldValue.serverTimestamp(), // 登録日時
});
// 送信後にtextareaを空にする処理
$('#text').val('');
```

# データ送信処理の動作確認

The screenshot shows the MongoDB Compass interface with a collection named 'chat'. A new document has been inserted with the ID 'L02qznemG9iUVsf0qv3P'. A large dark overlay box contains the message: '送信が成功するとここに表示される！(ランダムな文字列のidで追加)'.

chatapp20190507	chat	L02qznemG9iUVsf0qv3P	⋮
+ コレクションを追加	+ ドキュメントを追加	+ コレクションを追加	⋮
chat	>	<ul style="list-style-type: none"><li>! L02qznemG9iUVsf0qv3P &gt;</li><li>! Yh026bUB1aWGUYToo899</li><li>! qHzf106...GGgF8AzJH</li></ul>	+ フィールドを追加

! インターネット上の誰でもこのドキュメントの読み取りと書き込みができます

送信が成功するとここに表示される！  
(ランダムな文字列のidで追加)

user: "1"

# データ送信の処理を実装しよう！

---

- ここまで作ろう！
  - 送信ボタンを押したら入力されたデータを送信！
  - firebaseのコンソール画面で送信されているかどうか確認！
- chatapp.htmlに記述しよう！

# データ受信処理の実装

# データ受信処理の流れ

---

## やること

- firebaseのデータに変更があったときに. . .
  - 保存されているデータを新しい順に並び替えて取得.
  - 保存されているデータについて、1件ずつidとデータを取得.
  - 必要なデータ(idと名前メッセージ時間)のみ入った配列を作成
  - ブラウザに出力するためにデータを適当なタグに入れた配列を作成.
  - 実際にブラウザに表示する.

※ データがわかりにくいので、都度console.log()で確認しよう！

# データ受信処理のコード

```
// 受信処理の記述
db.orderBy('time', 'desc').onSnapshot(function (querySnapshot) {
    // onSnapshotでcloud firestoreのデータ変更時に実行される！
    // querySnapshot.docsにcloud firestoreのデータが配列形式で入る
    const dataArray = [];    // 必要なデータだけが入った新しい配列を作成
    querySnapshot.docs.forEach(function (doc) {
        const data = {
            id: doc.id,
            data: doc.data(),
        }
        dataArray.push(data);
    });
    // 次ページへ続く... (dataArrayをconsoleで確認！)
})
```

# データ受信処理のコード

```
// ...前ページの続き 「`」で囲んでタグ文字列を表現. `${}` で変数を埋め込み
const tagArray = [] ; // `dataArray` は前回出てきたオブジェクトの配列
dataArray.forEach(function (data) {
  tagArray.push(`- 

```

↓↓こんな感じで出力↓↓

```
<li id="データのキーネーム">
  <p>名前</p>
  <p>時間</p>
  <p>本文</p>
</li>
```

# データのとり方のイメージ

The screenshot shows the MongoDB Atlas interface for a collection named 'chat'. A specific document with \_id 'L02qznemG9iUVsf0qv3P' is selected. A callout bubble labeled 'doc.id' points to the document ID in the list. Another callout bubble labeled 'doc.data()' points to the expanded document view on the right.

chatapp20190507

chat

doc.id

+ ドキュメントを追加

L02qznemG9iUVsf0qv3P

Yh026bUB1aWGUYToo899

gHzf1Q6fXcgGGgE8AzJH

+ コレクションを追加

+ フィールドを追加

! インターネット上の誰でもこのドキュメントの読み取りと書き込みができます

text: "11111111"  
time: "2019-06-07T18:24:21"  
user: "1"

doc.data()

# やや特殊なデータ構造と扱いのポイント

```
// 配列からのデータ取得方法  
// {...}が一つのドキュメントデータ（不要なデータも入っている）  
const data = [{...}, {...}, {...}, ...];  
  
// 配列の各要素に対して下記の処理でデータが取れる。  
const id = {...}.id;  
const data = {...}.data();  
  
// 「必要なデータだけを入れた新しい配列」を作成  
const dataArray = [];  
dataArray.push({id: id, data: data}); // <- 繰り返し処理で実行
```

# データ受信処理の動作確認

The diagram illustrates the real-time communication between two instances of a "realtime chat" application.

**Left Instance (Client A):**

- Form fields:
  - name
  - send
- Message list:
  - test  
2020/05/18 23:25:37  
www
  - test  
2020/05/18 23:25:07  
wwww
  - test  
2020/05/18 00:00:00  
hoge

**Right Instance (Client B):**

- Form fields:
  - send
- Message list:
  - test  
2020/05/18 23:25:37  
www
  - test  
2020/05/18 23:25:07  
wwww
  - test  
2020/05/18 00:00:00  
hoge

**Annotations:**

1. 片方で送信すると... (When you send from one side...)
2. 両方で表示される！ (Both sides will display it!)

# データ受信の処理を実装しよう！

---

- ここまで作ろう！
  - 送信されたデータを画面に表示！
  - 別々のウインドウで開いてリアルタイムに同期されることを確認！

【おまけ】Enterキーで送信

# メッセンジャー的な操作

```
$('text').on('keydown', function (e) {
    console.log(e)
});

m.Event {originalEvent: KeyboardEvent, type: "keydown", isDefaultPrevented: f,
timeStamp: 9446.200000005774, jQuery11130337889318682532: true, ...} ⓘ
    altKey: false
    bubbles: true
    cancelable: true
    char: undefined
    charCode: 0
    ctrlKey: false
    ► currentTarget: textarea#text
    data: undefined
    ► delegateTarget: textarea#text
    eventPhase: 2
    ► handleObj: {type: "key", origType: "keydown", data: undefined, handler: f, gu...
    ► isDefaultPrevented: f()
    jQuery11130337889318682532: true
    key: "Enter"
    keyCode: 13
    metaKey: false
```

keydownイベント

エンターキーのキーコードを確認

(キーコードが13だったら...)

# 参考情報

---

- 情報の取得
  - `console.log(e)`;を使うとイベントの様々な情報を取得できます.
  - 例えば, `keydown`したキーの番号, クリックした座標, など
- `console.log`を活用していろいろな機能を開発できる！
  - (コナミコマンドとか)
  - 【参考】<https://shgam.hatenadiary.jp/entry/2013/06/27/022956>

# 課題

# Firebaseを使ったアプリケーションを作ろう！

---

- 最低限ここまで！
  - 「名前」「日時」「メッセージ」を送信&表示
  - 見た目をいい感じに！(LINEやメッセンジャーみたいに)
- 追加仕様の例
  - スタンプ送信機能
  - オンラインでじゃんけん
  - MMORPGを開発

※例によってfirebaseを使えば何でもOK！



注意点

重要！！絶対確認！！

# ⚠️ Githubにpushするときの注意点 ⚠️

---

## APIキーの扱い

- FirebaseにはAPIキーが必要になります.
- 誰でも見られるGithubにあげてしまうとあまりよろしくない.

## Githubにpushする前に. . .

- `git add .`する前にAPIキー部分は一旦削除しておきましょう.
- 提出フォームのコメント欄にAPIキーを記述してください！

# ⚠️ Githubにpushするときの注意点 ⚠️

```
22 | <!-- The core Firebase JS SDK is always required and must be listed first -->
23 | <script src="https://www.gstatic.com/firebasejs/6.0.0.firebaseio.js"></script>
24 |
25 | <!-- TODO: Add SDKs for Firebase products that you want to use
26 |      https://firebase.google.com/docs/web/setup#config-web-app -->
27 |
28 | <script>
29 |   // Your web app's Firebase configuration
30 |   var firebaseConfig = {
31 |     apiKey: "AIzaSyB_qDNjlAe0naKGxr5vBzbcK6kL1tUVY30",
32 |     authDomain: "testapp-279d8.firebaseio.com",
33 |     databaseURL: "https://testapp-279d8.firebaseio.com",
34 |     projectId: "testapp-279d8",
35 |     storageBucket: "testapp-279d8.appspot.com",
36 |     messagingSenderId: "1:760444444444",
37 |     appId: "1:760444444444:web:1234567890123456"
38 |   };
39 |   // Initialize Firebase
40 |   firebase.initializeApp(firebaseConfig);
41 | </script>
```

`git add .`する前にこの部分を一旦削除

削除したAPIキーは「提出フォームのAPIkey欄」に記述！

締切は次回授業前木曜「23:59:59」

やばいいい. . . (｀;ω;`)

完全に詰んだ. . . という方は

# 写経

※写経とは※

誰かが書いた「動くコード」をひたすら書き写すこと

```
1  <!DOCTYPE html>
2  <html lang="ja">
3
4  <head>
5  · · <meta charset="UTF-8">
6  · · <meta name="viewport" content="width=device-width, initial-scale=1.0">
7  · · <title>chatapp</title>
8  <style>
9  · · #output li {
10 · · · · background-color: #ccc;
11 · · }
12 </style>
13 </head>
14
```

```
15 <body>
16   <h1>realtime chat</h1>
17   <ul>
18     <li>
19       <label for="name">name</label>
20       <input type="text" id="name">
21     </li>
22     <li>
23       <textarea name="" id="text" cols="30" rows="10"></textarea>
24     </li>
25     <li>
26       <button id="send">send</button>
27     </li>
28   </ul>
29   <ul id="output"></ul>
30
```

```
31 <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
32 <script src="https://www.gstatic.com/firebasejs/7.14.4.firebaseio.js"></script>
33 <script>
34   var firebaseConfig = {
35     apiKey: "AIzaSyA
36     authDomain: "cha
37     databaseURL: "ht
38     projectId: "chat
39     storageBucket: "
40     messagingSenderId:
41     appId: "1:889157
42   };
43   firebase.initializeApp(firebaseConfig);
44 </script>
45
```



z1JCY",
,  
eio.com".
3c2dd1"

APIキーは自分のものを！

```
46 <script>
47   function convertFromFirestoreTimestampToDatetime(timestamp) {
48     const _d = timestamp ? new Date(timestamp * 1000) : new Date();
49     const Y = _d.getFullYear();
50     const m = (_d.getMonth() + 1).toString().padStart(2, '0');
51     const d = _d.getDate().toString().padStart(2, '0');
52     const H = _d.getHours().toString().padStart(2, '0');
53     const i = _d.getMinutes().toString().padStart(2, '0');
54     const s = _d.getSeconds().toString().padStart(2, '0');
55     return `${Y}/${m}/${d} ${H}:${i}:${s}`;
56   }
57 
```

```
58 const db = firebase.firestore().collection('chat');
59 $('#send').on('click', function(){
60   db.add({
61     name: $('#name').val(),
62     text: $('#text').val(),
63     time: firebase.firestore.FieldValue.serverTimestamp(),
64   });
65   $('#text').val('');
66 });
67
```

```
68 db.orderBy('time', 'desc').onSnapshot(function(querySnapshot) {
69   let str = '';
70   querySnapshot.docs.forEach(function(doc) {
71     const id = doc.id;
72     const data = doc.data();
73     const datetime = convertFromFirestoreTimestampToDatetime(data.time.seconds);
74     str += '<li id="' + id + '">' //idにkey名を追加
75     str += '<p>' + data.name + '</p>';
76     str += '<p>' + datetime + '</p>';
77     str += '<p>' + data.text + '</p>';
78     str += '</li>';
79   });
80   $('#output').html(str);
81 });
82
83 </script>
84 </body>
85
86 </html>
```

「写経」はあくまでヒント！  
「オリジナルの作品」で力がつく！

# P2Pタイム

まずはチーム内で解決を目指す！

訊かれた人は苦し紛れでも応える！！