

Login Using Firebase Phone Authentication

Step 1: Initialize Firebase

Go to console.firebaseio.google.com

Login using tech@laex.in

Use project UPSC.PRO

If the FE using a new domain, add it in Firebase – under authorized domains.

Config object:

From: Project Setting -> general -> apps-> UPSC.PRO (Web App)

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "",
  authDomain: "laex-upsc-pro.firebaseio.com",
  projectId: "laex-upsc-pro",
  storageBucket: "laex-upsc-pro.firebaseiostorage.app",
  messagingSenderId: "",
  appId: "",
  measurementId: ""
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

Step 2: Request Phone Number

In frontend, prompt the user to enter their phone number with country code (e.g., +91XXXXXXXXXX).

Step 3: Send Verification Code

Use Firebase's SDK to send a verification code via SMS:

Firebase SDK Example:

```
firebase.auth().signInWithPhoneNumber(phoneNumber, appVerifier)
  .then(confirmationResult => { /* Save confirmationResult for verification */ })
  .catch(error => { console.error(error); });
```

Step 4: Verify OTP

After user receives and enters the OTP, verify it with Firebase:

```
confirmationResult.confirm(otp)
  .then(result => { const user = result.user; /* User signed in */ })
  .catch(error => { console.error('Invalid OTP'); });
```

Step 5: Manage Authentication State

Use Firebase Auth's listener to track login state:

```
firebase.auth().onAuthStateChanged(user => {
  if (user) { /* User is logged in */
    /* store IDtoken from user in a variable or local store */
  }
  else { /* User logged out */
  });
});
```

Step 6: check if number exists in the Database

Use below API to check if user exists:

POST [/v2/exists/new](#)

```
body: { "user_name":string} //pass phone number to user_name field.
response: {
  "exists": true,
  "active": true,
  "tenantId": 0,
  "admin": true,
  "domain": "string"
}
```

Step 7: If user does not exist, then register the user. This step registers as well as gets JWT token from backend.

Use POST [/v2/register](#): to create a user and get JWT token

Body: { "phoneNumber":string}

Header: {“idToken”: string, “tenant”:1} //fix tenant is to 1. // idToken is from firebase.

Response: { “access_token”:string, “refresh_token”:string}

Step 8: If user exists, then fetch JWT token from server

Use POST [/auth/token](#): to fetch a JWT token

Query: { “provider”:string} // put “PHONE_NUMBER” here (not the actual number).

Header: {“idToken”: string} // as received from firebase.

Response: { “access_token”:string, “refresh_token”:string}

Step 9: fetch user details

Use GET `/v2/user/{id}`: to fetch a user detail

Step 10: From step 9, we will know if user is “onboarded” or not. If user is not onboarded, then onboard the user using:

Use POST `/v2/users/onboard`: to save basic details

Body : {

```
"email": "user@example.com",
"fullName": "string",
"gender": "Male",
"password": "string",
"aboutMe": "string",
"userType": "STUDENT",
"photo": "string",
"referredById": 0,
"referredByPhone": "strings"
```

}

Response: updated user details

Login Using Password based Authentication

Step 1: check if number exists in the backend

Use POST [/v2/exists/new](#) to check if phone number exists

body: { "user_name":string} // pass phone number here.

```
response: {  
  "exists": true,  
  "active": true,  
  "tenantId": 0,  
  "admin": true,  
  "domain": "string"  
}
```

Step 2: If user does not exist – then ask user to use phone auth method to get registered. This validates the phone number. Once user registers using phone auth method, subsequently user can use password method. (during onboarding, we ask user to provide password).

Step 3: Login (fetch a JWT token) (use if user exists, login using password)

Body: { "phoneNumber":string,"password":string}

Response: { "access_token":string, "refresh_token":string}

Step 4: Fetch user details

Use GET [/v2/user/{id}](#): to fetch a user detail

Payment using Axis Bank - Razorpay payment link

Step 1: Get All Products

Use GET v2/products to get list of products

query: { "filters":string,}

response : { data:[] }

Step 2: Create Purchase order (it can take a list of all products in a kart)

Use POST v2/purchases to create purchase order for each of the products selected

body:

```
{
  "purchases": [
    {
      "productId": 0,
      "priceId": 0,
      "studentId": 0,
      "admissionId": 0,
      "purchaseType": "BUY",
      "pricingModel": "ONE_TIME",
      "intallmentsCount": 0,
      "quantity": 1,
      "amount": 0,
      "purchaseDate": "2025-10-07T10:20:16.797Z",
      "discountId": 0,
      "discountAmount": 0,
      "additionalDiscountId": 0,
      "additionalDiscAmt": 0,
      "purchaseDetails": {
        "subjects": [
          {
            "id": 0,
            "name": "string",
            "description": "string",
            "code": "string",
            "isOptionalSubject": true,
            "isLanguage": true,
            "benchmark": [
              {
                "stage": {
                  "id": 0,
                  "name": "string",
                  "description": "string",
                  "stageSeq": 0},
                  "averageMarksPercentage": 0,
                  "aspirationalMarksPercentage": 0
                }]],
            "optionalSubject": {
              "id": 0,
              "name": "string",
              "description": "string",
              "code": "string",
              "isOptionalSubject": true,
              "isLanguage": true,
              "benchmark": [
                {
                  "stage": {
                    "id": 0,
                    "name": "string",
                    "description": "string",
                    "stageSeq": 0},
                    "averageMarksPercentage": 0,
                    "aspirationalMarksPercentage": 0
                  }]]}
        }
      ]
    }
  ]
}
```

```

"name": "string",
"description": "string",
"s"optionalSubject": true,
"isLanguage": true,
"benchmark": [{  

"stage": {  

"id": 0,  

"name": "string",  

"description": "string",  

"stageSeq": 0  

},  

"averageMarksPercentage": 0,  

"aspirationalMarksPercentage": 0  

}],  

"duration": 0,  

"info": "string"  

},  

"legalEntityDetails": {  

"breakupName": "string",  

"tenantId": 0,  

"name": "string",  

"gstin": "string",  

"gstRate": 0  

},]  

response : { data:[  

{id,product_id...}]}

```

Notes: Legal entity details must be selected from the price break up of product with highest available gst percent .

Step 3: Add installment if is required

Use POST /v2/purchaseinstallments to add installments to purchase order generated from step 2

Body:

```
{
"priceId": 0,
"productId": 0,
"purchaseId": 0,
"installmentDate": "2025-10-07T10:43:37.802Z",
"installmentAmount": 0,
"installmentStatus": "CREATED",
"isOriginal": true,"legalEntityDetails": {
"breakupName": "string",
"tenantId": 0,
"name": "string",
"gstin": "string",
"gstRate": 0
},
```

```
"productSessions":  
  "seqId": 0, "price": {}  
response : { data:[  
  {id,product_id,purchase_id...}]}}
```

Step 4: generate payment link

Use POST /v2/purchases/pay/links to add generate payment link from server

Body: {

```
  "purchaseIds": [ 0],  
  "purchaseInstallmentIds": [0 ],  
  "studentId": 0,  
  "amount": 0,  
  "expireBy": 0,  
  "customer": {  
    "name": "string",  
    "contact": "string",  
    "email": "string"},  
  "callbackUrl": "string",  
  "callbackMethod": "get",  
  "notify": {  
    "sms": true,  
    "email": true},  
  "reminderEnable": true,  
  "notes": {},  
  "tenantId": 1,  
  "legalentityName": "string"}
```

Response:

```
{id, short_url, reference_id,...}
```

Note:

1. short_url is the payment link
2. id is the payment link id (paymentlink_id in below step)
3. reference_id is the merchant transaction id

Once front end received payment link, show it to user, or / and open it in iFrame.

Step 5: Check payment status

Use GET /v2/purchases/paymentlink/{id}/status to check the payment status from the payment link (note: in “id” pass reference_id from above step)

Query : {“paymentlink_id”:string, “legalentity_name”:string,”tenant_id”:number}

Response: {id,txId,status...}

Notes: Check payment status link API can be called for every X sec until Y times to check the payment link status in the background or use the same api to check the status (triggered by user, say Refresh button).

Status values:

1. COMPLETED
2. FAILED
3. PENDING

Step 6: Create admission

Use POST [/v2/admission](#) to create admission for the user

Body: {

```
"userId": number,  
"walkinId": number,  
"admissionDate": date( yyyy-mm-dd)  
"branchId": number,  
"status": "NEW",  
"signedAdmissionFormUrl": null,  
"admissionDetails": {  
    "admissionManagerName": string,  
    "admissionManagerId": number,  
}  
}
```

Response :{id,branch,..}

Step 7: Create enrollment

Use POST [/v2/admission](#) to enroll student into a product

Body:{ "enrolledAs": string // "STUDENT",
"enrolledUserId": number,
"admissionId": number,
"batchId": number,
"offeringId": number,
"productId": number,
"enrollmentStatus": string // "ACTIVE",
}

Response: {id,...}

Step 8: Update purchase order

Use POST [/v2/purchases/{id}](#) to update purchase order with admission Id from step 6

Body:{ "enrolledAs": string // "STUDENT",
"enrolledUserId": number,
"admissionId": number,
"batchId": number,
"offeringId": number,
"productId": number,
"enrollmentStatus": string // "ACTIVE",
}

Response: {id,...}