BuildingBlocks

(version 8.0.0)

**Github repo :**

App source file can be found in the below given repo.

<https://github.com/CallystroInfotech/BuildingBlocks-HTML5>

you can clone or download this source code.

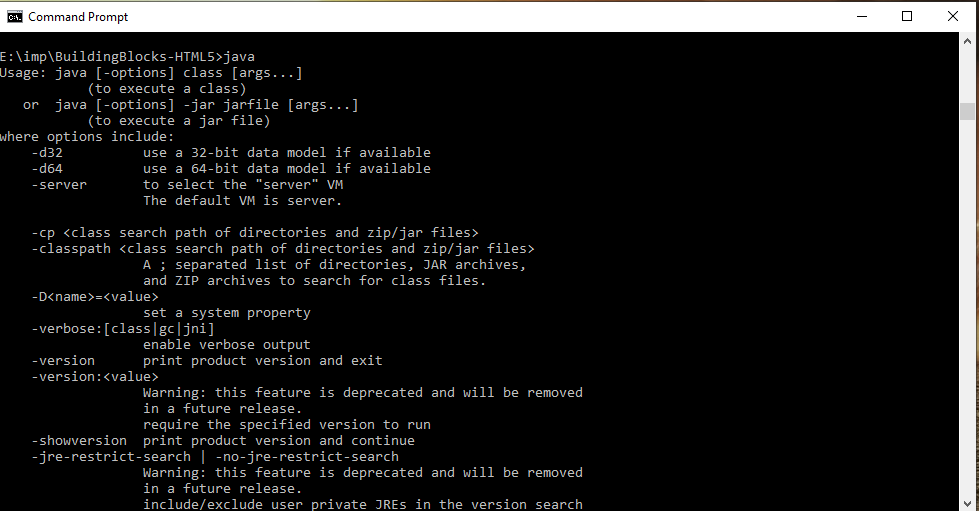
**Cordova installation :**

Prerequisites:

1. You need to have installed the java jdk tools in your system

Link:<https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html>

Check if java is installed in your system. Type java in command prompt to check.



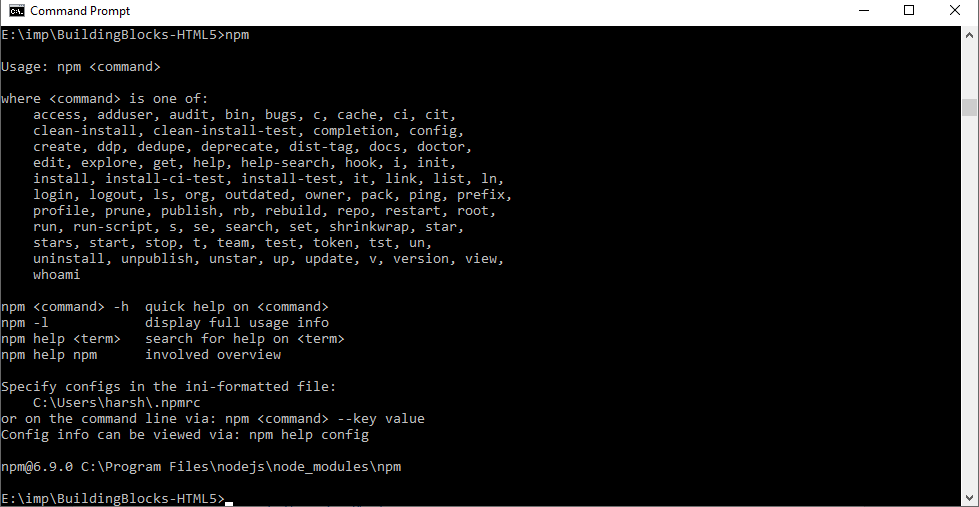
1. Install nodejs:

To install cordova in your system you need to install node js first.

Link:<https://nodejs.org/en/>

Once you have installed node js you could install cordova packages using npm.

Check npm installation, Type npm in command prompt to check.



1. Install gradle packages:

Link:<https://gradle.org/install/>

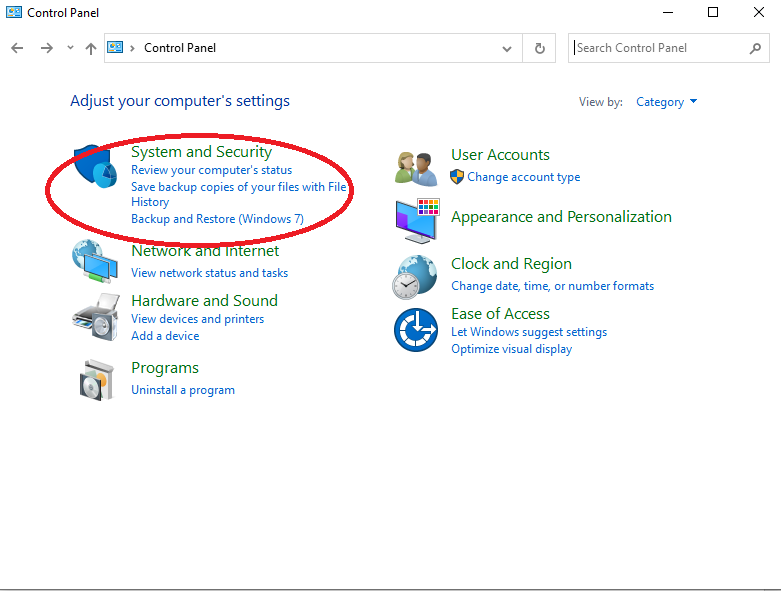
Download the zip file from the above link.

Create a new directory inside c: named “Gradle”

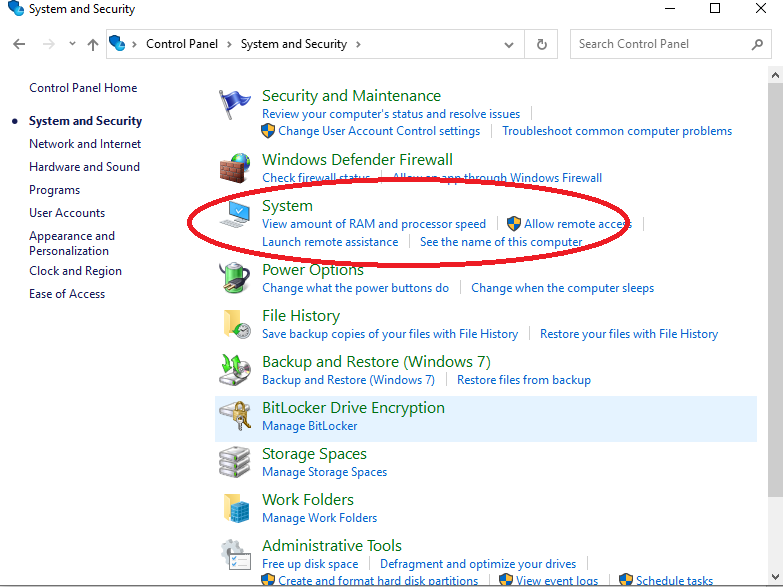
Unzip the downloaded file.

Copy gradle-6.5 to your newly created C:\Gradle folder.

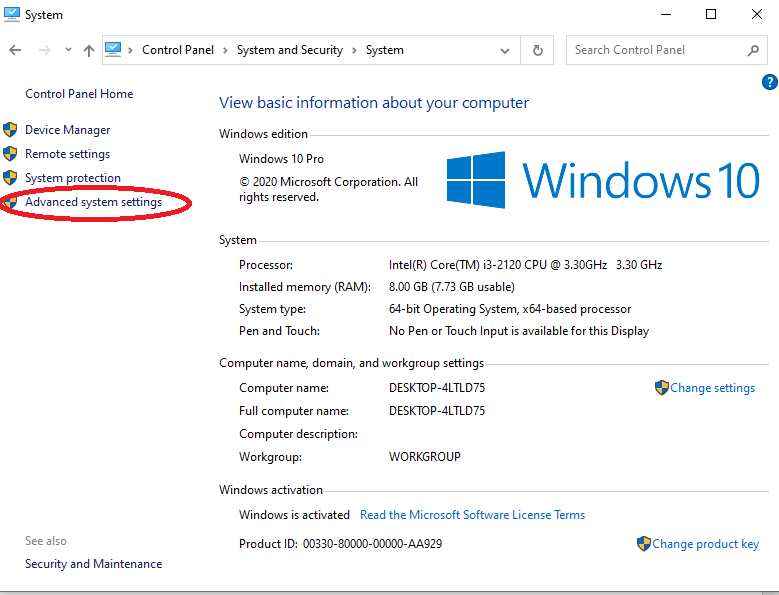
Next go to control panel-> system and security



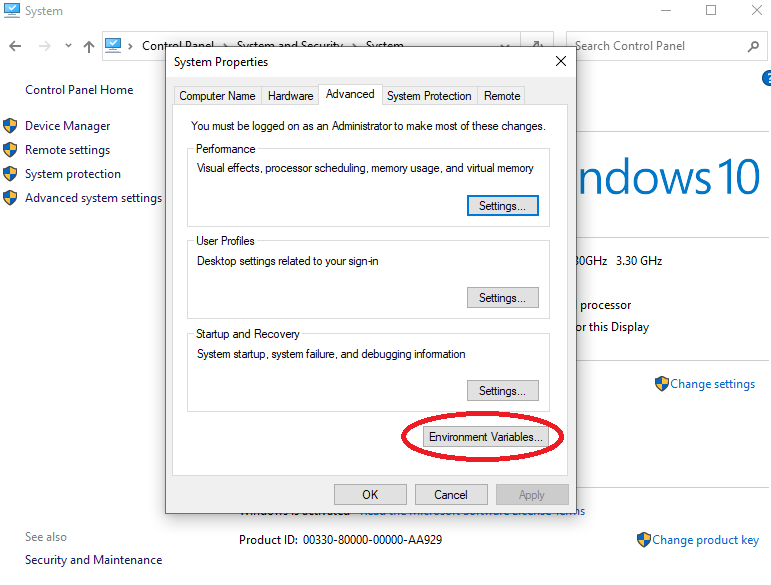
Next go to ->System



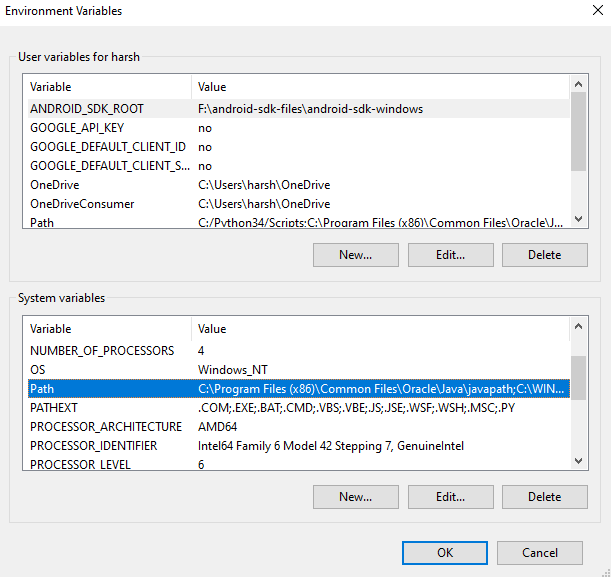
Click on Advanced system settings



Click on “environmental Variables” from the popup

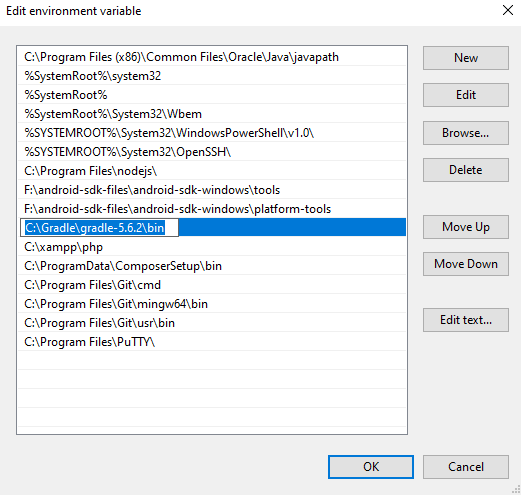


Goto System variables -> select path and click edit.



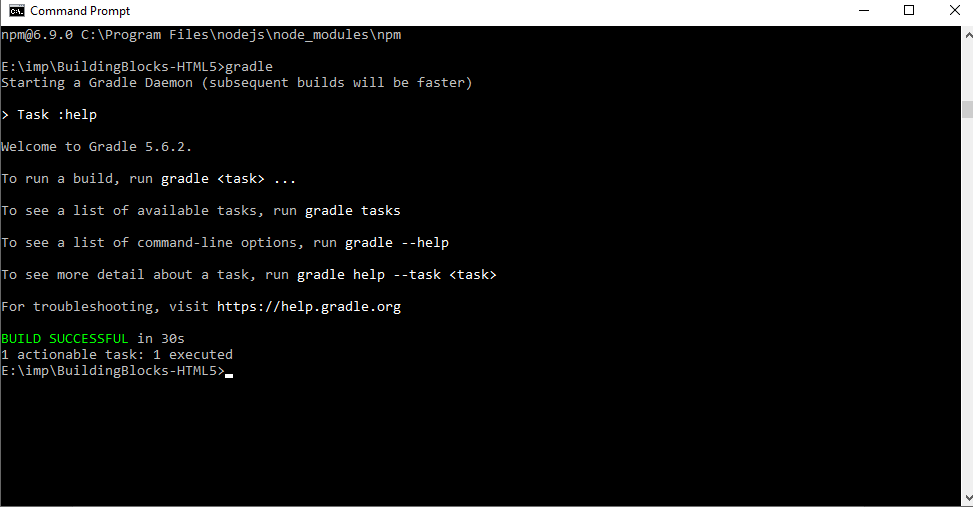
Click new add gradle path here:

Path: C:\Gradle\gradle-5.6.2\bin



Click ok and close.

You can now check in command prompt if gradle is installed by typing gradle.



4)you need to have android sdk tool packages in your system.

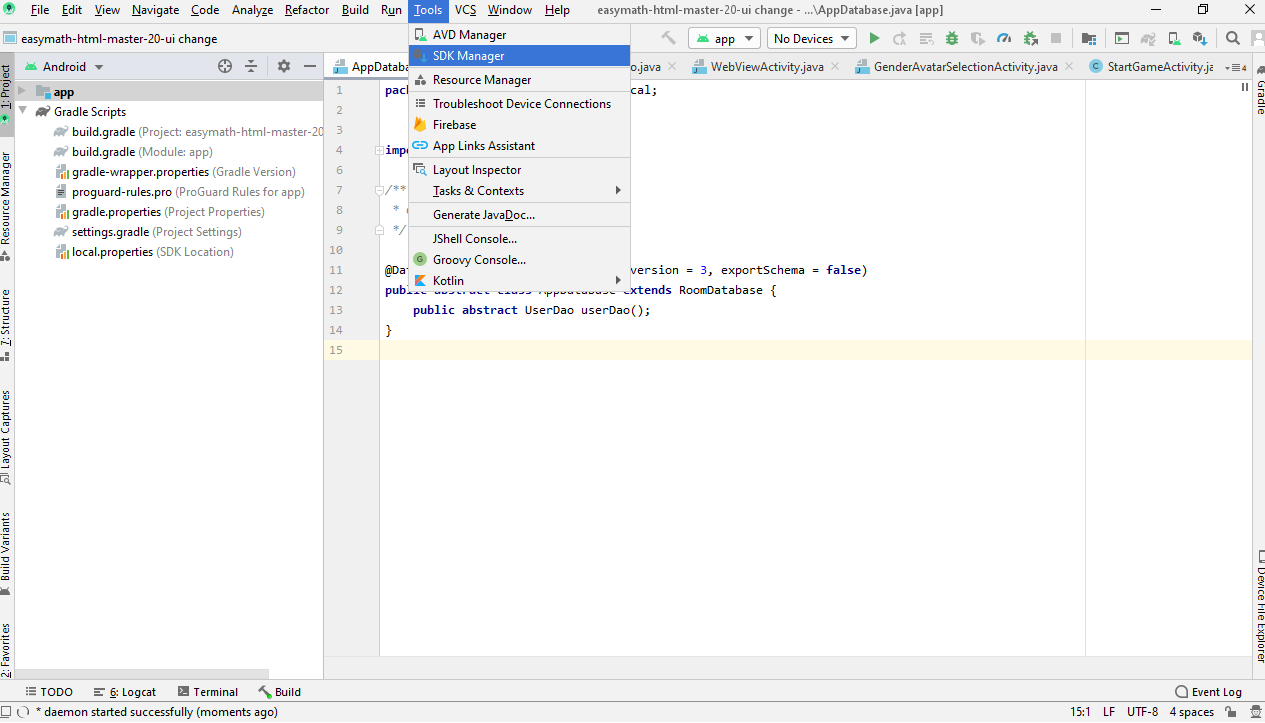
To download sdk tools in your system you can download and install “android studio”

Which will automatically download the sdk tools in your system.

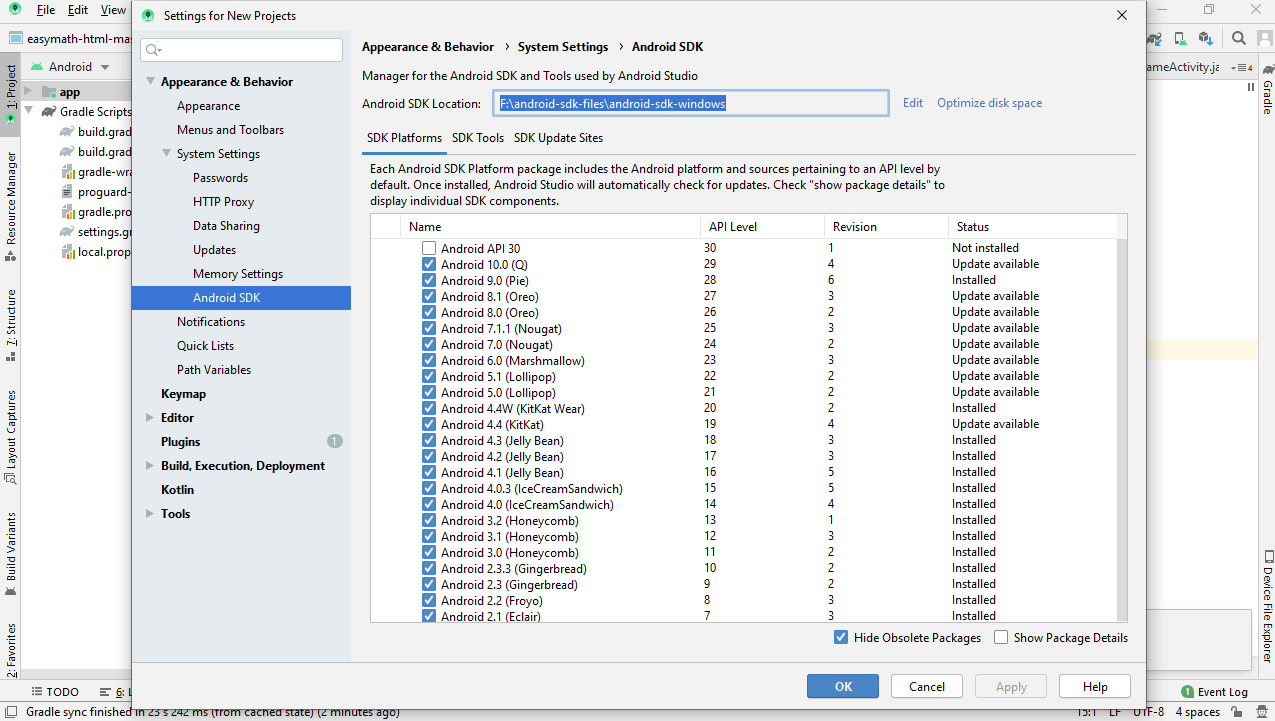
Android studio link -<https://developer.android.com/studio>.

Once android studio and sdk are installed, open android studio

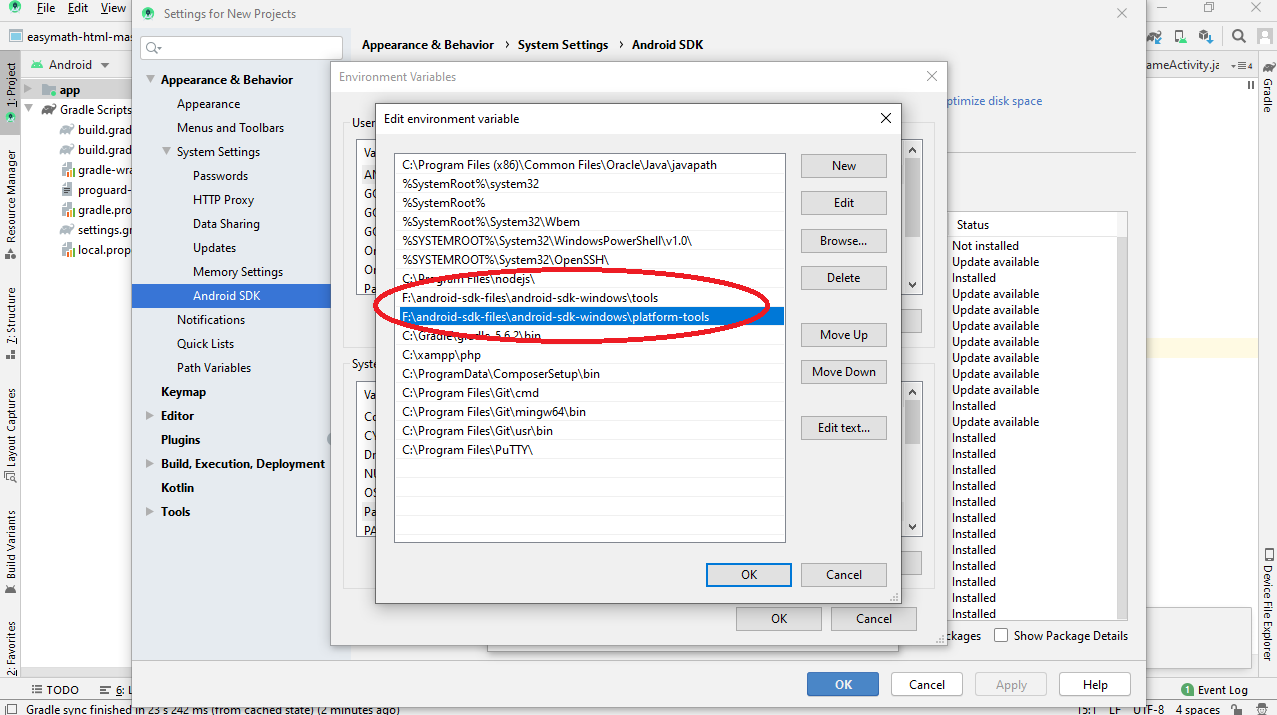
In navigation bar-> goto tools ->sdk manager.



Copy the sdk location and open the path in file explorer.



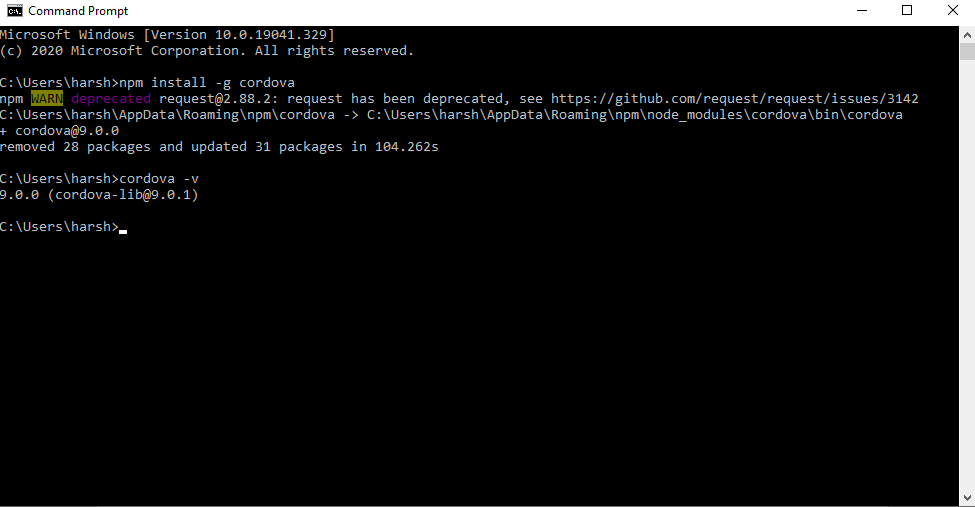
Copy platform-tools and tools path and add it to the environment variables similar to setting gradle path.



Prerequisites are done now you can install Cordova.

Open command prompt type:

npm install -g cordova



Type in “cordova -v” to check.

For more details you can check cordova document.

Link -<https://cordova.apache.org/>

Once cordova is installed navigate to the “Building blocks” source file path in your command prompt

Type “cordova build android” this should download the necessary packages required.

**Creating a release build:**

you can create a debug build by typing “cordova build android”

this will create a debug build in following directory

BuildingBlocks-HTML5\platforms\android\app\build\outputs\apk\debug\app-debug.apk

This is unsigned-debug build cannot be uploaded to playstore.

To create release build to upload to playstore you need to create a signed in apk.

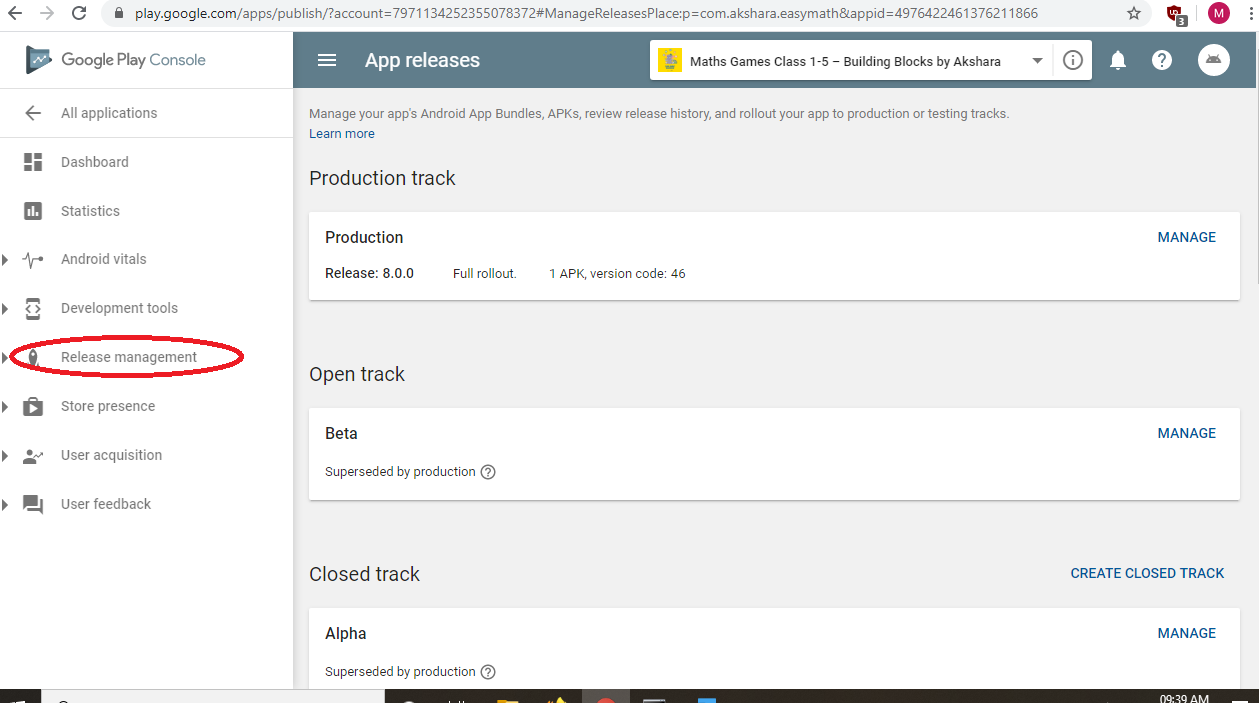
For that you need to create a “release-signing.properties” file.

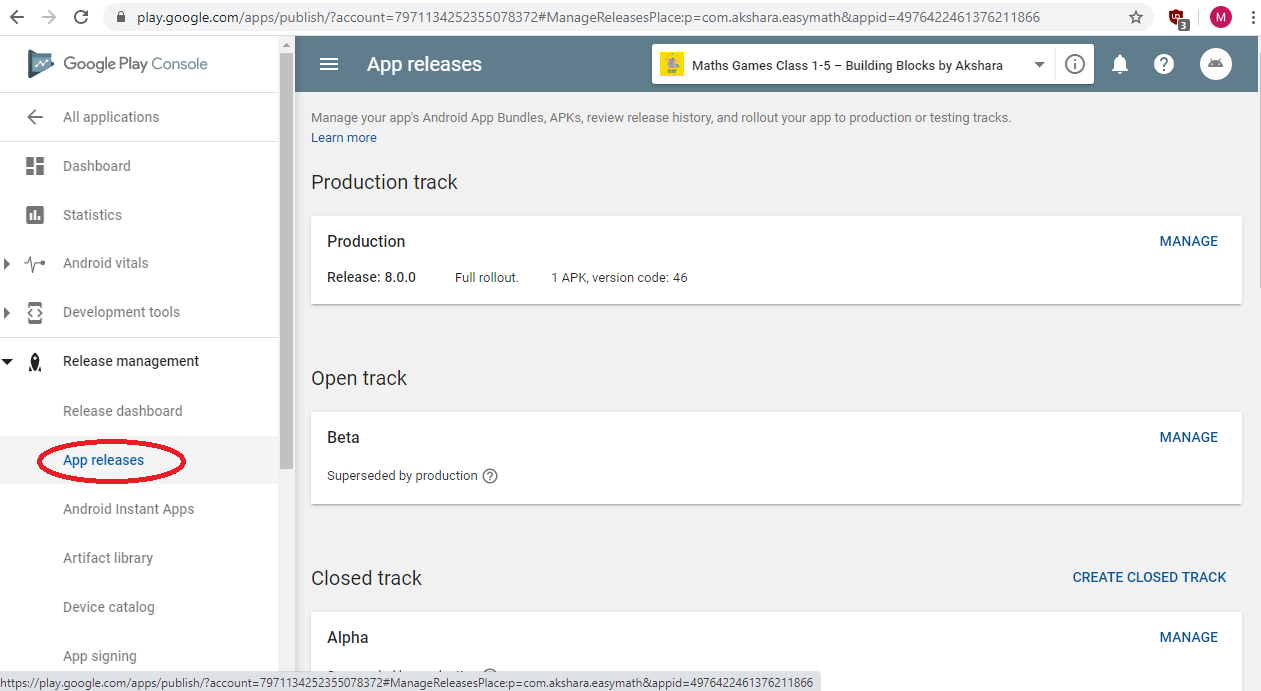
This file is already created and resides in – “BuildingBlocks-HTML5\platforms\android” folder

To create a release apk you just need to type “cordova build android --release” this creates signed apk. This apk can be uploaded to playstore.

**Uploading to playstore:**

login to playtsore console -> open building blocks -> goto release management -> App release.

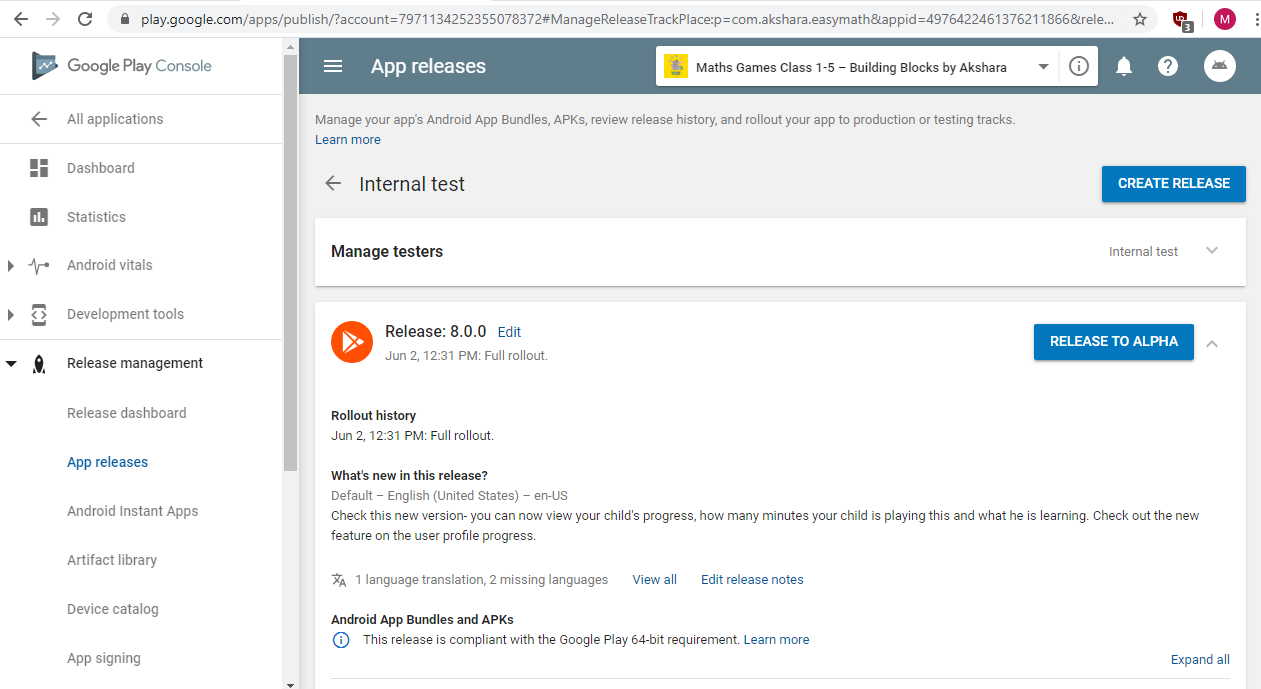




Here you can find 4 upload tracks.

1. Production track – If you release a build here it will be live globaly.
2. Open beta track – this release will be available for people who signed in for beta testers for building blocks app.
3. Closed alpha track – these are available for email id’s which are added as testers(google play developers will review the app before publishing, secure ,recommended).
4. Internal test track - these are available for email id’s which are added as testers(google play developers will not review the app before publishing, not secure).

Uploading the apk file click on any track you want to publish the app->goto manage->



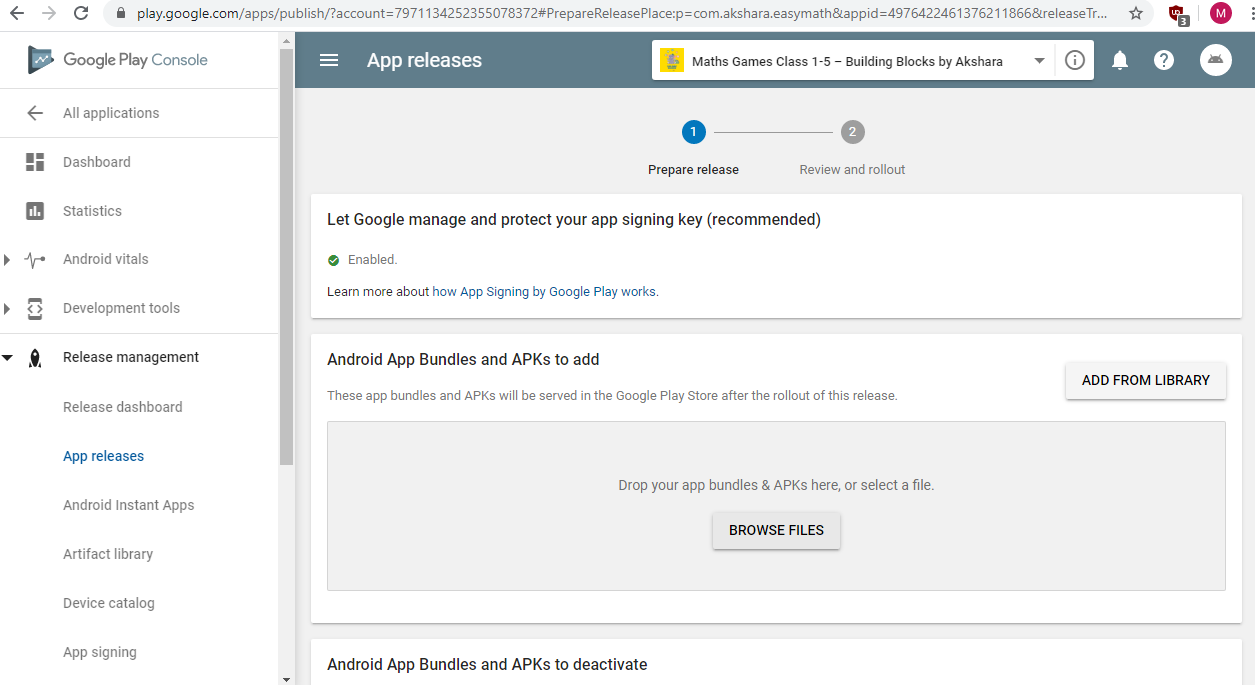
Click create release.

Here you can drag and drop the apk or simply browse the file.

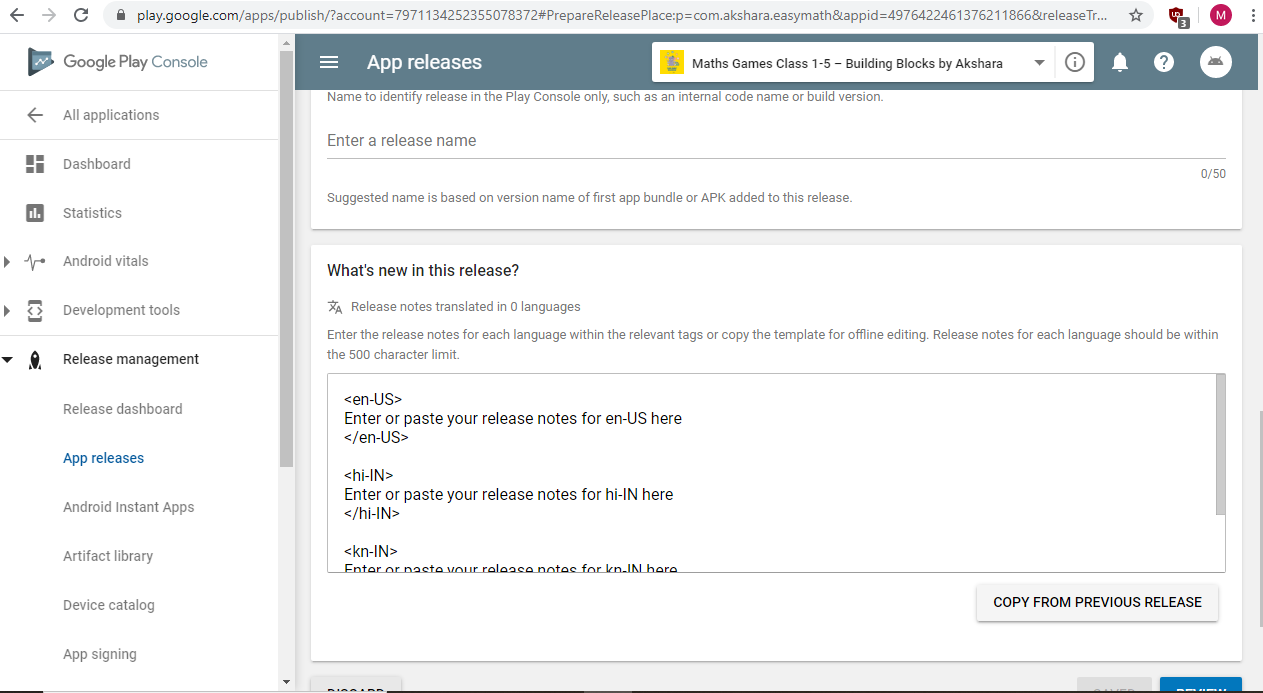
before creating the release apk always update the build version and build number in the config.xml file.

android-versionCode="46"

version="8.0.0"

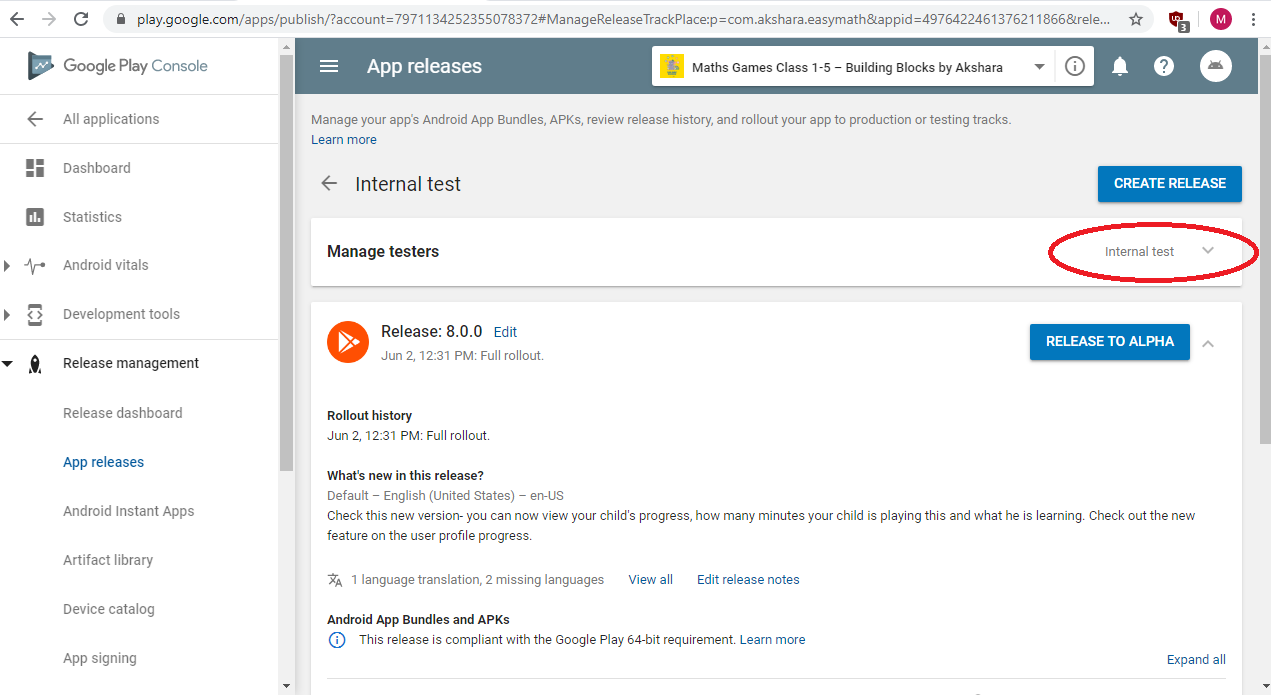


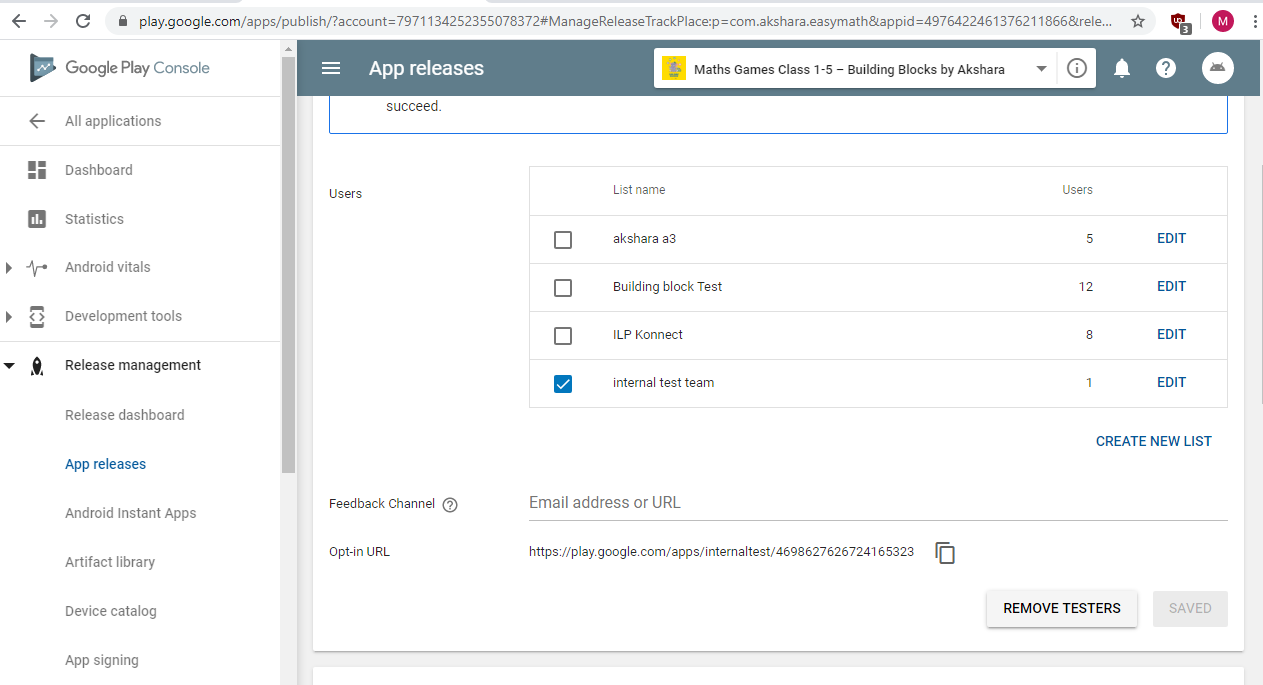
Fill in the required details. Review and click publish.



Managing app testers.

You can click on Manage testers -> create new list or edit new list just by adding the persons email ID -> save.





**Building Blocks:**

Architecture diagram:

User

Mobile device

Application Layer

Database Layer

Database Layer

Application Layer

Presentation Layer

User interface

Registration/login

Assets Download

Game logic

Technology used: Html5, js, phaser js, jquery, Ajax, hammer js.

Telemetry Api

Local storage Api

Server synchronize

Technology used: jquery, Ajax, cordova sqlite plugin.

Local storage

Data sync

Data utilities

Technology used: cordova sqlite plugin.

Technology used: cordova, cordova plugins, cordova android platform( apk packaging).

Registration Api

Telemetry Api

Queries/Entities

Technology used: php.

Technology used: mysql database.

Data Access

Html5 hybrid app created using cordova, phaser js, php for api, mysql for database.

Cordova-Apache Cordova is a mobile application development framework.

Detailed description -<https://cordova.apache.org/docs/en/latest/>

Phaser js- used for creating the game contents for the app

Detailed description -<https://phaser.io/download/phaserce>

BuildingBlocks-html5 -> www ->

->Assets - where all the game assets resides png files.

->Js - where all the javscript files for the game resides.

->json – when we use sprite sheet it requires json data and all the png json data files resides inside this folder.

->questionSounds- all the game question sounds are in this folder.

->sounds- all the sounds like celebration, wrong answer etc can be found here.

->Index.html file this is an important file which is present to load and start the contents.

Assets folder contains subfolders->common assets(contains files which are common for all games)

demoVideos->contains all the demo videos used for games.

demoVo’s->contains voice overs used for demo videos.

gradeAssets-> contains all the assets required for the game individualy.

Js folder contains subfolders.

Commonjsfiles->contains all the common javascript files.

Gradejs->content javascript files individualy.

gradeSelectionScreenJS->containg script files for game selection, grade selection etc.

registrationAndLogin-> contains all the script files related to registration and login.

Indes.js this file gets loaded when the app starts this file contains code related to initializing db, initializing phaser etc.

abbchmprmdsjsapilib-1.0.js – used for app api interaction.

hammer.min.js – used for app scrolling.

jquery.min.js – used for ajax calls.

phaser.min.js – for creating phaser games.

Preloader.js – for loading the common game assets, sounds, videos etc.

Shake.js – used for wrong animation in games.

Userprogress.js – to track the user progress in challenge mode.

challengeModeGgradeSelectionScreen.js – used to load challenge mode grade selection screen.

gameModeSelectionScreen.js – used for selection screen of selecting practice or challenge mode.

Practice.js-used for loading practice mode games.

quizCommonFile.js- used for daily quiz.

selectgrade1MicroConceptScreen.js – used for selecting the microconcept like measurement, numbersence etc for grade1.

Selectgrade2MicroConceptScreen.js – used for selecting the microconcept like measurement, numbersence etc for grade2.

Selectgrade3MicroConceptScreen.js – used for selecting the microconcept like measurement, numbersence etc for grade3.

Selectgrade4MicroConceptScreen.js – used for selecting the microconcept like measurement, numbersence etc for grade4.

Selectgrade5MicroConceptScreen.js – used for selecting the microconcept like measurement, numbersence etc for grade5.

grade1MeasurementScreen.js- used for loading contents of measurement for grade1.

Grade2MeasurementScreen.js-used for loading contents of measurement for grade2.

Grade3MeasurementScreen.js-used for loading contents of measurement for grade3.

Grade4MeasurementScreen.js-used for loading contents of measurement for grade4.

Grade5MeasurementScreen.js-used for loading contents of measurement for grade5.

grade1NumberOperationScreen.js-used for loading contents of NumberOperation for grade1.

Grade2NumberOperationScreen.js-used for loading contents of NumberOperation for grade2.

Grade3NumberOperationScreen.js-used for loading contents of NumberOperation for grade3.

Grade4NumberOperationScreen.js-used for loading contents of NumberOperation for grade4.

Grade5NumberOperationScreen.js-used for loading contents of NumberOperation for grade5.

Grade1,2,3,4,5NumberScenseScreen.js-used for loading contents of NumberSense for grade1,2,3,4,5.

Grade1,2,3,4,5Shapes.js-used for loading contents of Shapes for grade1,2,3,4,5.

Grade1,2,3,4,5DataHadling.js-used for loading contents of DataHadling for grade1,2,3,4,5.

measurement1,2,3,4 – challenge mode measurement games loading screen.

numberoperation1,2,3,4 – challenge mode numberoperation games loading screen.

numbersence1,2,3,4,5,6,7 – challenge mode numbersense games loading screen.

appLoginScreen.js – starting screen of the app where we login using avatar.

Bbregloginuserdb.js – database initialization, updation etc for the user registration and login.

appLoginEditScreen.js – used for downloading assets, game start screen.

editLangScreen.js – used for editing the language for particular avatar.

index2- after registration and game start button is clicked this file is called to load all the game contents.

Nativeapp.js – in this we have all the native functionality we using for the app like share feature, firebase etc.

registrationLangSelectionScreen.js – this is loaded when the user clicks on register button and taken to language selection screen

registrationPicSelectionScreen.js – this loaded when user needs to select avatar for registration.

Userdoa.js – user structure for user database.

telemetryInitializer.js – used to store telemetry data from the contents to the database.

**Packaging the content for Diksha:**

you can copy the assets, json, questionsounds, sounds, and js file for the particular content.

Remove all other files like registration, game selection database, telemetry files etc only particular content file needs to be copied.

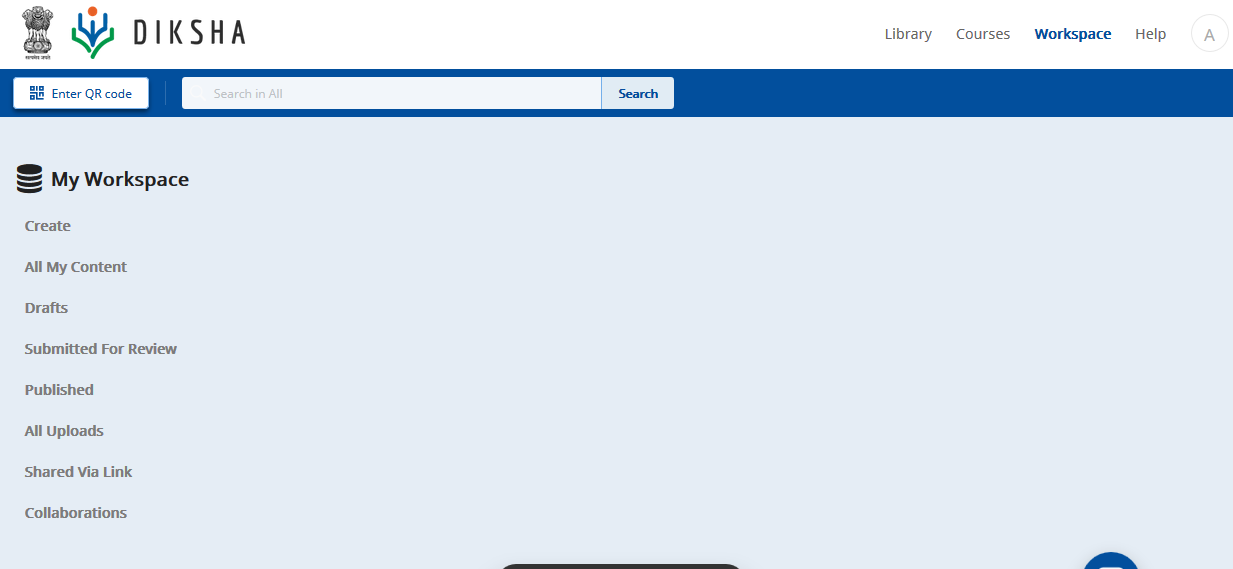
You need to separate out the content from the existing building block package.

Create a zip file for the content which contains all the necessary assets, json, questionsounds, sounds, and js files.

Once the zip file is ready login to diksha platform.

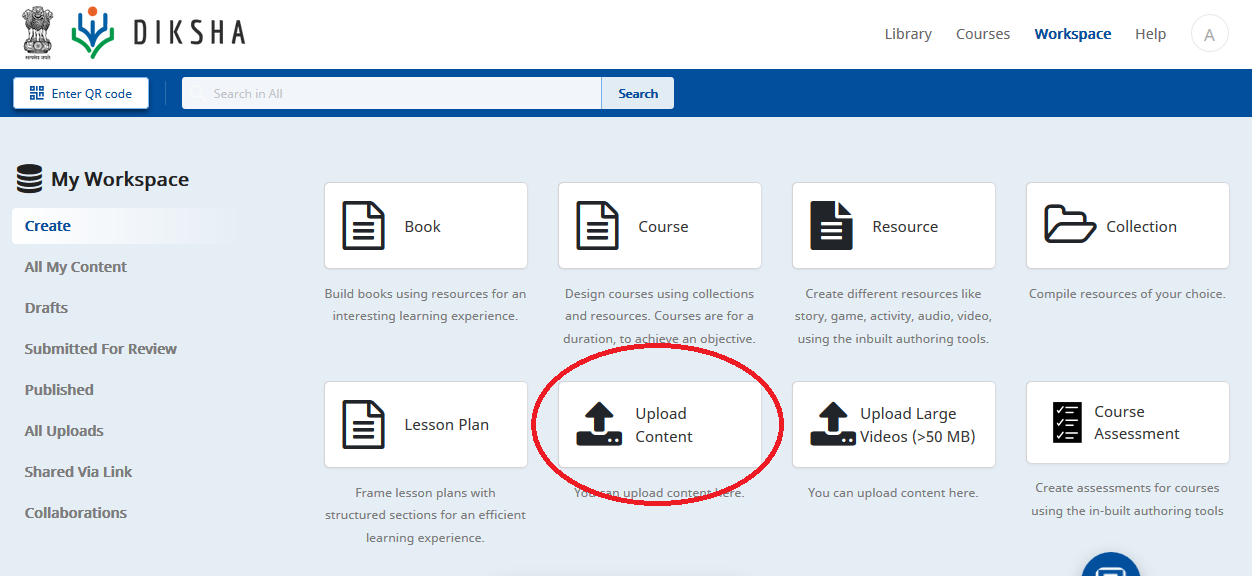
<https://diksha.gov.in/>

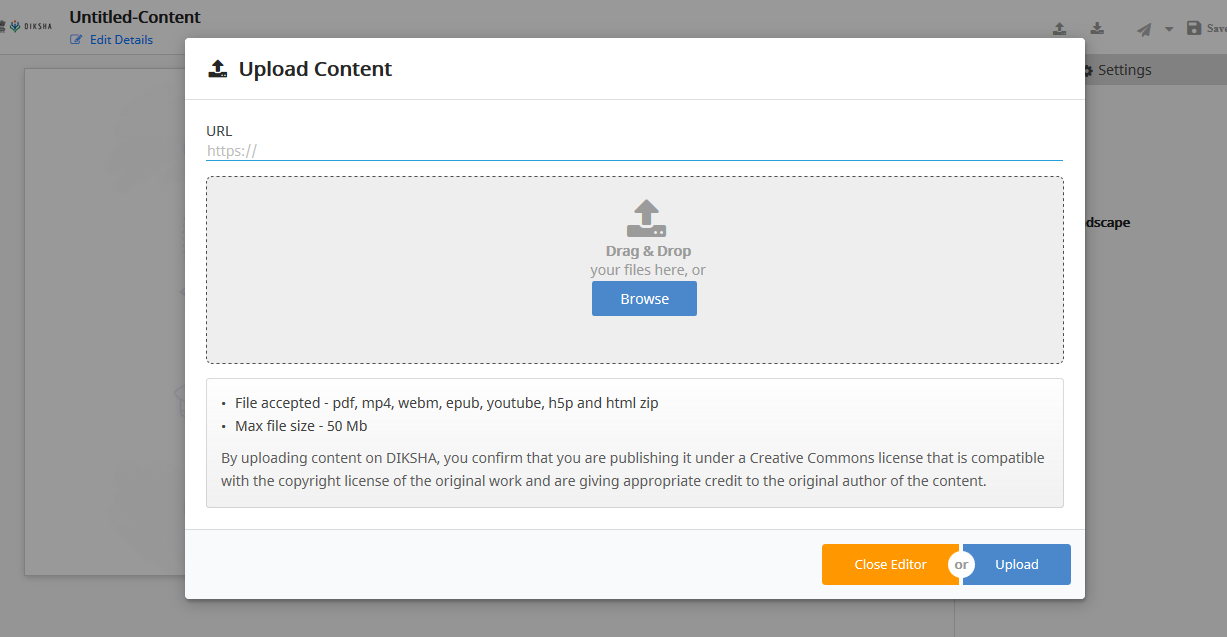
goto<https://diksha.gov.in/workspace/content/>



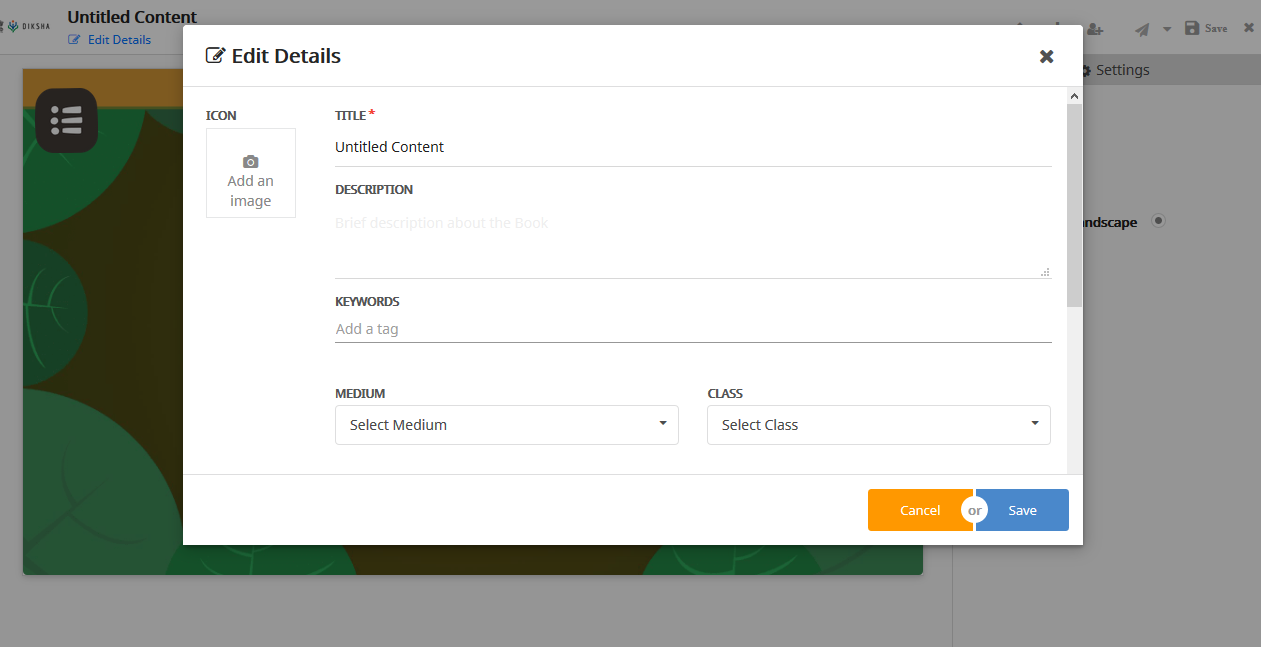
Here you can use create to create new content, if you want to update alredy existing content then click on all my content search for the particular content and edit.

After that it will show this screen, where you need to select the upload the zip file.

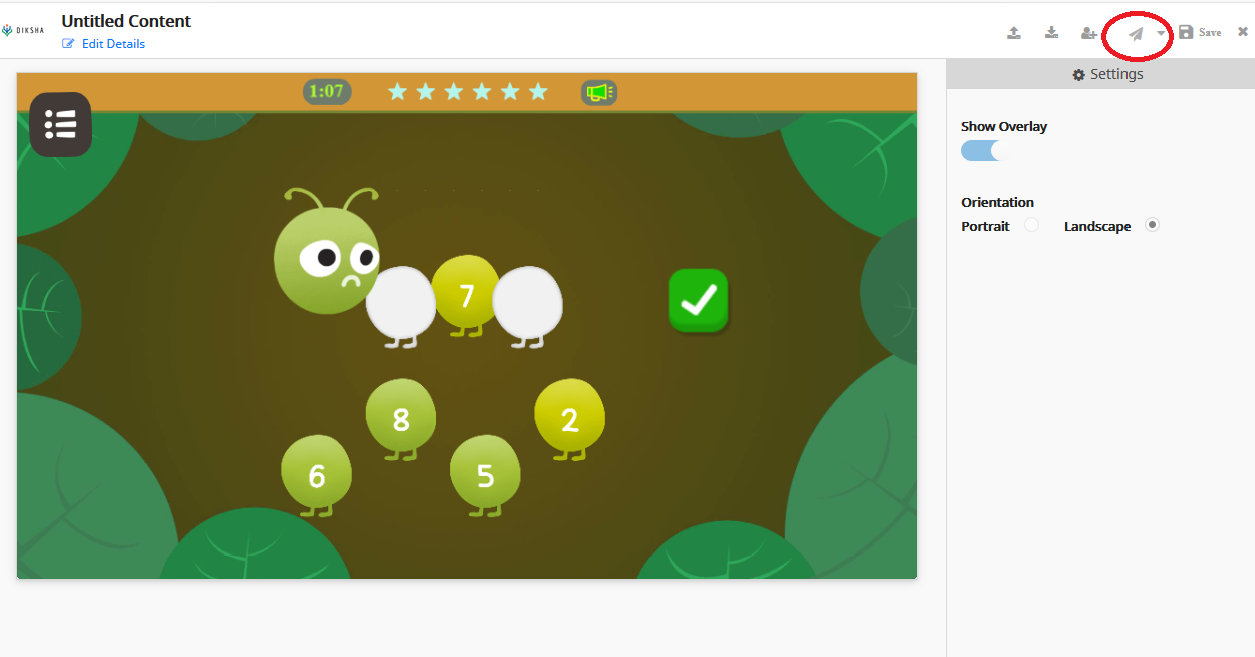




Once content is uploaded test and fill in the details and save it.



Once finished send it to review by clicking on the send for review button.



Note: content creator account is used for creating the content.

Reviewer account is used for reviewing and publishing the contents.

Once content is sent to review, login in with reviewer account and review the content and publish it.

Content will be then live in diksha.

FAQs:

1. release-signing.properties file:

Location- BuildingBlocks-html5/platforms/android

storeFile - relative path of the jks file. Change the path to where the jks file is located.

storeType=jks

keyAlias=aksharabuildingblockskey //alias used for the key file

keyPassword=[Aksharabb@org.in](mailto:Aksharabb@org.in) //password for key file

storePassword=[Aksharabb@org.in](mailto:Aksharabb@org.in) //password created for store file.

1. Importance of jks file :

It's a **file** type to store a set of cryptographic keys. They can be used to identify the author of an **Android** app during a build and when publishing to **Android** Market in Google Play or in ssl encryption.

A **keystore** has a password because it is a security-related entity. The purpose of a key store is to protect the privacy and integrity of cryptographic keys using password-based algorithms

Keep your keystore and private key in a safe and secure place, and ensure that you have secure backups of them. If you publish an app to Google Play and then lose the key with which you signed your app, you will not be able to publish any updates to your app, since you must always sign all versions of your app with the same key.

1. Backward compatibility-

Right now we are using min-sdk version to 19(ie android kitkat 4.4).

We must ensure cordova plugins support min-sdk version , some plugins might not support this versions, so we need to check the compatibility of the plugins.

You can change the min-sdk version and target sdk version in config.xml file.

Ensure sdk manager is updated before changing the version in config.xml file.

1. Updating cordova, gradle and other build tools-

Gradle- for updating gradle you can use the following command in terminal-

“gradle wrapper --gradle-version 5.0”

Change the version to latest gradle version.

Cordova - cordova can be updated by using following command

“npm *update* -g *cordova”*

Cordova plugins- to update plugins you can simply use the following command it will find the latest plugin version and update it

npm install -g cordova-check-plugins

cordova-check-plugins --update=auto

1. Api’s used in app:

**CHALLENGE MODE REST APIs:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API** | **JSON Payload** | **JSON**  **Response** | **Parameter**  **description** | **Comments** |
| txabbchmgameplay | [ | [ | **objid:** Identifier for this | The ‘txabbchmgameplay’ API |
|  | { | { | object/packet | may be invoked to transmit the |
|  | objid, | objid, | **avatarname:** | Challenge Mode ‘gameplay’ |
|  | avatarname, | status, | avatarname | telemetry data. |
|  | deviceid, | description | (childname) of the User |  |
|  | id\_game\_play, | }, | **deviceid**: ID of the | ‘objid’ is an identifier for the |
|  | id\_game, | . | device | object/packet (this can be used |
|  | start\_time, | . | **id\_game\_play**: ID of | to check the ‘response’ to see |
|  | hints | . | the game play (a 10- | if this object/packet is valid |
|  | }, | objid, | char unique ID) | and was successfully received |
|  | . | status, | **id\_game**: ID of the | by the Server) |
|  | . | description | game |  |
|  | . | } | **start\_time**: Starting | The ‘id\_game\_play’ may be a |
|  | { | ] | Time of the game play | 15-char unique string to |
|  | objid, |  | **hints**: Number of hints | uniquely identify the |
|  | avatarname, |  | taken/used while | ‘gameplay’ (to be generated by |
|  | deviceid, |  | playing this game | the mobile game/app). |
|  | id\_game\_play, |  |  |  |
|  | id\_game, |  | **status**: ‘failed’ if the | The response also is an array of |
|  | start\_time, |  | operation fails. | JSON response objects. Each |
|  | hints |  | ‘success’ if the | object provides the status for |
|  | } |  | operation succeeds | corresponding payload JSON |
|  | ] |  | **description**: reason for | object transmitted. |
|  |  |  | the failure (if ‘status’ is |  |
|  |  |  | ‘failed’), success |  |
|  |  |  | message otherwise |  |
| txabbchmgameplaydetail | [ | [ | **id\_game\_play**: ID of | The ‘txabbchmgameplaydetail’ |
|  | { | { | the game play (a 10- | API may be invokedto send the |
|  | objid, | objid, | char unique ID) | Challenge Mode |
|  | avatarname, | status, | **id\_question**: ID of the | ‘gameplaydetail’ telemetry |
|  | deviceid, | description | question/screen | data to ABB Server. The |
|  | id\_game\_play, | }, | **pass**: ‘Yes’ or ‘No’ (if | payload for the API is an array |
|  | id\_question, | . | the child has answered | of telemetry JSON objects (so |
|  | pass, | . | the question correctly) | that the mobile app can send |
|  | time2answer, | . | **time2answer**: time | multiple telemetry objects in |
|  | date\_time\_submission | objid, | taken to answer the | one shot, especially while |
|  | }, | status, | question (in secs) | syncing offline data together |
|  | . | description | **date\_time\_submission**: | when the device goes online). |
|  | . | } | DATETIME of the final |  |
|  | . | ] | submission | The response also is an array of |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | { |  |  | JSON response objects. Each |
| objid, | object provides the status for |
| avatarname, | corresponding payload JSON |
| deviceid, | object transmitted. |
| id\_game\_play, |  |
| id\_question, |  |
| pass, |  |
| time2answer, |  |
| date\_time\_submission |  |
| } |  |
| ] |  |
| txabbchmwalletscore | [ | [ | **score:** walletscore for | The ‘txabbchmwalletscore’ API |
|  | { | { | the Child | may be invoked to send |
|  | objid, | objid, | **datetime\_lastupdated**: | the‘walletscore’ value to ABB |
|  | avatarname, | status, | Date/Time of latest | Server. The payload for the API |
|  | deviceid, | description | score update | is an array of telemetry JSON |
|  | score, | }, |  | objects (so that the mobile app |
|  | datetime\_lastupdated | . |  | can send multiple telemetry |
|  | }, | . |  | objects in one shot, especially |
|  | . | . |  | while syncing offline data |
|  | . | objid, |  | together when the device goes |
|  | . | status, |  | online). |
|  | { | description |  |  |
|  | objid, | } |  | The response also is an array of |
|  | avatarname, | ] |  | JSON response objects. Each |
|  | deviceid, |  |  | object provides the status for |
|  | score, |  |  | corresponding payload JSON |
|  | datetime\_lastupdated |  |  | object transmitted |
|  | } |  |  |  |
|  | ] |  |  |  |
| rxabbchmgetwalletscore | {  avatarname, deviceid,  } | {  status, description, childid, avatarname, deviceid, score,  datetime\_lastupdated  } |  | The ‘rxabbchmgetwalletscore’ API may be invoked to fetch the value of the walletscore for a Child from the ABB Server.  Input is a JSON Object. Response is also a JSON Object. |
| rxabbchmgetgamemasterdata | No Inputs | { |  | The API |
|  |  | status, | ‘rxabbchmgetgamemasterdata’ |
|  |  | description, | may be invoked to download |
|  |  | id\_game, | the Game Master Data |
|  |  | game\_description, |  |
|  |  | grade, |  |
|  |  | gametoopen, |  |
|  |  | prerequisitegame |  |
|  |  | } |  |

# **PRACTICE MODE REST APIs:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API** | **JSON Payload** | **JSON**  **Response** | **Parameter**  **description** | **Comments** |
| txabbprmgameplay | [ | [ | **id\_game\_play**: ID of | The ‘txabbprmgameplay’ API may be |
|  | { | { | the game play (a 10- | invoked to transmit the Practice Mode |
|  | objid, | objid, | char unique ID) | ‘gameplay’ telemetry data. |
|  | avatarname, | status, | **id\_game**: ID of the |  |
|  | deviceid, | description | game | ‘objid’ is an identifier for the object/packet |
|  | id\_game\_play, | }, | **start\_time**: Starting | (this can be used to check the ‘response’ |
|  | id\_game, | . | Time of the game play | to see if this object/packet is valid and was |
|  | start\_time | . |  | successfully received by the Server) |
|  | }, | . |  |  |
|  | . | objid, |  | The ‘id\_game\_play’ may be a 10-char |
|  | . | status, |  | unique string to uniquely identify the |
|  | . | description |  | ‘gameplay’ (to be generated by the mobile |
|  | { | } |  | game/app). |
|  | objid, | ] |  |  |
|  | avatarname, |  |  | The response also is an array of JSON |
|  | deviceid, |  |  | response objects. Each object provides the |
|  | id\_game\_play, |  |  | status for corresponding payload JSON |
|  | id\_game, |  |  | object transmitted. |
|  | start\_time |  |  |  |
|  | } |  |  |  |
|  | ] |  |  |  |
| txabbprmgameplaydetail | [ | [ | **id\_game\_play**: ID of | The ‘txabbprmgameplaydetail’ API may be |
|  | { | { | the game play (a 10- | invokedto transmit the Practice Mode |
|  | objid, | objid, | char unique ID) | ‘gameplaydetail’ telemetry data to ABB |
|  | avatarname, | status, | **id\_question**: ID of the | Backend Server. The payload for the API is |
|  | deviceid, | description | question/screen | an array of telemetry JSON objects (so that |
|  | id\_game\_play, | }, | **pass**: ‘Yes’ or ‘No’ (if | the mobile app can send multiple |
|  | id\_question, | . | the child has answered | telemetry objects in one shot, especially |
|  | pass, | . | the question correctly) | while syncing offline data together when |
|  | time2answer, | . | **time2answer**: time | the device goes online). |
|  | attempts, | objid, | taken to answer the |  |
|  | date\_time\_submission | status, | question (in secs) | The response also is an array of JSON |
|  | }, | description | **attempts**: number of | response objects. Each object provides the |
|  | . | } | attempts before the | status for corresponding payload JSON |
|  | . | ] | final submission | object transmitted. |
|  | . |  | **date\_time\_submission**: |  |
|  | { |  | DATETIME of the final |  |
|  | objid, |  | submission |  |
|  | avatarname, |  |  |  |
|  | deviceid, |  |  |  |
|  | id\_game\_play, |  |  |  |
|  | id\_question, |  |  |  |
|  | pass, |  |  |  |
|  | time2answer, |  |  |  |
|  | attempts, |  |  |  |
|  | date\_time\_submission |  |  |  |
|  | } |  |  |  |
|  | ] |  |  |  |
| txabbprmekstepevents | [ | [ | **id\_game\_play**: ID of | The ‘txeksteptelemetry’ API may be |
|  | { | { | the game | invoked to transmit the EkStep events |
|  | objid, | objid, | **id\_question**: ID of the | (corresponding to individual |
|  | avatarname, | status, | question/screen | id\_game\_play/id\_game\_play\_detail). |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | deviceid, | description | **ekstep\_eventid**: EkStep |  |
| id\_game\_play, | }, | Event ID (e.g | The “edata” object is the EkStep event |
| id\_question, | . | OE\_INTERACT) | data structure/JSON object for the event |
| ekstep\_eventid, | . | **date\_time\_event**: | as specified in the EkStep Telemetry API |
| date\_time\_event, | . | DATETIME of the event | Specification |
| edata | objid, | **edata {** |  |
| eks:{ | status, | **eks {** | For OE\_INTERACT, use following structure |
|  | description |  | for ‘eks’ |
| } | } | **}** | eks: { |
| }, | ] | **} :** EkStep event data | type, |
| . |  | structure for the Event | id |
| . |  | **status**: ‘failed’ if the | } |
| . |  | operation fails. | **type**: type of interaction (e.g TOUCH, |
| { |  | ‘success’ if the | DRAG, DROP, PINCH, ZOOM, SHAKE, |
| objid, |  | operation succeeds | ROTATE, SPEAK, LISTEN, WRITE, DRAW, |
| access\_token, |  | **description**: reason for | START, END, CHOOSE, ACTIVATE, SHOW, |
| id\_game\_play, |  | the failure (if ‘status’ is | HIDE, SCROLL, HEARTBEAT, OTHER |
| id\_question, |  | ‘failed’), success | **id**: resource id (e.g DEVICE\_BAK\_BUTTON) |
| ekstep\_eventid, |  | message otherwise |  |
| date\_time\_event, |  |  |  |
| edata |  |  |  |
| eks:{ |  |  |  |
| } |  |  |  |
| } |  |  |  |
| ] |  |  |  |

**Common REST APIs:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **API** | **JSON**  **Payload** | **JSON**  **Response** | **Parameter**  **description** | **Comments** |
| login | {  name, deviceid  } | {  status, description  } | **name**: child’s (avatar) name  **deviceid**: ID of the device on which the account was registered  **status**: ‘failed’ if the operation fails. ‘success’ if the operation succeeds **description**: reason for the failure (if ‘status’ is ‘failed’).access\_token (13 digit alpha numeric) if status is ‘success’ | The ‘login’ API may be invoked to authenticate a Child from a device. ‘login’ is not required if the device already has access\_token for the Child (identified by the name&deviceid).  The ‘access\_token’ returned by ‘login’ API may be saved locally on the device. This access\_token need to be passed to APIs that need authentication. An access\_token is unique for the combination of child name and device id). |
| register | {  name, deviceid, | {  status, description | **name**: child’s (avatar) name  **deviceid**: ID of the | The ‘register’ APImay be invoked to create a new account for a Child (uniquely identified by the name  and phone number). |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | grade, schooltype, geo, language, organization, avatarpic  } | } | device  **grade:** gradeof the child  **schooltype:** 0-govt, 1- pvt  **geo:** location (“long, latt”)  **language:** language (eg. KANNADA, ENGLISH, HINDI,…)  **organization**: Name of the Organization Child belongs to  **avatarpic**: avatar picture image as base64 encoded string **status**: ‘failed’ if the operation fails. ‘success’ if the operation succeeds **description**: reason for the failure (if ‘status’ is ‘failed’).access\_token (13 digit alpha numeric) if status is ‘success’ | The API will return access\_token on successful completion of the registration (Child will be considered as logged-in after registration and an access\_token will be generated)  The input parameters ‘name’, ‘deviceid’, ‘grade’ and ‘language’ are mandatory. Rest are optional.  An account is uniquely defined by the ‘name’ and ‘deviceid’ (device on which the account has been registered) |
| getavatarpic | { | { | **name**: child’s (avatar) | The ‘getavatarpic’ API may be invoked to pull the |
|  | name, | status, | name | avatar picture of a Child (name/phone number) |
|  | deviceid | description | **deviceid**: ID of the | from the backend |
|  | } | } | device on which the |  |
|  |  |  | registration was done |  |
|  |  |  | **avatarpic**: avatar |  |
|  |  |  | picture image as a |  |
|  |  |  | base64 encoded string |  |
| updateavatarpic | { | { | **avatarpic**: avatar | The ‘updateavatarpic’ API may be invoked to |
|  | name, | status, | picture image as a | upload new avatar picture image, if/when the |
|  | deviceid, | description | base64 encoded string | profile/avatar picture is changed for a Child. The |
|  | avatarpic | } |  | picture image is to be send as a ‘base64 encoded |
|  | } |  |  | string’. |
| uploadpicfile | name, deviceid, avatarpicfile | {  status, description  } | **avatarpicfile:** File uploaded as ‘multipart’ type | This API may be used to upload the picture file as a file (multipart/form-data) instead of sending as base64 encoded string (the app can use either ‘updateavatarpic’ API or ‘uploadpicfile’ to upload the picture to the Server) |
|  |  |  |  | The input parameters ‘name’, ‘deviceid’ and ‘avatarpicfile’ should be sent as ‘multipart/form- data’ by POST method. |
|  |  |  |  | A sample web client invoking this API would look like below: |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | <form enctype="multipart/form-data" action=["http://www.kodvin.com/abs/uploadpicfile](http://www.kodvin.com/abs/uploadpicfile)" method="POST">  <input type="hidden" name="**name**" value="5a5786567091e" />  <input type="hidden" name="**deviceid**" value="abcde" />  Select the file: <input name="**avatarpicfile**" type="file" />  <input type="submit" value="Send File" />  </form>  The response is a JSON object |
| getaccesstoken | {  name, deviceid  } | {  status, description  } | **name**: child’s (avatar) name  **deviceid**: ID of the device on which the account has been  registered | The ‘getaccesstoken’ API may be invoked to receive the ‘access\_token’ for a given Child for a given device (name,phone,deviceid combination) |
| getchild | {  name, deviceid  } | {  childid, name, deviceid, grade, schooltype, geo, language, organization, avatarpic, status, description  } | **childid:** childID (unique ID to identify the Child in the ABS backend)  **grade:** gradeof the child  **schooltype:** 0-govt, 1- pvt  **geo:** location (“long, latt”)  **language:** language (eg. KANNADA, ENGLISH, HINDI,…)  **organization**: Name of the Organization Child belongs to  **avatarpic**: avatar  picture image as base64 encoded string | The ‘getchild’ API may be invoked to pull the details of a given Child (for the name and phone number combination) |
| updateprofile | {  name, deviceid, grade, schooltype, language, organization, avatarpic  } | {  status, description  } | **grade:** gradeof the child  **schooltype:** 0-govt, 1- pvt  **language:** language (eg. KANNADA, ENGLISH, HINDI,…)  **organization**: Name of the Organization Child belongs to  **avatarpic**: avatar picture image as base64 encoded string | Invoke this API to update the profile details of a Child (avatarname/deviceid) |

1. Flow Diagram:

App login Screen

Start game Screen

Registration language selection Screen

Registration avatar selection Screen

Game mode selection Screen

(Practice or challenge)

Practice mode

Challenge mode

Grade selection

Concept selection selection

concept selection

game selection

game selection

1. Facebook campaign:

For for book there is one third party plugin to use facebook sdk in cordova

link-<https://github.com/jeduan/cordova-plugin-facebook4>

We had used this before but we were getting notification from google play that we cannot use third party sdk’s only google play recommended sdk should be added as our target audience is below 13.

1. Changing the server url:

To point telemetry to the dev server just change the url variable in

“abbchmprmdsjsapilib-1.0.js” file, appLoginScreen,js, editLangScreen.js, registrationPicSelectionScreen.js file.

1. Changing url for download file:

You can change the url for assets download in the “appLoginEditScreen.js” file.

Current url: “<https://abbmath.klp.org.in/abbchmprm/assets/bb5_0_5/>”

Change the url to any specified url to change the download path/server.

1. Steps for creating diksha content:

I have shared a sample file in the mail, also shared a sheet containing game mapping with microconcept and grade.

* 1. Get a sample diksha content file(Editing the sample file is easy and efficient).
  2. Replace Assets for particular content from assets folder(folder will have the same content name).
  3. Replace the json file from the json folder.
  4. Replace the question sounds from questionsounds folder.
  5. Replace the content js file from js folder
  6. Edit the js file path in index.html file.
  7. Upload and test the content on the diksha platform.