

**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**APLICACIONES MÓVILES Y SERVICIOS TELEMÁTICOS**

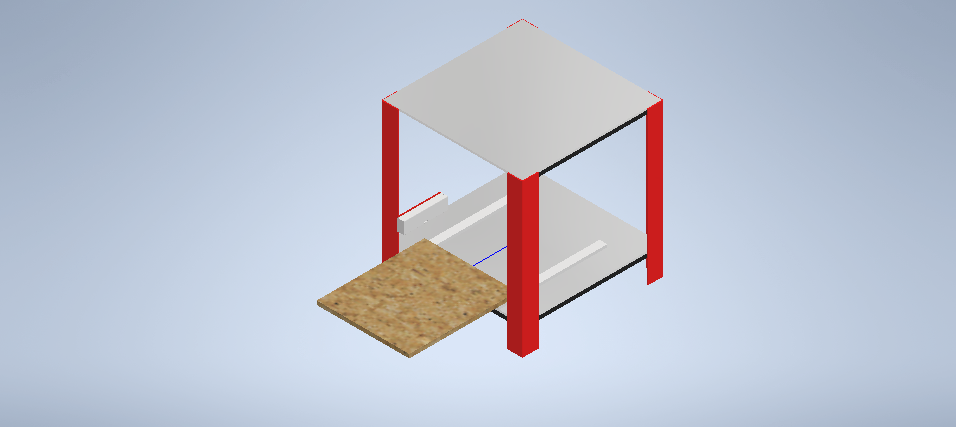
**ESPECIFICACIONES TÉNICAS**

**MÁQUINA CONTROLADORA DE PRESTAMOS DE EQUIPOS TELEMÁTICOS**



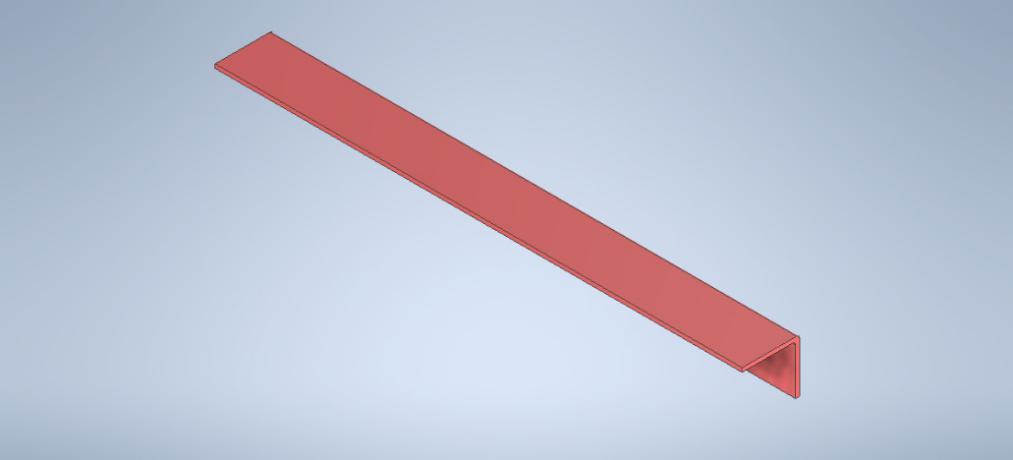
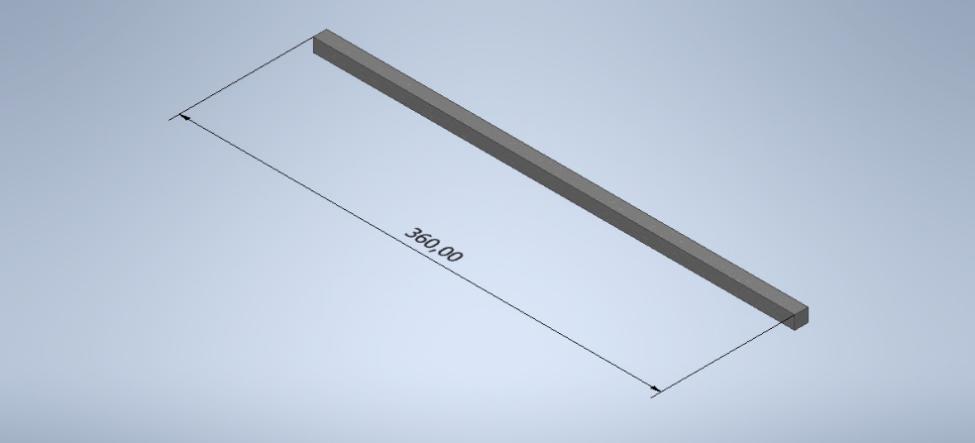
**1.- FUNCIONALIDADES**

La máquina consta de un tablero móvil que se deslizara de manera paralela a la base a una distancia aproximada de 28 cm en la cual accionara un sensor (ibutton red) permitiendo que diagnostique el estado del equipo telemático (disponible o ausente), la superficie es desmontable para poder utilizar equipos que supere las dimensiones de la maquina (altura).

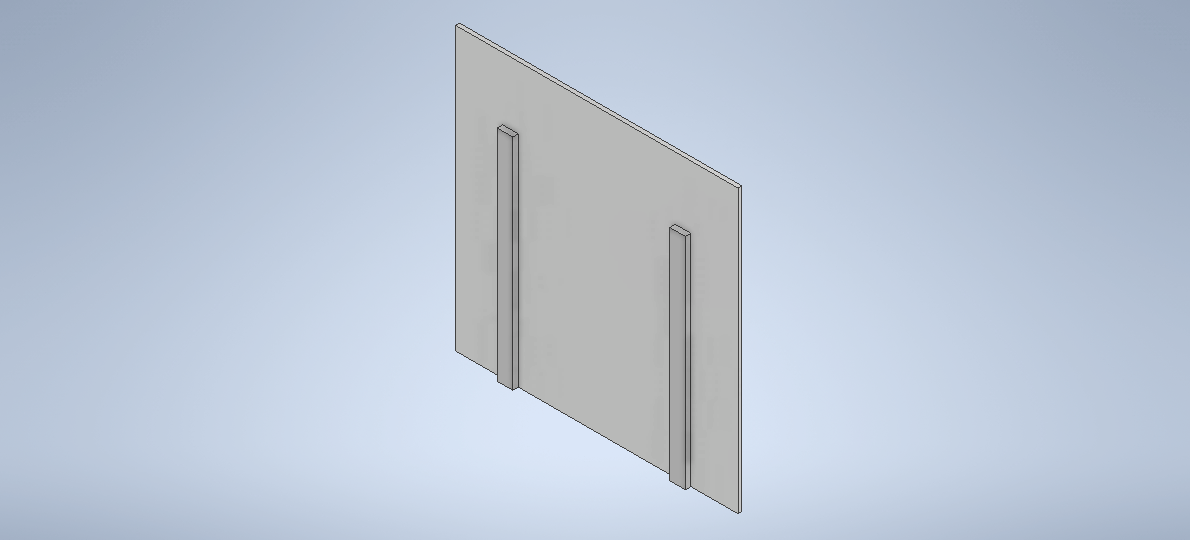


**2.- DESCRICIÓN DE LA ESTRUCTURA**

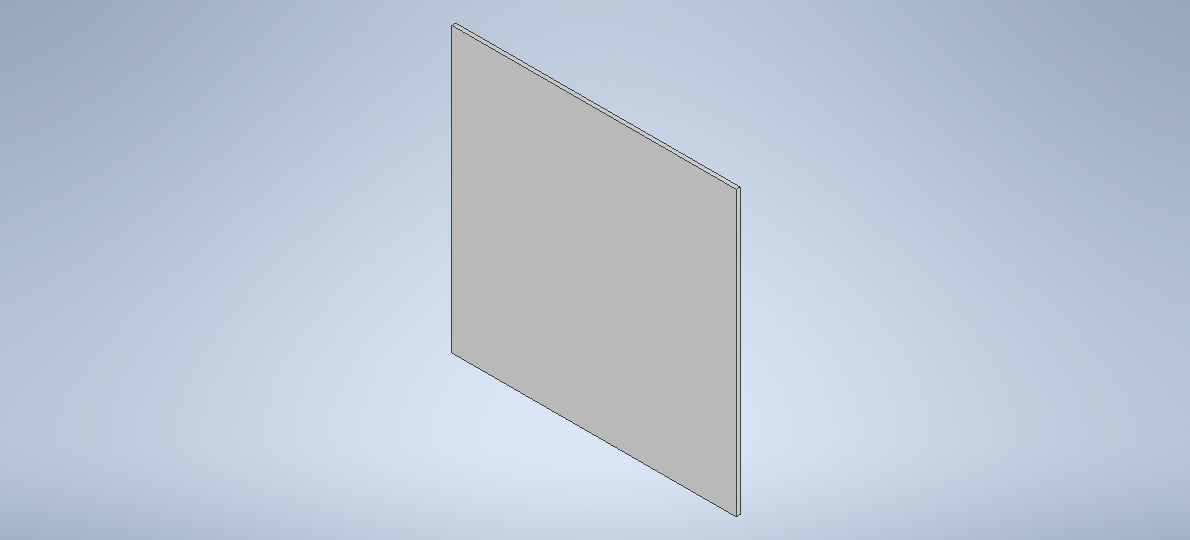
La máquina consta de tres partes fundamentales, la estructura fabricada a partir de ángulos de “Acero Sae 1018” con las siguientes dimensiones 1” x 1” x 3mm de espesor con una longitud de 395 mm por ángulo (total 4 ángulos), consta de barillas cuadradas de “Acero Sae 1018”, con las siguientes dimensiones 10mm x 10mm con una longitud de 360mm de longitud (total 6 barillas cuadradas), todo este conjunto esta soldado con electrodos E6011 con un pensó en conjunto de 4 kg.

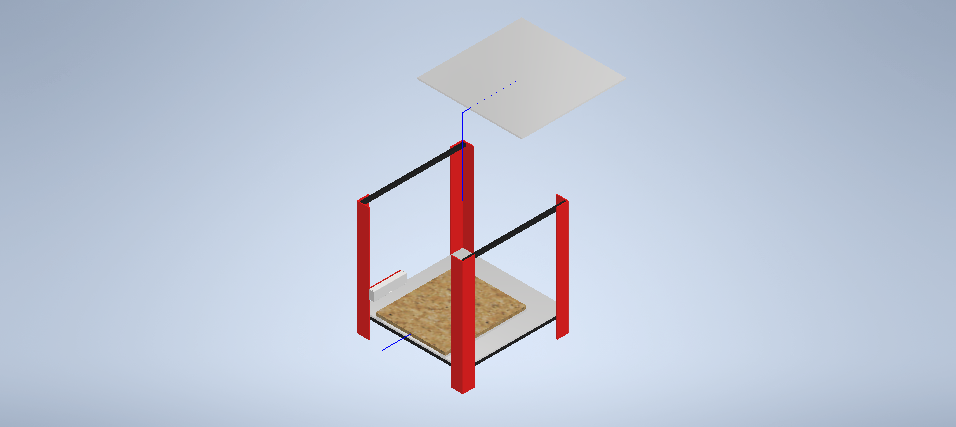
Base, fabricada a partir de planchas de aluminio de 360mm x 360mm y 5mm de espesor con un peso aproximado de 3,7 kg. Consta de un juego de rieles de 28 cm de recorrido, y una plancha de plywood de 280 mm x 250mm x 7mm de espesor, un peso en conjunto de la base es igual a 3,9 kg

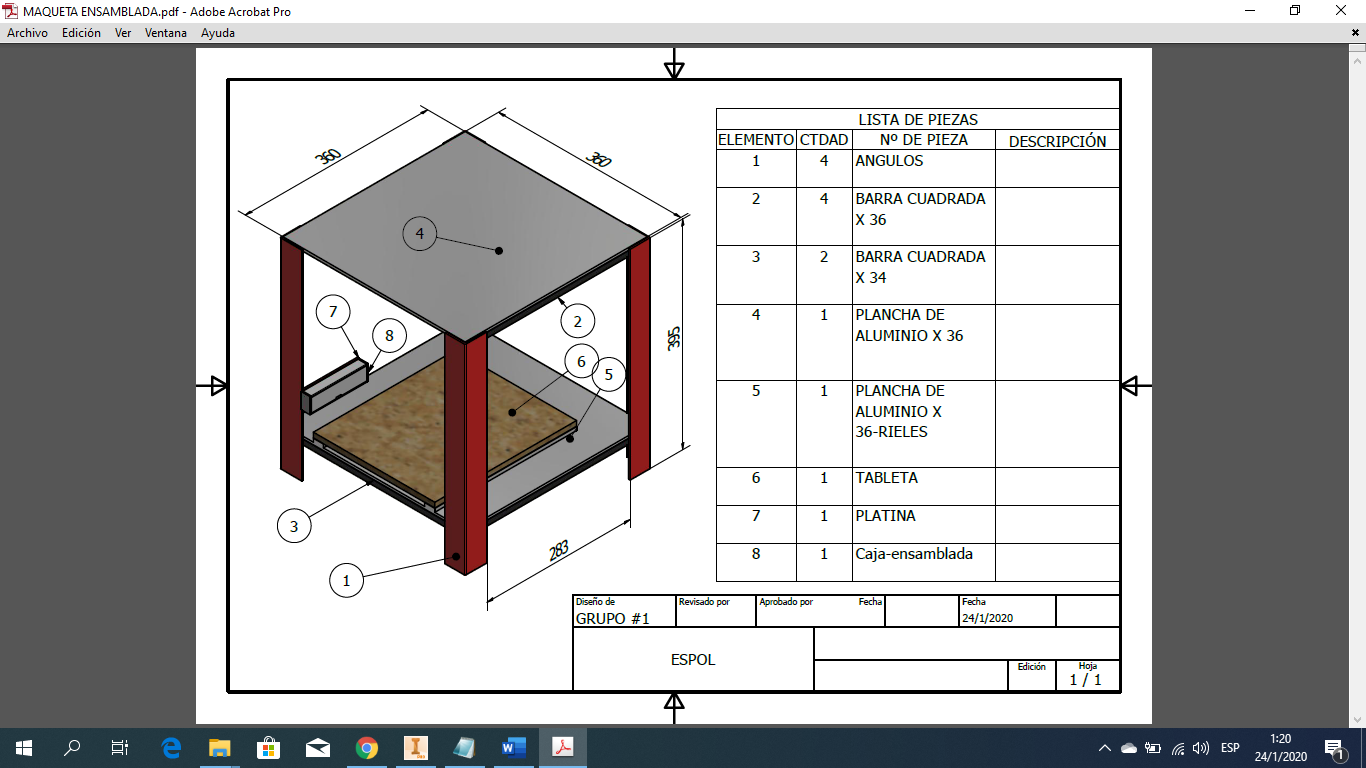


La superficie esta fabricada a partir planchas de aluminio de 360mm x 360mm y 5mm de espesor con un peso aproximado de 3,7 kg



Se realizo un ensamble básico y resistente, con facilidad de desmontado rápido y un peso igual a 11,6 kg (total de peso de la maqueta).

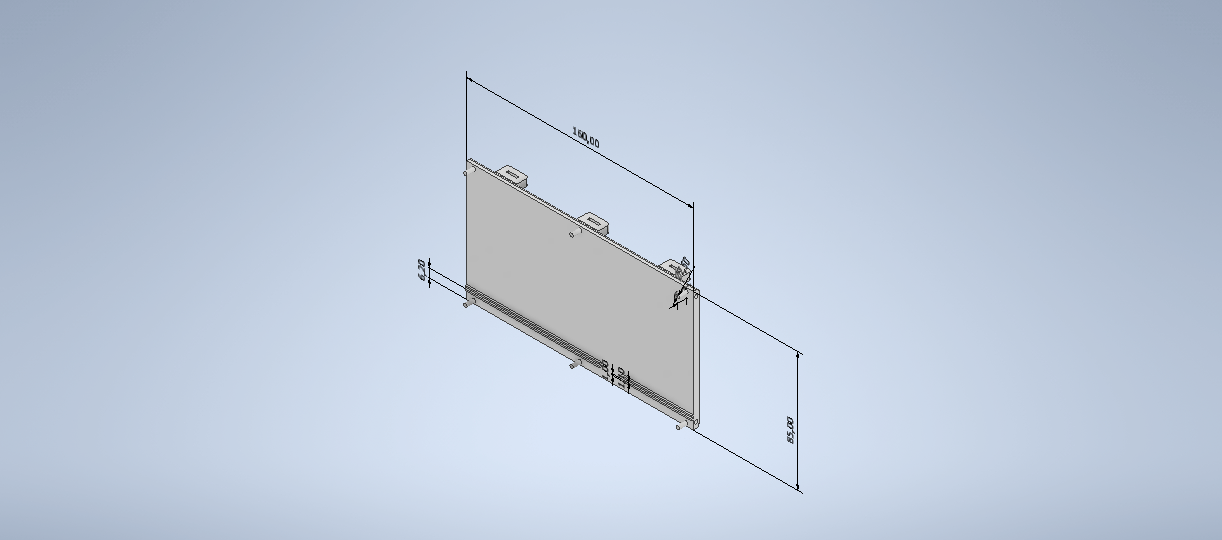
Se detalla las dimensiones generales de la estructura completa:

**3.- DESCRIPCIÓN DE CAJA EMSAMBLADORA**

La caja para el circuito Arduino consta de 6 caras cada cara con distintos componentes y desmontable

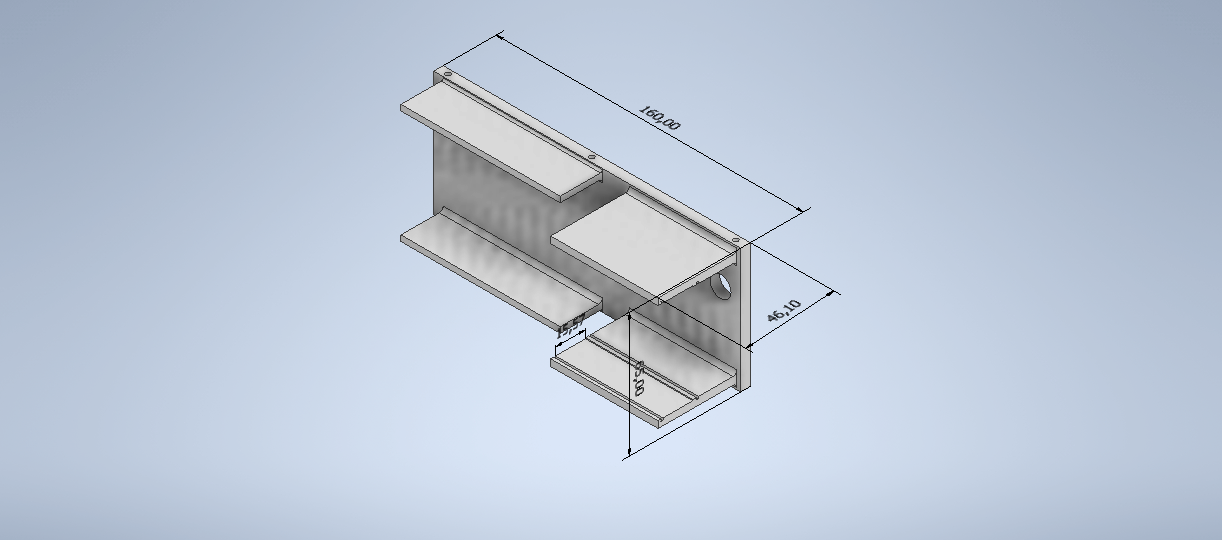
**3.1.- Base inferior**

La base inferior está diseña para se una guía para el ibutton red, sirve de soporte para ensamblar, la caja con la platina que está colocada en la estructura (maqueta) teniendo las siguientes dimensiones: 160mm de largo x 85mm de alto x 4mm de espesor



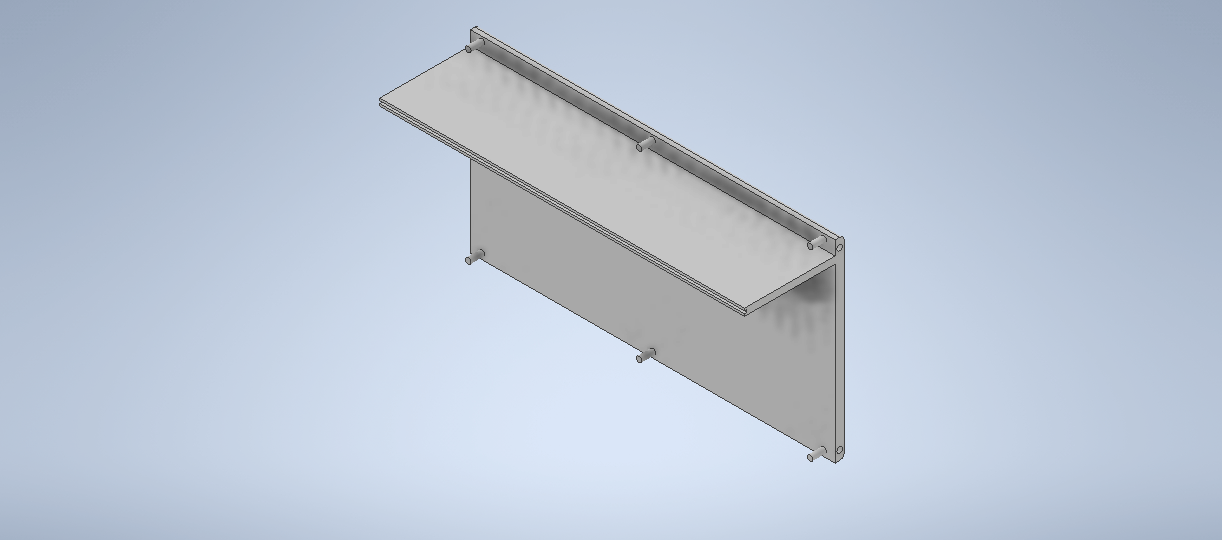
**3.2.- Base sujetadora**

Esta cara contiene la mayor parte de las guías para colocar, el Arduino, dispositivo Sigfox, y protoboard, ibutton red, consta de las siguientes dimensiones: 160 mm de largo x 65 mm de alto x 4mm de espesor



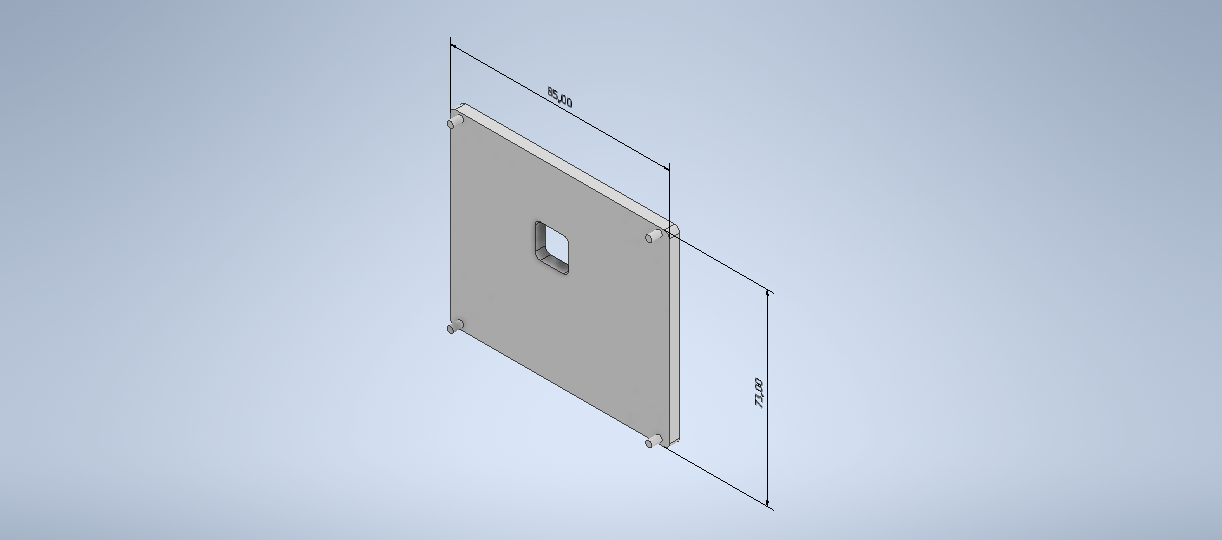
**3.3.- Base superior**

Esta cara es el complemento de la Base inferior que permitirá una guía para el ibutton red, consta de las siguientes dimensiones: 160 mm de largo x 85 mm de ancho x 4mm de espesor



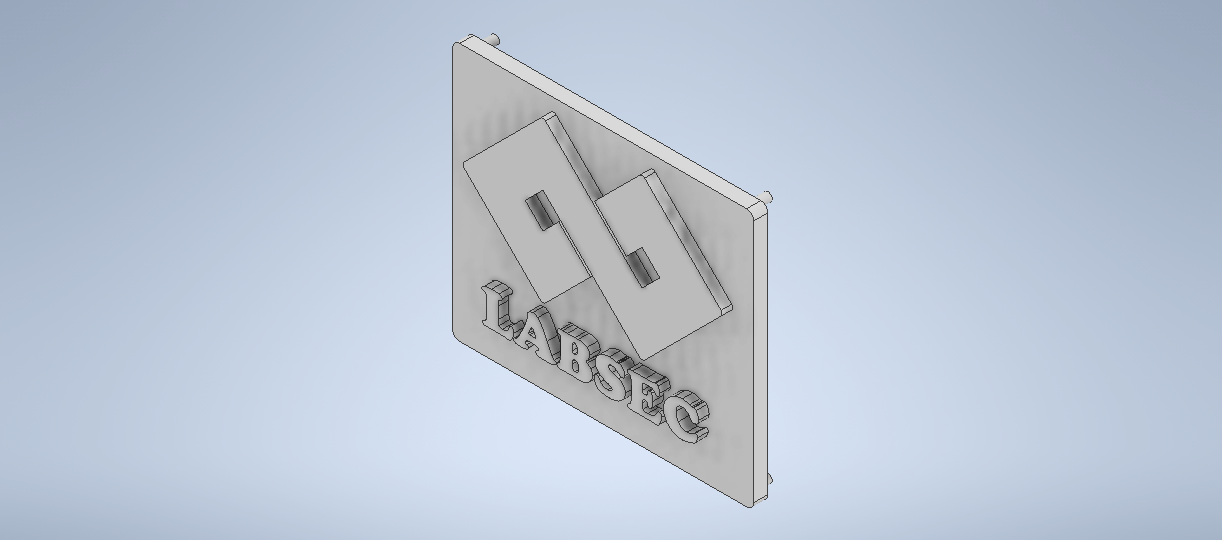
**3.4.- Tapa selladora -2**

Esta cara consta de un agujero cuadrado que permitirá al Arduino uno conectarse a una fuente de energía; constas de las siguientes dimensiones: 85 mm de largo x 73 mm de ancho x 4 mm de espesor



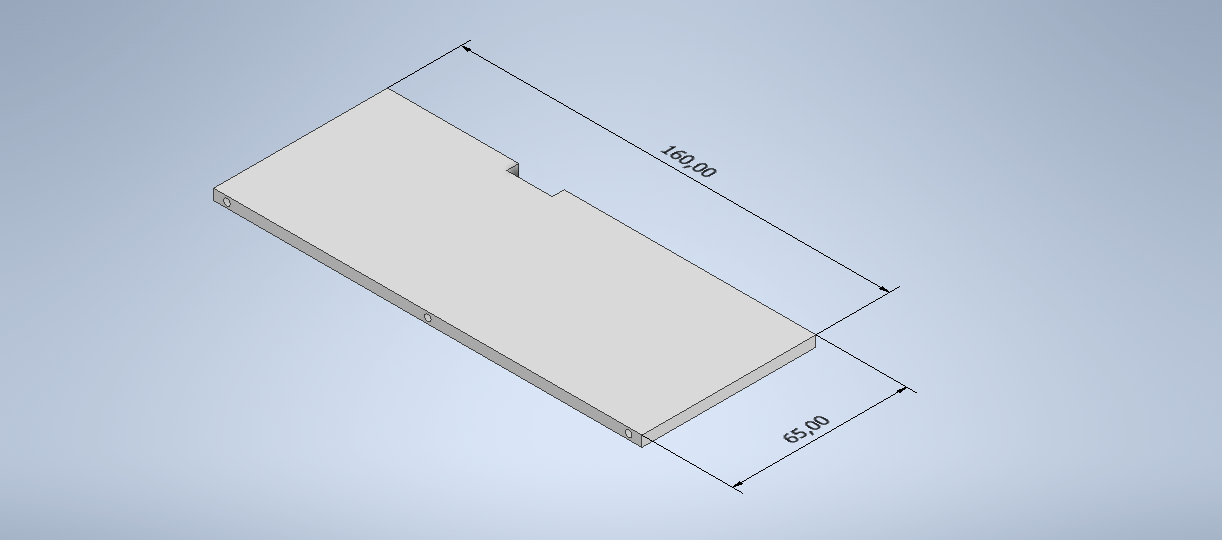
**3.5.- Tapa selladora**

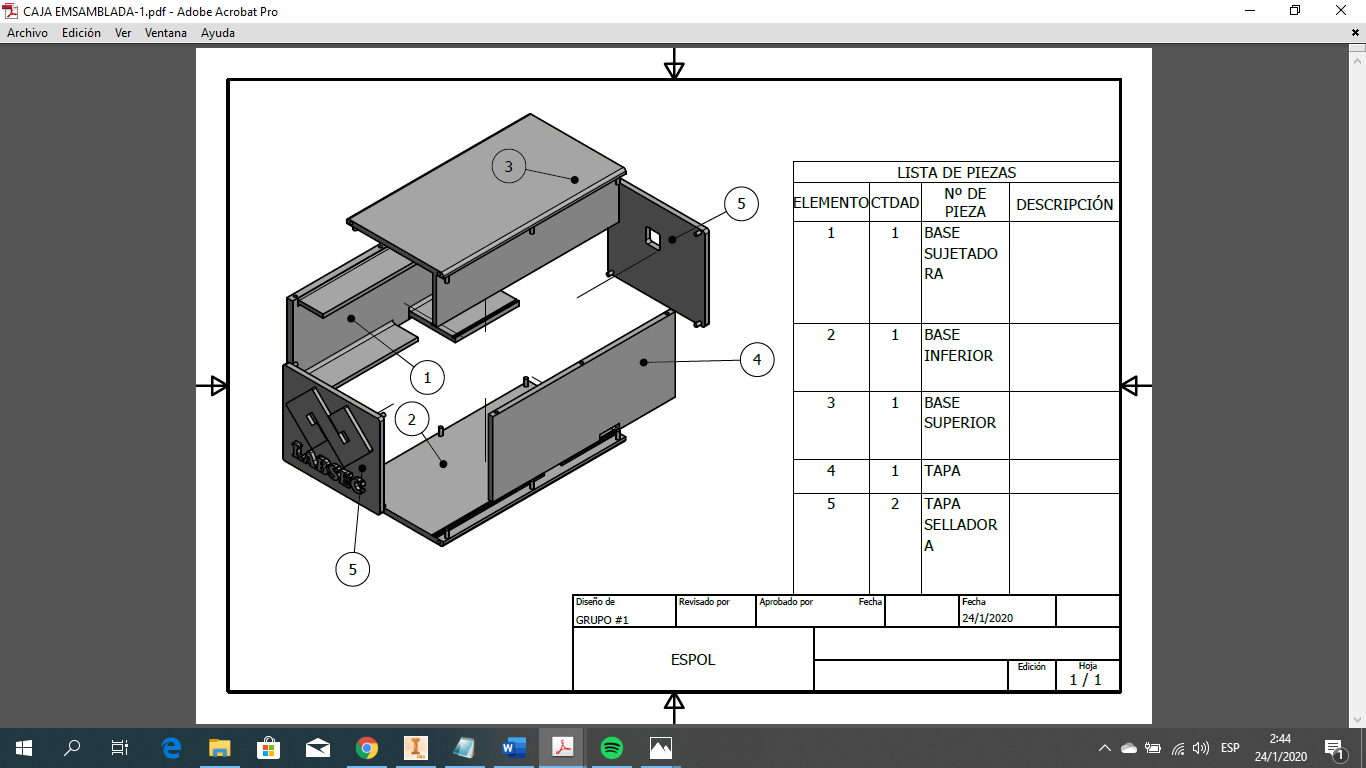
Esta cara es la cara final para sellar la caja completamente una vez ensamblado todo el circuito eléctrico en ella, consta de las siguientes dimensiones: 85mm de largo x 73 mm de ancho.



**3.6. Tapa**

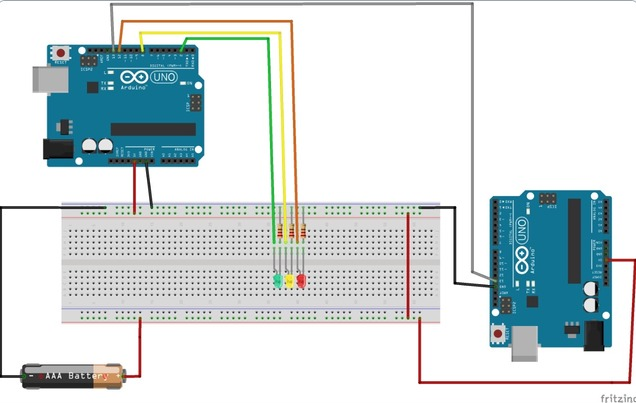
Esta cara contiene un agujero cuadrado que permitirá que el ibutton red no tenga interferencia en el campo magnético al momento de censar algún equipo al momento de salir de la estructura (maqueta), consta de las siguientes dimensiones: 160mm de largo x 65mm de ancho x 4 mm de espesor



Se detalla una estructura general de la caja:

**4.- DETALLE DEL CIRCUITO CON ARDUINO 1**

En este diagrama se demuestra un circuito que nos permitirá visualizar por medio de la ampliación el porcentaje de batería del dispositivo IOT (Ibutton Red).



# Software

### Recursos utilizados

* IDE Android Studio 3.5.2
* Firebase plan Spark
  + Realtime data base
  + Cloud functions
  + Cloud messages

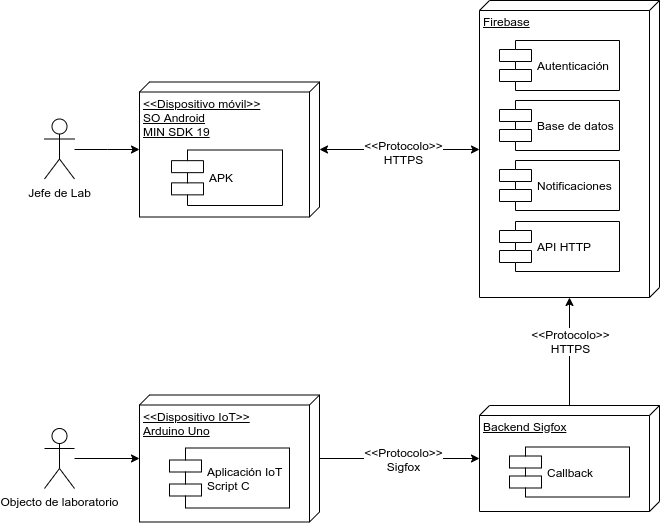
### Presupuesto

Android studio no tiene costo

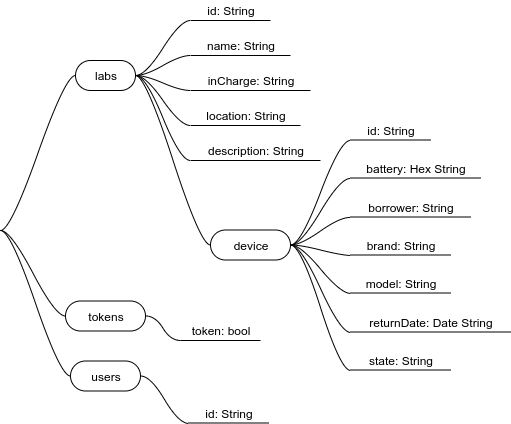
El plan spark de Firebase no tiene costo, sin embargo tiene limitaciones en cuanto a la cantidad de peticiones que se pueden realizar de manera diaria, una gran ventaja de los servicios en la nube es su esquema pago por uso.

### Diagramas

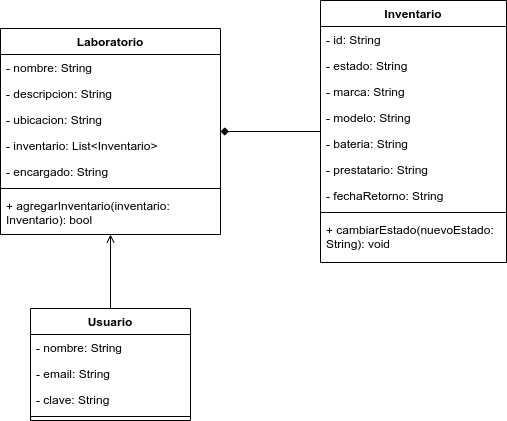
#### Despliegue del sistema



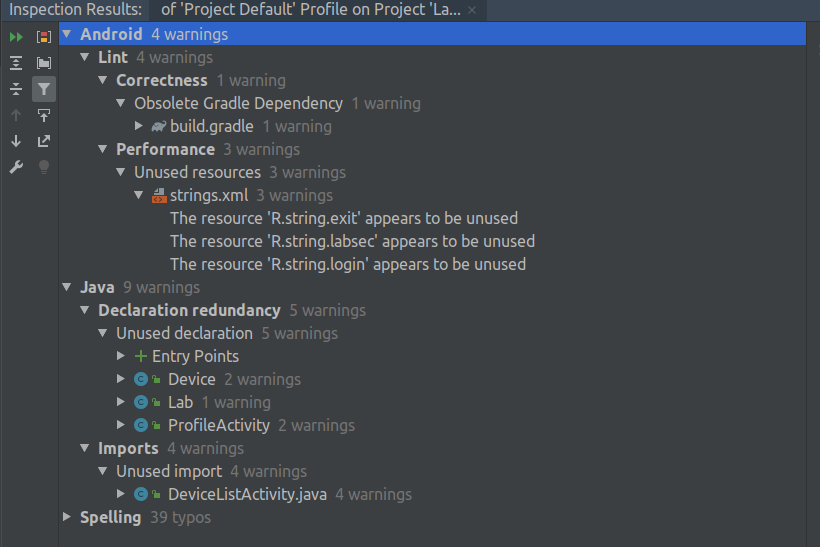
#### Base de datos



#### Clases



### Resultados del analizador de código



### Explicacion del codigo fuente

#### Aplicación móvil

***Splash Activity***

public class SplashActivity extends AppCompatActivity {  
  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.*activity\_splash*);  
  
 getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,  
 WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);  
 Objects.*requireNonNull*(getSupportActionBar()).hide();  
  
 FirebaseAuth mAuth = FirebaseAuth.*getInstance*();  
 FirebaseUser currentUser = mAuth.getCurrentUser();  
 if (currentUser != null) {  
 FirebaseInstanceId.*getInstance*().getInstanceId()  
 .addOnCompleteListener(new OnCompleteListener<InstanceIdResult>() {  
 @Override  
 public void onComplete(@NonNull Task<InstanceIdResult> task) {  
 Intent intent;  
 if (!task.isSuccessful()) {  
 Log.*w*("Instance", "getInstanceId failed",  
 task.getException());  
 Toast.*makeText*(getApplicationContext(),  
 "Error getting current user", Toast.*LENGTH\_SHORT*).show();  
 intent = new Intent(getApplicationContext(),  
 LoginActivity.class);  
 } else {  
 intent = new Intent(getApplicationContext(),  
 LabsListActivity.class);  
 }  
 startActivity(intent);  
 finish();  
 }  
 });  
 } else {  
 Intent intent = new Intent(getApplicationContext(), LoginActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 }  
}

La actividad splash es el punto de entrada de la aplicación, y lo que hace es que en el método oncreate de ciclo de vida de al actividad, una vez que setea el xml a mostrar revisa si hay algún usuario de firebase logueado en la aplicación, si lo hay entonces redirige a la actividad de listado de laboratorios, caso contrario se muestra la actividad de inicio de sesión.

***Login Activity***

@Override  
protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 binding = DataBindingUtil.*setContentView*(this, R.layout.*activity\_login*);  
  
 getWindow().setFlags(WindowManager.LayoutParams.*FLAG\_FULLSCREEN*,  
 WindowManager.LayoutParams.*FLAG\_FULLSCREEN*);  
 Objects.*requireNonNull*(getSupportActionBar()).hide();  
  
 mAuth = FirebaseAuth.*getInstance*();  
 binding.fabLogin.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 login();  
 }  
 });  
 binding.fabGmail.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 googleLogin();  
 }  
 });  
  
 GoogleSignInOptions gso = new GoogleSignInOptions  
 .Builder(GoogleSignInOptions.*DEFAULT\_SIGN\_IN*)  
 .requestIdToken(getString(R.string.*default\_web\_client\_id*)).requestEmail().build();  
 mGoogleSignInClient = GoogleSignIn.*getClient*(this, gso);  
  
 FirebaseUser currentUser = mAuth.getCurrentUser();  
 updateUI(currentUser, false);  
}  
  
private void updateUI(FirebaseUser currentUser, final boolean withRegister) {  
 if (currentUser != null) {  
 FirebaseInstanceId.*getInstance*().getInstanceId()  
 .addOnCompleteListener(new OnCompleteListener<InstanceIdResult>() {  
 @Override  
 public void onComplete(@NonNull Task<InstanceIdResult> task) {  
 if (!task.isSuccessful()) {  
 Log.*w*("Instance", "getInstanceId failed",  
 task.getException());  
 return;  
 }  
 String token = Objects.*requireNonNull*(task.getResult()).getToken();  
 if (withRegister)  
 sendRegistrationToServer(token);  
 Intent intent = new Intent(getApplicationContext(),  
 LabsListActivity.class);  
 startActivity(intent);  
 finish();  
 }  
 });  
 }  
}  
  
private void googleLogin() {  
 Intent signInIntent = mGoogleSignInClient.getSignInIntent();  
 startActivityForResult(signInIntent, *GOOGLE\_SIGN\_IN*);  
}  
  
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
 super.onActivityResult(requestCode,resultCode,data);  
 if (requestCode == *GOOGLE\_SIGN\_IN*) {  
 Task<GoogleSignInAccount> task = GoogleSignIn.*getSignedInAccountFromIntent*(data);  
 try {  
 GoogleSignInAccount account = task.getResult(ApiException.class);  
 if (account != null) firebaseAuthWithGoogle(account);  
 } catch (ApiException e) {  
 if (e.getStatusCode() == 12501)  
 Toast.*makeText*(mContext, "No google account selected",  
 Toast.*LENGTH\_SHORT*).show();  
 else  
 Toast.*makeText*(mContext, e.getMessage(), Toast.*LENGTH\_SHORT*).show();  
 }  
 }  
}  
  
private void firebaseAuthWithGoogle(final GoogleSignInAccount acct) {  
 DatabaseReference database = FirebaseDatabase.*getInstance*().getReference();  
 DatabaseReference ref = database.child("users");  
 ref.addListenerForSingleValueEvent(new ValueEventListener() {  
 @Override  
 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {  
 boolean showError = true;  
 for (DataSnapshot singleSnapshot : dataSnapshot.getChildren()) {  
 if (Objects.*requireNonNull*(singleSnapshot.getValue()).toString()  
 .equals(acct.getEmail())) {  
 binding.pbLogin.setVisibility(View.*VISIBLE*);  
 showError = false;  
 AuthCredential credential = GoogleAuthProvider.*getCredential*(  
 acct.getIdToken(), null);  
 mAuth.signInWithCredential(credential).addOnCompleteListener(  
 (Activity) mContext,  
 new OnCompleteListener<AuthResult>() {  
 @Override  
 public void onComplete(@NonNull Task<AuthResult> task) {  
 if (task.isSuccessful()) {  
 FirebaseUser user = mAuth.getCurrentUser();  
 updateUI(user, true);  
 } else {  
 System.*out*.println("error");  
 }  
 }  
 });  
 }  
 }  
 if (showError) {  
 Toast.*makeText*(mContext, "User has no access", Toast.*LENGTH\_SHORT*).show();  
 mGoogleSignInClient.signOut();  
 }  
 }  
 @Override  
 public void onCancelled(@NonNull DatabaseError databaseError) {  
 Toast.*makeText*(mContext, databaseError.getMessage(),Toast.*LENGTH\_SHORT*).show();  
 }  
 });  
}  
  
private void login() {  
 String mail = Objects.*requireNonNull*(binding.etLoginEmail.getText()).toString();  
 String password = Objects.*requireNonNull*(binding.etLoginPassword.getText()).toString();  
  
 if (mail.isEmpty()) {  
 binding.etLoginEmail.setError("This field is required");  
 }  
  
 if (password.isEmpty()) {  
 binding.etLoginPassword.setError("This field is required");  
 }  
  
 if (mail.isEmpty() || password.isEmpty()) {  
 Toast.*makeText*(getApplicationContext(), "Please enter your crendentials",  
 Toast.*LENGTH\_SHORT*).show();  
 return;  
 }  
  
 binding.fabLogin.setEnabled(false);  
 binding.pbLogin.setVisibility(View.*VISIBLE*);  
 mAuth.signInWithEmailAndPassword(mail, password)  
 .addOnSuccessListener(this, new OnSuccessListener<AuthResult>() {  
 @Override  
 public void onSuccess(AuthResult authResult) {  
 FirebaseUser user = mAuth.getCurrentUser();  
 updateUI(user, true);  
 binding.fabLogin.setEnabled(true);  
 binding.pbLogin.setVisibility(View.*GONE*);  
 }  
 })  
 .addOnFailureListener(this, new OnFailureListener() {  
 @Override  
 public void onFailure(@NonNull Exception e) {  
 if (Objects.*equals*(e.getMessage(), "An internal error has occurred. [ 7: ]")) {  
 Toast.*makeText*(getApplicationContext(), "No internet connection",  
 Toast.*LENGTH\_SHORT*).show();  
 } else {  
 Toast.*makeText*(getApplicationContext(), "Please check your credentials",  
 Toast.*LENGTH\_SHORT*).show();  
 updateUI(null, false);  
 }  
 binding.fabLogin.setEnabled(true);  
 binding.pbLogin.setVisibility(View.*GONE*);  
 }  
 })  
 .addOnCanceledListener(this, new OnCanceledListener() {  
 @Override  
 public void onCanceled() {  
 Log.*v*("CANCELED", "CANCELED");  
 }  
 });  
}  
  
private void sendRegistrationToServer(String token) {  
 FirebaseDatabase.*getInstance*().getReference().child("tokens").child(token).setValue(true);  
 Toast.*makeText*(getApplicationContext(), "token registered",  
 Toast.*LENGTH\_SHORT*).show();  
}

#### Cloud function