NSE+BSE Corporate Action Alerts — Kivy Android App (Kivy + Buildozer in Colab)

> Foreground service with persistent notification. Scrapes NSE & BSE. Filters corporate actions. Emails only new items. Manual "Check Updates" button. Buildable on Google Colab with Buildozer.

Repo Layout

```
📦 nse_bse_alerts
├── main.py
├── ui.kv
├── buildozer.spec
├── requirements.txt
├── config.example.toml
├── service/
│   └── main.py
├── app/
│   ├── __init__.py
│   ├── config.py
│   ├── constants.py
│   ├── util.py
│   ├── store.py
│   ├── scraper.py
│   ├── filtering.py
│   ├── emailer.py
│   └── runner.py
└── README.md
```

`main.py`

```
# file: main.py
from __future__ import annotations
import os
import sys
import threading
import time
from datetime import datetime
```

```python
from kivy.app import App
from kivy.clock import Clock
from kivy.lang import Builder
from kivy.properties import StringProperty
from kivy.uix.boxlayout import BoxLayout

from app.runner import check_and_alert, ensure_app_dirs
from app.util import in_android, ensure_runtime_permissions, is_ist_now

KV = None

class Root(BoxLayout):
    status = StringProperty("Idle")
    last_checked = StringProperty("—")

    def trigger_check(self):
        # Why thread: avoid blocking UI with network/IO
        self.status = "Checking…"
        threading.Thread(target=self._do_check, daemon=True).start()

    def _do_check(self):
        try:
            new_count = check_and_alert(manual=True)
            self.status = f"Done. New alerts: {new_count}"
        except Exception as e:  # surface unexpected errors into UI
            self.status = f"Error: {e}"
        finally:
            self.last_checked = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

class AlertsApp(App):
    def build(self):
        global KV
        if KV is None:
            KV = Builder.load_file("ui.kv")
        ensure_app_dirs()
        Clock.schedule_once(lambda *_: self._bootstrap_service(), 0)
        return Root()

    def _bootstrap_service(self):
        # Start foreground service + request permissions on Android
        if in_android():
            try:
                ensure_runtime_permissions()
                from android import AndroidService  # type: ignore
                service = AndroidService(
                    "NSE+BSE Corporate Alerts",
                    "Monitoring corporate actions…",
                )
                service.start("service-start")
            except Exception as e:
                print("[WARN] Could not start service:", e)

if __name__ == "__main__":
    AlertsApp().run()
```

```
# file: ui.kv
#:kivy 2.2.0
<Root>:
    orientation: 'vertical'
    padding: dp(16)
    spacing: dp(12)
    Label:
        text: "NSE + BSE Corporate Action Alerts"
        font_size: '20sp'
        bold: True
        size_hint_y: None
        height: self.texture_size[1] + dp(12)
    Label:
        text: "Status: " + root.status
        size_hint_y: None
        height: self.texture_size[1] + dp(8)
    Label:
        text: "Last checked: " + root.last_checked
        size_hint_y: None
        height: self.texture_size[1] + dp(8)
    Button:
        text: 'Check Updates'
        size_hint_y: None
        height: dp(48)
        on_release: root.trigger_check()
    Widget:
```

Foreground Service — `service/main.py`

```
# file: service/main.py
from __future__ import annotations
import os
import sys
import time
import traceback
```

```
from datetime import datetime, timedelta, time as dtime

# Ensure we can import app/*
SERVICE_DIR = os.path.dirname(os.path.abspath(__file__))
ROOT_DIR = os.path.abspath(os.path.join(SERVICE_DIR, os.pardir))
if ROOT_DIR not in sys.path:
    sys.path.insert(0, ROOT_DIR)

from app.runner import check_and_alert, ensure_app_dirs
from app.util import get_ist_now, in_android

# --- Android notification plumbing via pyjnius ---
try:
    from jnius import autoclass, cast
except Exception:  # running off-device
    autoclass = cast = None

NOTIF_CHANNEL_ID = "nsebse_alerts_channel"
NOTIF_ID = 101


def _android_context():
    if not in_android():
        return None
    PythonService = autoclass('org.kivy.android.PythonService')
    return PythonService.mService


def _create_channel_if_needed(ctx):
    # For Android O+; safe to call repeatedly
    try:
        Build = autoclass('android.os.Build')
        if Build.VERSION.SDK_INT < 26:
            return
        NotificationChannel = autoclass('android.app.NotificationChannel')
        NotificationManager = autoclass('android.app.NotificationManager')
        Importance = NotificationManager.IMPORTANCE_LOW
        channel = NotificationChannel(NOTIF_CHANNEL_ID, 'Corporate Alerts',
Importance)
        manager = cast('android.app.NotificationManager',
ctx.getSystemService(ctx.NOTIFICATION_SERVICE))
        manager.createNotificationChannel(channel)
    except Exception:
        traceback.print_exc()


def _small_icon_id(ctx):
    try:
```

```python
        # Use app mipmap/icon
        res = ctx.getResources()
        pkg = ctx.getPackageName()
        icon_id = res.getIdentifier('icon', 'mipmap', pkg)
        if icon_id == 0:
            icon_id = android.R.drawable.stat_notify_sync_noanim  # type: ignore
        return icon_id
    except Exception:
        return 0


def _build_notification(ctx, text):
    try:
        _create_channel_if_needed(ctx)
        Builder = autoclass('androidx.core.app.NotificationCompat$Builder')
        PendingIntent = autoclass('android.app.PendingIntent')
        Intent = autoclass('android.content.Intent')
        # Open app when tapping the notification
        intent = Intent(ctx, autoclass('org.kivy.android.PythonActivity'))
        flags = PendingIntent.FLAG_UPDATE_CURRENT
        pending = PendingIntent.getActivity(ctx, 0, intent, flags)

        builder = Builder(ctx, NOTIF_CHANNEL_ID)
        builder.setContentTitle('NSE+BSE Corporate Alerts')
        builder.setContentText(text)
        builder.setSmallIcon(_small_icon_id(ctx))
        builder.setOngoing(True)
        builder.setContentIntent(pending)
        return builder.build()
    except Exception:
        traceback.print_exc()
        return None


def _start_foreground(text="Monitoring…"):
    ctx = _android_context()
    if not ctx:
        return
    notif = _build_notification(ctx, text)
    if notif is not None:
        ctx.startForeground(NOTIF_ID, notif)


def _update_notification(text: str):
    ctx = _android_context()
    if not ctx:
        return
    notif = _build_notification(ctx, text)
    if notif is None:
        return
    NotificationManagerCompat =
autoclass('androidx.core.app.NotificationManagerCompat')
    nm = NotificationManagerCompat.from(ctx)
    nm.notify(NOTIF_ID, notif)


# --- Scheduling policy ---
MARKET_START = dtime(9, 15)
MARKET_END = dtime(15, 30)


def _within_market_hours(now):
    if now.weekday() >= 5:  # Sat/Sun
        return False
    t = now.time()
    return (t >= MARKET_START) and (t <= MARKET_END)


def service_main():
    ensure_app_dirs()
    _start_foreground("Starting…")
    last_daily = None  # last date (IST) when 07:00 run executed

    while True:
        try:
            now = get_ist_now()
            # 07:00 daily run
            seven = dtime(7, 0)
            if (last_daily != now.date()) and (now.time() >= seven):
                n = check_and_alert(trigger="07:00 daily")
                last_daily = now.date()
                _update_notification(f"Daily run: {n} new at {now.strftime('%H:
%M')}")

            # Market-hours every 10 minutes
            if _within_market_hours(now) and (now.minute % 10 == 0):
                n = check_and_alert(trigger="10m market window")
                _update_notification(f"Last: {n} new at {now.strftime('%H:
%M')}")

        except Exception:
            traceback.print_exc()
        finally:
            time.sleep(60)  # tick once per minute
```

```
if __name__ == '__main__':
    service_main()
```

App Core Modules — `app/`

```
# file: app/__init__.py
__all__ = []
```

```
# file: app/constants.py
CA_KEYWORDS = [
    "dividend",
    "bonus",
    "split",
    "rights",
    "buyback",
    "merger",
    "demerger",
]
# Greedy match patterns (lowercased matching)
```

```
# file: app/util.py
from __future__ import annotations
import os
import re
import socket
import sys
from datetime import datetime, timezone, timedelta

try:
    import pytz  # packaged for reliable IST tz
    IST = pytz.timezone('Asia/Kolkata')
except Exception:
    IST = timezone(timedelta(hours=5, minutes=30))


def get_ist_now() -> datetime:
    try:
        return datetime.now(IST)
    except Exception:
        return datetime.now(timezone(timedelta(hours=5, minutes=30)))
```

```
def is_ist_now(hour: int, minute: int = 0) -> bool:
    now = get_ist_now()
    return now.hour == hour and now.minute == minute


def in_android() -> bool:
    return 'ANDROID_ARGUMENT' in os.environ


def ensure_runtime_permissions():
    # Request POST_NOTIFICATIONS (API 33+) and ignore-battery-optimizations
    if not in_android():
        return
    try:
        from jnius import autoclass, cast
        from android import mActivity  # type: ignore
        Build = autoclass('android.os.Build')
        if Build.VERSION.SDK_INT >= 33:
            ActivityCompat = autoclass('androidx.core.app.ActivityCompat')
            Manifest = autoclass('android.Manifest')
            perms = [Manifest.permission.POST_NOTIFICATIONS]
            ActivityCompat.requestPermissions(mActivity, perms, 1001)
        # Ask user to whitelist from battery optimizations
        PowerManager = autoclass('android.os.PowerManager')
        Context = autoclass('android.content.Context')
        Intent = autoclass('android.content.Intent')
        Settings = autoclass('android.provider.Settings')
        Uri = autoclass('android.net.Uri')
        pkg = mActivity.getPackageName()
        pm = cast('android.os.PowerManager',
mActivity.getSystemService(Context.POWER_SERVICE))
        if not pm.isIgnoringBatteryOptimizations(pkg):
            intent =
Intent(Settings.ACTION_REQUEST_IGNORE_BATTERY_OPTIMIZATIONS)
            intent.setData(Uri.parse(f"package:{pkg}"))
            mActivity.startActivity(intent)
    except Exception as e:
        print("[perm] warn:", e)


def app_storage_dir() -> str:
    # Kivy App.user_data_dir when available; fallback to private dir
    path = None
    try:
        from kivy.app import App
        app = App.get_running_app()
        if app:
            path = app.user_data_dir
    except Exception:
        pass
    if not path:
        path = os.path.join(os.getcwd(), ".appdata")
    os.makedirs(path, exist_ok=True)
    return path
```

```
# file: app/config.py
from __future__ import annotations
import os
import tomllib
from dataclasses import dataclass
from typing import List
from .util import app_storage_dir

CONFIG_PATH = os.path.join(app_storage_dir(), "config.toml")

@dataclass
class MailConfig:
    smtp_host: str
    smtp_port: int
    username: str
    password: str
    from_addr: str
    to_addrs: List[str]

@dataclass
class AppConfig:
    mail: MailConfig


def load_config() -> AppConfig:
    # ENV overrides for CI/testing
    env = os.environ
    if os.path.exists(CONFIG_PATH):
        with open(CONFIG_PATH, 'rb') as f:
            data = tomllib.load(f)
    else:
        data = {
            'mail': {
                'smtp_host': env.get('SMTP_HOST', 'smtp.gmail.com'),
                'smtp_port': int(env.get('SMTP_PORT', '587')),
                'username': env.get('SMTP_USER', ''),
                'password': env.get('SMTP_PASS', ''),
                'from_addr': env.get('MAIL_FROM', env.get('SMTP_USER', '')),
                'to_addrs': [x.strip() for x in env.get('MAIL_TO',
'').split(',') if x.strip()],
            }
        }
    mail = MailConfig(**data['mail'])
    return AppConfig(mail=mail)
```

```
# file: app/store.py
from __future__ import annotations
import os
import sqlite3
from contextlib import closing
from typing import Iterable, Tuple
from .util import app_storage_dir

DB_PATH = os.path.join(app_storage_dir(), 'seen.db')

SCHEMA = """
CREATE TABLE IF NOT EXISTS seen (
  k TEXT PRIMARY KEY,
  ts INTEGER
);
"""


def _conn():
    os.makedirs(os.path.dirname(DB_PATH), exist_ok=True)
    conn = sqlite3.connect(DB_PATH)
    with closing(conn.cursor()) as c:
        c.execute(SCHEMA)
        conn.commit()
    return conn


def has(key: str) -> bool:
    with closing(_conn()) as conn, closing(conn.cursor()) as c:
        c.execute("SELECT 1 FROM seen WHERE k=?", (key,))
        return c.fetchone() is not None


def add_all(keys: Iterable[Tuple[str, int]]):
    with closing(_conn()) as conn, closing(conn.cursor()) as c:
        c.executemany("INSERT OR IGNORE INTO seen (k, ts) VALUES (?, ?)",
list(keys))
        conn.commit()
```

```python
# file: app/filtering.py
from __future__ import annotations
import re
from typing import Dict, Any
from .constants import CA_KEYWORDS

PAT = re.compile(r"|".join(rf"\b{re.escape(k)}\b" for k in CA_KEYWORDS), re.I)


def is_corporate_action(item: Dict[str, Any]) -> bool:
    text = " ".join(
        str(item.get(k, ""))
        for k in ("headline", "subject", "category", "details")
    )
    return bool(PAT.search(text))
```

```python
# file: app/scraper.py
from __future__ import annotations
import hashlib
import json
import time
from dataclasses import dataclass
from datetime import datetime
from typing import Dict, List

import requests
from bs4 import BeautifulSoup

from .util import get_ist_now

UA = (
    "Mozilla/5.0 (Linux; Android 13; Pixel 7) "
    "AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0 Mobile Safari/537.36"
)

@dataclass
class Item:
    source: str  # NSE or BSE
    symbol: str
    headline: str
    url: str
    when: datetime

    @property
    def key(self) -> str:
        base = f"{self.source}|{self.symbol}|{self.headline}|
{self.when.isoformat()}"
        return hashlib.sha1(base.encode()).hexdigest()


# --- NSE ---

def _nse_session() -> requests.Session:
    s = requests.Session()
    s.headers.update({
        "User-Agent": UA,
        "Accept": "application/json, text/plain, */*",
        "Referer": "https://www.nseindia.com/",
    })
    # warm cookies
    s.get("https://www.nseindia.com", timeout=15)
    return s


def fetch_nse(limit: int = 100) -> List[Item]:
    s = _nse_session()
    url = "https://www.nseindia.com/api/corporate-announcements"
    params = {"index": "equities"}
    r = s.get(url, params=params, timeout=20)
    r.raise_for_status()
    data = r.json()
    rows = data.get('data') or data.get('corporateActions') or []

    out: List[Item] = []
    for row in rows[:limit]:
        symbol = row.get('symbol') or row.get('scrip') or ''
        headline = row.get('headline') or row.get('subject') or ''
        urlp = row.get('pdfUrl') or row.get('attachment') or row.get('moreLink')
or ''
        when_str = row.get('date') or row.get('sm_dt') or
row.get('announcedDate') or ''
        # Try multiple formats
        when = _parse_dt_guess(when_str)
        if when is None:
            when = get_ist_now()
        out.append(Item('NSE', symbol, headline, urlp, when))
    return out


# --- BSE ---

def fetch_bse(limit: int = 100) -> List[Item]:
    s = requests.Session()
    s.headers.update({"User-Agent": UA, "Accept": "application/json"})
    # Endpoint 1: Announcements (broad)
    today = get_ist_now().strftime('%Y%m%d')
    url = (
        "https://api.bseindia.com/BseIndiaAPI/api/AnnGetData/w"
        f"?strCat=-1&strType=C&strFromDate={today}&strToDate={today}
&strSearch=P&scripcode="
    )
    r = s.get(url, timeout=20)
    r.raise_for_status()
    data = r.json()
    if isinstance(data, dict) and 'Table' in data:
        rows = data['Table']
    else:
        rows = data if isinstance(data, list) else []

    out: List[Item] = []
    for row in rows[:limit]:
        symbol = (row.get('SCRIP_CD') or row.get('Scripcode') or row.get('SC')
or '')
        headline = row.get('HEADLINE') or row.get('HEAD_TEXT') or
row.get('Newssub') or ''
        urlp = row.get('ATTACHMENTNAME') or row.get('PdfLink') or row.get('Url')
or ''
        when_str = row.get('NEWS_DT') or row.get('DtTm') or row.get('NEWS_TIME')
or ''
        when = _parse_dt_guess(when_str)
        if when is None:
            when = get_ist_now()
        out.append(Item('BSE', str(symbol), headline, urlp, when))
    return out


# --- helpers ---

def _parse_dt_guess(s: str | None) -> datetime | None:
    if not s:
        return None
    s = s.strip()
    fmts = [
        "%d-%b-%Y %H:%M:%S",
        "%d %b %Y %H:%M",
        "%Y-%m-%d %H:%M:%S",
        "%d %b %Y",
        "%Y-%m-%d",
    ]
    for f in fmts:
        try:
            return datetime.strptime(s, f)
        except Exception:
            pass
    return None
```

```
# file: app/emailer.py
from __future__ import annotations
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from typing import List

from .config import load_config


def send_mail(subject: str, html_body: str):
    cfg = load_config().mail
    if not cfg.username or not cfg.to_addrs:
        raise RuntimeError("Mail not configured: set username/password/
to_addrs")

    msg = MIMEMultipart('alternative')
    msg['Subject'] = subject
    msg['From'] = cfg.from_addr
    msg['To'] = ", ".join(cfg.to_addrs)
    msg.attach(MIMEText(html_body, 'html'))

    with smtplib.SMTP(cfg.smtp_host, cfg.smtp_port, timeout=30) as s:
        s.starttls()
        s.login(cfg.username, cfg.password)
        s.sendmail(cfg.from_addr, cfg.to_addrs, msg.as_string())
```

```
# file: app/runner.py
from __future__ import annotations
import html
import os
import time
from typing import Iterable, List
from datetime import datetime

from . import store
from .filtering import is_corporate_action
from .scraper import fetch_nse, fetch_bse, Item
from .emailer import send_mail
from .util import app_storage_dir, get_ist_now
```

```
LOCK_PATH = os.path.join(app_storage_dir(), 'run.lock')


def ensure_app_dirs():
    os.makedirs(app_storage_dir(), exist_ok=True)


class FileLock:
    # Why: prevent overlapping runs between UI and service
    def __init__(self, path: str, stale_seconds: int = 600):
        self.path = path
        self.stale = stale_seconds

    def __enter__(self):
        try:
            if os.path.exists(self.path):
                age = time.time() - os.path.getmtime(self.path)
                if age < self.stale:
                    raise RuntimeError("another run is in progress")
            with open(self.path, 'w') as f:
                f.write(str(time.time()))
        except Exception:
            raise

    def __exit__(self, exc_type, exc, tb):
        try:
            if os.path.exists(self.path):
                os.remove(self.path)
        except Exception:
            pass


def _render_email(items: List[Item]) -> str:
    rows = []
    for it in items:
        rows.append(
            f"<tr><td>{html.escape(it.source)}</td><td>{html.escape(it.symbol)}
</td>"
            f"<td>{html.escape(it.headline)}</td>"
            f"<td>{it.when.strftime('%Y-%m-%d %H:%M')}</td>"
            f"<td><a href='{html.escape(it.url)}'>link</a></td></tr>"
        )
    table = (
        "<table border=1 cellspacing=0 cellpadding=6>"
        "<tr><th>Src</th><th>Symbol</th><th>Headline</th><th>When</th><th>Doc</
th></tr>"
        + "".join(rows)
        + "</table>"
    )
    return table


def _collect() -> List[Item]:
    items: List[Item] = []
    try:
        items.extend(fetch_nse())
    except Exception as e:
        print("[nse]", e)
    try:
        items.extend(fetch_bse())
    except Exception as e:
        print("[bse]", e)
    return items


def _filter_new(items: List[Item]) -> List[Item]:
    fresh = [it for it in items if is_corporate_action(it.__dict__)]
    unseen = [it for it in fresh if not store.has(it.key)]
    return unseen


def _remember(items: List[Item]):
    now_ts = int(time.time())
    store.add_all((it.key, now_ts) for it in items)


def check_and_alert(trigger: str | None = None, manual: bool = False) -> int:
    with FileLock(LOCK_PATH):
        items = _collect()
        new_items = _filter_new(items)
        if new_items:
            subj = f"Corp Actions ({len(new_items)}) —
{get_ist_now().strftime('%d %b %Y %H:%M')}"
            body = _render_email(new_items)
            if trigger:
                body = f"<p><i>Trigger: {trigger}</i></p>" + body
            send_mail(subj, body)
            _remember(new_items)
        return len(new_items)


if __name__ == "__main__":
    # CLI helper: run once and print count
    cnt = check_and_alert(trigger="manual CLI", manual=True)
    print("New:", cnt)
```

~~~
---

## Buildozer & Packaging

### `buildozer.spec`
```ini
# file: buildozer.spec
[app]
title = NSE+BSE Corporate Alerts
package.name = nsebsealerts
package.domain = com.yourdomain
source.dir = .
version = 0.1.0
orientation = portrait
fullscreen = 0

# Icons/splash (optional)
icon.filename = %(source.dir)s/.buildozer/android/platform/build-armeabi-v7a/
dists/nsebsealerts/src/main/res/mipmap/ic_launcher.png

requirements = python3,kivy==2.2.1,requests,beautifulsoup4,lxml,plyer,pytz
android.api = 33
android.minapi = 24
android.archs = arm64-v8a, armeabi-v7a

# Service entry point (foreground)
services = service:foreground

# Permissions
android.permissions =
INTERNET,FOREGROUND_SERVICE,RECEIVE_BOOT_COMPLETED,WAKE_LOCK,POST_NOTIFICATIONS

# Use androidx for NotificationCompat
android.gradle_dependencies = androidx.core:core:1.12.0

# Keep network working with NSE site
android.add_default_config = android:usesCleartextTraffic="true"

[buildozer]
log_level = 2
warn_on_root = 0
~~~

`requirements.txt`

```
kivy==2.2.1
requests
beautifulsoup4
lxml
plyer
pytz
```

`config.example.toml`

```
# Place a copy as <app-storage>/config.toml on device, or set env vars.
[mail]
smtp_host = "smtp.gmail.com"
smtp_port = 587
username  = "your@gmail.com"      # Use App Password
password  = "xxxx xxxx xxxx xxxx"  # 16-char Gmail App Password
from_addr = "your@gmail.com"
to_addrs  = ["you@example.com"]
```

Google Colab: Build Steps

Start a fresh Colab notebook and run the cells below, in order.

1) Prepare environment

```
%%bash
apt-get -y update && apt-get -y install git zip unzip openjdk-17-jdk libffi-dev
libssl-dev libsqlite3-dev zlib1g-dev
python3 -m pip install --upgrade pip
python3 -m pip install buildozer cython==0.29.33
```

2) Clone your repo (or create)

```
%%bash
# Example: replace with your repository URL
REPO_URL="https://github.com/<you>/nse_bse_alerts.git"
if [ ! -d nse_bse_alerts ]; then
  git clone "$REPO_URL" nse_bse_alerts
fi
cd nse_bse_alerts
```

```
# If starting from scratch in Colab, you can write files here using %%writefile
```

3) Initialize Buildozer (first time only)

```
%%bash
cd nse_bse_alerts
if [ ! -f buildozer.spec ]; then
  buildozer init
fi
# Overwrite buildozer.spec with the one from this repo
```

4) Build APK (debug)

```
%%bash
set -e
cd nse_bse_alerts
export PATH=$PATH:$HOME/.buildozer/android/platform/android-sdk/cmdline-tools/
latest/bin
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
# The first run downloads SDK/NDK; it takes time.
buildozer -v android debug
```

5) Retrieve APK

```
%%bash
cd nse_bse_alerts
APK=$(find . -path "*.apk" -name "*.apk" | tail -n 1)
echo "APK => $APK"
```

Output APK typically at: `.buildozer/android/platform/build-*/dists/nsebsealerts/build/outputs/apk/debug/nsebsealerts-debug.apk`

6) (Optional) Sign & Align for release

```
%%bash
cd nse_bse_alerts
# Create keystore once
keytool -genkey -v -keystore release.keystore -alias nsebse -keyalg RSA -
keysize 2048 -validity 10000
  -storepass password -keypass password -dname "CN=You, OU=IT, O=YourOrg,
L=City, S=State, C=IN"

# Build release
buildozer -v android release
```

GitHub Setup

- Create repo `nse_bse_alerts` with the structure above.
- Commit all files **except** your real `config.toml`. Commit `config.example.toml`.
- Optionally add `.gitignore`:

```
.appdata/
*.keystore
*.apk
.buildozer/
__pycache__/
```

Email Configuration (Gmail)

1. Turn on 2-Step Verification in Google account.
2. Create an **App Password** (type: Mail, device: Other/Custom).
3. On device, place `config.toml` at app storage path:
4. First run the app once; it prints the storage path in logcat.
5. Typical path (Kivy): `/storage/emulated/0/Android/data/com.yourdomain.nsebsealerts/files/` or app-private.
6. Fill values as per `config.example.toml`.

   Alternatively set environment variables `SMTP_HOST/SMTP_PORT/SMTP_USER/SMTP_PASS/MAIL_FROM/MAIL_TO` before launching (useful for desktop testing).

Testing Checklist

- **Desktop dry run** (no APK):

```
python -m venv .venv && . .venv/bin/activate
pip install -r requirements.txt
export SMTP_USER=... SMTP_PASS=... MAIL_TO=you@example.com MAIL_FROM=...
python app/runner.py  # prints number of new alerts and sends mail if any
python main.py        # opens Kivy window, press "Check Updates"
```

- **On-device**:

- Install APK.
- Launch app once → allow notification permission → approve battery optimization exception when prompted.
- Confirm persistent notification “NSE+BSE Corporate Alerts”.
- Tap **Check Updates**: status updates and (if new) an email arrives.

- Leave app; service keeps running. Observe emails at ~07:00 IST and every 10 minutes during market hours.

- **Logcat tips**:

```
adb logcat | grep -i Python
```

Notes & Limits

- Exchange endpoints occasionally change; the scraper handles common fields and may need tweaks if JSON schemas shift.
- Market holidays aren’t accounted for; the 10-minute loop runs on weekdays only.
- Android power saving can still kill services on some OEMs; whitelisting from battery optimization greatly improves persistence.

README.md (quick copy)

```
# NSE+BSE Corporate Action Alerts (Android, Kivy)

Foreground service with persistent notification. Scrapes NSE & BSE; filters
corporate actions (dividend, bonus, split, rights, buyback, merger, demerger);
emails only new items; manual "Check Updates" button.

## Build (Colab)
See sections in this document for exact commands.

## Configure Email
Copy `config.example.toml` to device storage path as `config.toml` and fill SMTP
values (Gmail App Password recommended).

## Run
Install APK → open app → allow notifications and battery optimization exemption
→ done.
```

```
```