LetzGrade — MVP Plan (Week of Aug 19–25, 2025)

Target: **Minimal Viable Product** with core loop: Auth → Select Year+Category → Auto-assign Courses → Dashboard → Add/View Grades → Basic Settings.

✅Scope & Definition of Done (DoD)

- Users can **sign up / log in** (email+password).
- Users pick **Year** → **Category** (e.g., 7 → 7C). The app **creates a study program** with predefined courses & coefficients.
- **Dashboard** shows user's years; tapping a year shows its **courses**.
- **Course screen** lists **grades** and a control to **add a grade** (value, optional weight, note, date auto-set).
- The app displays a **course average** (weighted if weights provided, otherwise simple mean).
- **Firestore Security Rules** ensure users can only access their own data.
- Basic **settings** (edit name/email/password individually) and **account deletion** flow in place (from prior work).
- No crashes, essential error handling & validation present.

☯️MVP Checklist with Acceptance Criteria

- [ ] **1) Firestore Data Model finalized**
- *AC:* Document shapes consistent; supports grades with optional weights; timestamps recorded.
- [ ] **2) Firestore Security Rules deployed**
- *AC:* Authenticated users can only read/write under `users/{uid}`; invalid grade values rejected.
- [ ] **3) Program creation flow** (Year+Category → create program)
- *AC:* Selecting year+category creates `years` and `courses` with coefficients based on template.
- [ ] **4) Dashboard list** (years visible)
- *AC:* Shows all user years; empty state handled.
- [ ] **5) Year detail** (courses visible)
- *AC:* Shows course name + coefficient; average computed from grades (if any).
- [ ] **6) Course detail** (grades list + Add Grade)
- *AC:* Add grade modal validates input; list updates live; weighted avg displays.
- [ ] **7) Error handling**
- *AC:* User-facing toasts/messages for failures; basic retry guidance.
- [ ] **8) Settings review**
- *AC:* Edit name/email/password works with reauth where needed; delete account flow gated by reauth modal.
- [ ] **9) QA scenarios**
- *AC:* Fresh user; returning user; offline moment (graceful); deletion; odd weights; 0/100 boundary.

Tip: check off each box as you complete it; the code below gives you drop-in pieces for grades.

🖤Firestore Data Model (Proposed)

```
users/{uid}
  profile: {
    displayName: string,
    email: string,
    createdAt: timestamp,
    grading: { min: number, max: number } // e.g., {min: 0, max: 100}
  }
  years/{yearId}:
    {
      yearNumber: number,          // e.g., 7
      category: string,            // e.g., "7C"
      hasExams: boolean,
      createdAt: timestamp
    }
    courses/{courseId}:
      {
        name: string,              // e.g., "Mathematics"
        coefficient: number,       // e.g., 3
        createdAt: timestamp
      }
      grades/{gradeId}:
        {
          value: number,           // e.g., 52.5
          weight: number|null,     // default 1 if omitted
          note: string|null,
          date: timestamp          // serverTimestamp
        }
```

- **Computation:** Averages computed client-side for MVP (no Cloud Functions).
- **Weights:** If `weight` missing → treat as `1`.

🚫Firestore Security Rules (MVP)

Save as `firestore.rules` and deploy with `firebase deploy --only firestore:rules`.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Users can only access their own subtree under users/{uid}
    match /users/{uid}/{document=**} {
      allow read, write: if request.auth != null && request.auth.uid == uid
&& validData();
    }
```

```
    function validData() {
      // Basic gate: allow deletes/reads quickly; validate writes when data
present
      return request.method in ['get', 'list', 'delete'] ||
             (request.resource != null &&
validateDocument(request.resource.data));
    }

    function validateDocument(data) {
      // Only enforce basic constraints for grades; other docs pass-through
for MVP
      // If it's a grade doc, ensure numeric value in [0,100] and weight
positive if provided
      return !(data.keys().hasAll(['value']) && data.keys().hasAny(['weight',
'note', 'date']))
             || (
                  (data.value is number && data.value >= 0 && data.value <=
100) &&
                  (!data.keys().hasAny(['weight']) || (data.weight is number
&& data.weight > 0)) &&
                  (!data.keys().hasAny(['note']) || (data.note is string &&
data.note.size() <= 280))
                );
    }
  }
}
```

Adjust the 0–100 bounds if you use a different national grading scale.

Predefined Curriculum Template (Example)

Place this in `src/data/templates.js` and expand as needed.

```
// src/data/templates.js
export const CURRICULUM_BY_YEAR = {
  7: {
    hasExams: false,
    categories: {
      '7C': {
        courses: [
          { name: 'Mathematics', coefficient: 3 },
          { name: 'Physics', coefficient: 2 },
          { name: 'Biology', coefficient: 2 },
          { name: 'Languages', coefficient: 3 },
          { name: 'History', coefficient: 1 },
        ]
      },
      '7G': {
```

```
        courses: [
          { name: 'Mathematics', coefficient: 2 },
          { name: 'Languages', coefficient: 3 },
          { name: 'Geography', coefficient: 1 },
          { name: 'Art', coefficient: 1 },
        ]
      }
    }
  },
  6: {
    hasExams: false,
    categories: {
      '6C': { courses: [ { name: 'Mathematics', coefficient: 3 }, { name:
'Chemistry', coefficient: 2 } ] },
      '6G': { courses: [ { name: 'Mathematics', coefficient: 2 }, { name:
'Languages', coefficient: 3 } ] }
    }
  }
};
```

🔐 Program Creation Utility

```
// src/services/program.js
import { db } from '../firebase';
import { collection, doc, setDoc, addDoc, serverTimestamp } from 'firebase/
firestore';
import { CURRICULUM_BY_YEAR } from '../data/templates';

export async function createStudyProgram({ uid, yearNumber, category }) {
  const yearCfg = CURRICULUM_BY_YEAR[yearNumber];
  if (!yearCfg) throw new Error('Unsupported year');
  const catCfg = yearCfg.categories[category];
  if (!catCfg) throw new Error('Unsupported category');

  const yearsCol = collection(db, 'users', uid, 'years');
  // Use generated id so the same year can exist with different categories if
needed
  const yearRef = doc(yearsCol);
  await setDoc(yearRef, {
    yearNumber,
    category,
    hasExams: !!yearCfg.hasExams,
    createdAt: serverTimestamp(),
  });

  const coursesCol = collection(yearRef, 'courses');
  for (const c of catCfg.courses) {
    const courseRef = doc(coursesCol);
```

```
    await setDoc(courseRef, {
      name: c.name,
      coefficient: Number(c.coefficient) || 1,
      createdAt: serverTimestamp(),
    });
  }

  return yearRef.id;
}
```

---

9 Grade Math Helpers

```
// src/utils/grades.js
export function computeWeightedAverage(grades) {
  if (!grades || grades.length === 0) return null;
  let sumWeights = 0;
  let sum = 0;
  for (const g of grades) {
    const w = typeof g.weight === 'number' && g.weight > 0 ? g.weight : 1;
    sumWeights += w;
    sum += (Number(g.value) || 0) * w;
  }
  if (sumWeights === 0) return null;
  const avg = sum / sumWeights;
  return Number.isFinite(avg) ? avg : null;
}

export function formatAvg(avg) {
  return avg == null ? '—' : avg.toFixed(2);
}
```

✔️Firestore Grade Services

```
// src/services/grades.js
import { db } from '../firebase';
import {
  collection, doc, addDoc, deleteDoc,
  onSnapshot, serverTimestamp, query, orderBy
} from 'firebase/firestore';

export function gradesCollectionRef(uid, yearId, courseId) {
  return collection(db, 'users', uid, 'years', yearId, 'courses', courseId,
'grades');
}
```

```
export async function addGrade({ uid, yearId, courseId, value, weight,
note }) {
  const col = gradesCollectionRef(uid, yearId, courseId);
  const payload = {
    value: Number(value),
    date: serverTimestamp(),
  };
  if (weight != null && weight !== '') payload.weight = Number(weight);
  if (note) payload.note = String(note).slice(0, 280);
  return await addDoc(col, payload);
}

export async function deleteGrade({ uid, yearId, courseId, gradeId }) {
  const ref = doc(db, 'users', uid, 'years', yearId, 'courses', courseId,
'grades', gradeId);
  await deleteDoc(ref);
}

export function listenGrades({ uid, yearId, courseId, onChange }) {
  const q = query(gradesCollectionRef(uid, yearId, courseId),
orderBy('date', 'desc'));
  return onSnapshot(q, (snap) => {
    const items = snap.docs.map(d => ({ id: d.id, ...d.data() }));
    onChange(items);
  });
}
```

有Course Detail Screen (Grades List + Add Grade)

```
// src/screens/CourseDetailScreen.js
import React, { useEffect, useState } from 'react';
import { View, Text, FlatList, Modal, TextInput, Pressable, Alert } from
'react-native';
import { useRoute } from '@react-navigation/native';
import { listenGrades, addGrade, deleteGrade } from '../services/grades';
import { computeWeightedAverage, formatAvg } from '../utils/grades';
import { auth } from '../firebase';

export default function CourseDetailScreen() {
  const route = useRoute();
  const { yearId, courseId, courseName, coefficient } = route.params;
  const uid = auth.currentUser?.uid;

  const [grades, setGrades] = useState([]);
  const [adding, setAdding] = useState(false);
  const [value, setValue] = useState('');
  const [weight, setWeight] = useState('');
  const [note, setNote] = useState('');
```

```
  useEffect(() => {
    if (!uid) return;
    const unsub = listenGrades({ uid, yearId, courseId, onChange:
setGrades });
    return () => unsub && unsub();
  }, [uid, yearId, courseId]);

  const avg = computeWeightedAverage(grades);

  async function onAdd() {
    const v = Number(value);
    if (!Number.isFinite(v) || v < 0 || v > 100) {
      Alert.alert('Invalid grade', 'Enter a number between 0 and 100.');
      return;
    }
    const w = weight === '' ? undefined : Number(weight);
    if (w != null && (!Number.isFinite(w) || w <= 0)) {
      Alert.alert('Invalid weight', 'Weight must be a positive number.');
      return;
    }
    try {
      await addGrade({ uid, yearId, courseId, value: v, weight: w, note });
      setValue(''); setWeight(''); setNote(''); setAdding(false);
    } catch (e) {
      Alert.alert('Failed to add grade', e.message);
    }
  }

  return (
    <View style={{ flex: 1, padding: 16 }}>
      <Text style={{ fontSize: 22, fontWeight: '700' }}>{courseName}</Text>
      <Text style={{ marginTop: 4 }}>Coefficient: {coefficient}</Text>
      <Text style={{ marginTop: 8, fontSize: 16 }}>Average: {formatAvg(avg)}
</Text>

      <Pressable onPress={() => setAdding(true)} style={{ marginTop: 16,
padding: 12, borderRadius: 12, backgroundColor: '#eee' }}>
        <Text style={{ textAlign: 'center', fontWeight: '600' }}>+ Add
Grade</Text>
      </Pressable>

      <FlatList
        style={{ marginTop: 16 }}
        data={grades}
        keyExtractor={(item) => item.id}
        ListEmptyComponent={<Text>No grades yet. Add your first one.</Text>}
        renderItem={({ item }) => (
          <View style={{ padding: 12, borderRadius: 12, backgroundColor:
'#fafafa', marginBottom: 8 }}>
            <Text style={{ fontSize: 16, fontWeight: '600' }}>Grade:
```

```
{item.value}</Text>
            <Text>Weight: {item.weight ?? 1}</Text>
            {item.note ? <Text>Note: {item.note}</Text> : null}
            <Pressable
              onPress={() => deleteGrade({ uid, yearId, courseId, gradeId:
item.id })}
              style={{ marginTop: 8, padding: 8, borderRadius: 10,
backgroundColor: '#f2f2f2' }}
            >
              <Text style={{ textAlign: 'center' }}>Delete</Text>
            </Pressable>
          </View>
        )}
      />

      <Modal visible={adding} animationType="slide" onRequestClose={() =>
setAdding(false)}>
        <View style={{ flex: 1, padding: 16, justifyContent: 'center' }}>
          <Text style={{ fontSize: 20, fontWeight: '700', marginBottom: 12 }}
>Add Grade</Text>
          <TextInput placeholder="Value (0–100)" keyboardType="numeric"
value={value} onChangeText={setValue}
            style={{ borderWidth: 1, borderColor: '#ddd', borderRadius: 10,
padding: 12, marginBottom: 8 }} />
          <TextInput placeholder="Weight (optional, default 1)"
keyboardType="numeric" value={weight} onChangeText={setWeight}
            style={{ borderWidth: 1, borderColor: '#ddd', borderRadius: 10,
padding: 12, marginBottom: 8 }} />
          <TextInput placeholder="Note (optional)" value={note}
onChangeText={setNote}
            style={{ borderWidth: 1, borderColor: '#ddd', borderRadius: 10,
padding: 12, marginBottom: 16 }} />
          <View style={{ flexDirection: 'row', gap: 12 }}>
            <Pressable onPress={() => setAdding(false)} style={{ flex: 1,
padding: 12, borderRadius: 12, backgroundColor: '#eee' }}>
              <Text style={{ textAlign: 'center' }}>Cancel</Text>
            </Pressable>
            <Pressable onPress={onAdd} style={{ flex: 1, padding: 12,
borderRadius: 12, backgroundColor: '#ddd' }}>
              <Text style={{ textAlign: 'center', fontWeight: '700' }}>Save</
Text>
            </Pressable>
          </View>
        </View>
      </Modal>
    </View>
  );
}
```

Navigation: Push this screen with params: `{ yearId, courseId, courseName, coefficient }` from your Year detail screen.

🟩 Year Detail Screen (List Courses + Average)

Example skeleton that shows each course tile and navigates to `CourseDetailScreen`.

```
// src/screens/YearDetailScreen.js
import React, { useEffect, useState } from 'react';
import { View, Text, FlatList, Pressable } from 'react-native';
import { useNavigation, useRoute } from '@react-navigation/native';
import { auth, db } from '../firebase';
import { collection, onSnapshot, query, orderBy } from 'firebase/firestore';
import { computeWeightedAverage } from '../utils/grades';

export default function YearDetailScreen() {
  const nav = useNavigation();
  const { params } = useRoute();
  const { yearId } = params;
  const uid = auth.currentUser?.uid;

  const [courses, setCourses] = useState([]);

  useEffect(() => {
    if (!uid || !yearId) return;
    const q = query(collection(db, 'users', uid, 'years', yearId,
'courses'), orderBy('name'));
    const unsub = onSnapshot(q, (snap) => {
      setCourses(snap.docs.map(d => ({ id: d.id, ...d.data() })));
    });
    return () => unsub();
  }, [uid, yearId]);

  return (
    <View style={{ flex: 1, padding: 16 }}>
      <Text style={{ fontSize: 22, fontWeight: '700' }}>Courses</Text>
      <FlatList
        style={{ marginTop: 12 }}
        data={courses}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
          <Pressable
            onPress={() => nav.navigate('CourseDetail', {
              yearId,
              courseId: item.id,
              courseName: item.name,
              coefficient: item.coefficient,
            })}
            style={{ padding: 12, borderRadius: 12, backgroundColor:
'#f7f7f7', marginBottom: 8 }}
          >
            <Text style={{ fontSize: 16, fontWeight: '600' }}>{item.name}</
```

```
Text>
            <Text>Coefficient: {item.coefficient}</Text>
          </Pressable>
        )}
      />
    </View>
  );
}
```

🔕QA Scenarios (Manual)

1. **Fresh user** → select year 7 → category 7C → verify courses created.
2. Add grades: 75 (w=2), 60 (w omitted), 40 (w=0.5) → verify average = (75*2* + *60*1 + 40*0.5) / (2 + 1 + 0.5) = 66.67.
3. **Boundary values**: 0 and 100 accepted; 101 rejected by rules and client.
4. **Delete a grade** → list updates, average recomputes.
5. **Returning user** (logout/login) → data persists and loads.
6. **Settings**: change email, password; delete account (ensure reauth modal works).

🟩Next Execution Steps

1. **Deploy rules**: `firebase deploy --only firestore:rules`.
2. Create files from snippets above; wire routes (add `CourseDetail` and `YearDetail` to your navigator).
3. Confirm Year+Category selection uses `createStudyProgram(...)`.
4. Run on a device/simulator; execute **QA Scenarios**.

   When the boxes are checked, your MVP loop is complete. Expand templates, polish UI, and add analytics later.