lib\model.dart

```dart
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'package:google_fonts/google_fonts.dart';

class FNNModelApp extends StatefulWidget {
  const FNNModelApp({super.key});

  @override
  FNNModelAppState createState() => FNNModelAppState();
}

class FNNModelAppState extends State<FNNModelApp> {
  TextEditingController textController = TextEditingController();
  List<int> numericValues = [];
  List<double> predictionResult = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0];
  String err = "";

  // Map for DNA sequence to numeric conversion
  Map<String, int> genoMap = {
    'AA': 1,
    'AT': 2,
    'AG': 3,
    'AC': 4,
    'TT': 5,
    'TG': 6,
    'TC': 7,
    'GG': 8,
    'CG': 9,
    'CC': 10
  };

  void mapDnaToNumeric(String dnaSequence) {
    // Convert the input sequence to uppercase to handle both cases

    dnaSequence = dnaSequence.toUpperCase();

    // Iterate through the DNA sequence in pairs and map to numeric values
    List<int> values = [];
    print(dnaSequence.length);
    if (dnaSequence.length == 48) {
      for (int i = 0; i < dnaSequence.length - 1; i += 2) {
        String pair = dnaSequence.substring(i, i + 2);
        if (genoMap.containsKey(pair)) {
          values.add(genoMap[pair]!);
        } else {
          setState(() {
            err = "Error: Invalid DNA sequence pair: $pair";
            numericValues.clear();
          });
          return;
        }
      }

      setState(() {
        numericValues = values;
        err = "";
      });
    } else {
      setState(() {
        err = "Error: Invalid DNA sequence pair 48 chars required";
        numericValues.clear();
      });
    }
  }

  Future<void> sendInputToServer(List<int> input) async {
    print(input);
    // final url = Uri.parse('http://10.0.2.2:5000/pred?ip=' '$input');
    final url = Uri.parse('http://10.0.2.2:10000/predict');
    // Replace with your Flask server URL
    final headers = {"Content-Type": "application/json"};
    String jsonString = jsonEncode({'ip': input});

    print(jsonString);

    // Split the input string and trim any leading/trailing whitespace from each element.

    // Validate that inputList contains valid doubles before proceeding.
    try {
      final response = await http.post(url, headers: headers, body: jsonString);

      // final response = await http.get(url);

      if (response.statusCode == 200) {
        final data = json.decode(response.body);
        print(data);
        final predictions = data['preds'];
        setState(() {
          predictionResult = [for (var prediction in predictions) prediction];
        });
      } else {
        setState(() {
          err = "Error: Unable to make predictions";
        });
      }
    } catch (e) {
      setState(() {
        err = "Error: Invalid input please try again and $e";
        print(err);
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey,
      appBar: AppBar(
        backgroundColor: Colors.black,
        title: Text(
          "Genomic Prediction of Wheat 🌾",
          style: GoogleFonts.alegreya(
            fontSize: 19,
            fontWeight: FontWeight.bold,
            color: Colors.white,
            wordSpacing: 3,
          ),
        ),
      ),
      body: SingleChildScrollView(
        child: Container(
          padding: const EdgeInsets.all(16.0),
          alignment: Alignment.center,
          child: Column(
            children: [
              Text(
                "INPUT RULES !!\n",
                style: GoogleFonts.alegreya(
                  fontSize: 19,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                  wordSpacing: 3,
                ),
              ),
              Text(
                "Basic meaning of ATGC:\n A - Adenine\n T - Thymine\n G - Guanine \n C - Cytosine",
                style: GoogleFonts.alegreya(
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                  wordSpacing: 3,
                ),
              ),
              Text(
                "1) Enter only meaningful alphabets {A,T,G,C}\n2) Maintain the relative Order",
                style: GoogleFonts.alegreya(
                  fontSize: 19,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                  wordSpacing: 3,
                ),
              ),
              const SizedBox(height: 16),
              TextField(
                controller: textController,
                decoration: const InputDecoration(
                  labelText: 'Enter Input (A,T,G,C)',
                  labelStyle: TextStyle(
                    color: Colors.black,
                    fontSize: 19,
```

```dart
                ),
                contentPadding:
                    EdgeInsets.symmetric(horizontal: 16.0, vertical: 10.0),
              ),
            ),
            const SizedBox(height: 16),
            ElevatedButton(
              onPressed: () {
                mapDnaToNumeric(textController.text);
                sendInputToServer(numericValues);
              },
              style: ElevatedButton.styleFrom(
                backgroundColor: Color.fromARGB(255, 213, 226, 235),
              ),
              child: Text(
                "GET PREDICTION 🌾",
                style: GoogleFonts.alegreya(
                  fontSize: 13,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                ),
              ),
            ),
            const SizedBox(height: 16),
            Text(
              "Your Prediction:",
              style: GoogleFonts.alegreya(
                fontSize: 18,
                fontWeight: FontWeight.bold,
                color: const Color.fromARGB(255, 177, 4, 4),
              ),
            ),
            // Table to display predictions
            DataTable(
              columns: const [
                DataColumn(
                    label: Text('Trait',
                        style: TextStyle(
                            fontWeight: FontWeight.bold,
                            color: Colors.black))),
                DataColumn(
                        label: Text('Prediction',
                            style: TextStyle(
                                fontWeight: FontWeight.bold,
                                color: Colors.black))),
                  ],
                  rows: [
                    DataRow(cells: [
                      const DataCell(Text(
                        'Days to Heading (DH)',
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                          color: Colors.black,
                        ),
                      )),
                      DataCell(Text(
                        predictionResult.isNotEmpty
                            ? predictionResult[0].toStringAsFixed(2)
                            : '',
                        style: const TextStyle(
                          fontWeight: FontWeight.bold,
                          color: Color.fromARGB(255, 255, 255, 255),
                        ),
                      )),
                    ]),
                    DataRow(cells: [
                      const DataCell(Text(
                        'Grain Filling Duration (GFD)',
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                          color: Colors.black,
                        ),
                      )),
                      DataCell(Text(
                        predictionResult[1].toStringAsFixed(2),
                        style: const TextStyle(
                          fontWeight: FontWeight.bold,
                          color: Color.fromARGB(255, 255, 255, 255),
                        ),
                      )),
                    ]),
                    DataRow(cells: [
                  const DataCell(Text(
                    'Grain Number per Spike (GNPS)',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.black,
                    ),
                  )),
                  DataCell(Text(
                    predictionResult[2].toStringAsFixed(2),
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Color.fromARGB(255, 255, 255, 255),
                    ),
                  )),
                ]),
                DataRow(cells: [
                  const DataCell(Text(
                    'Grain Weight per Spike (GWPS)',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.black,
                    ),
                  )),
                  DataCell(Text(
                    predictionResult[3].toStringAsFixed(2),
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Color.fromARGB(255, 255, 255, 255),
                    ),
                  )),
                ]),
                DataRow(cells: [
                  const DataCell(Text(
                    'Plant Height (PH)',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.black,
                    ),
                  )),
                  DataCell(Text(
                    predictionResult[4].toStringAsFixed(2),
```

```dart
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Color.fromARGB(255, 255, 255, 255),
                    ),
                  )),
                ]),
                DataRow(cells: [
                  const DataCell(Text(
                    'Grain Yield (GY)',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Colors.black,
                    ),
                  )),
                  DataCell(Text(
                    predictionResult[5].toStringAsFixed(2),
                    style: const TextStyle(
                      fontWeight: FontWeight.bold,
                      color: Color.fromARGB(255, 255, 255, 255),
                    ),
                  )),
                ]),
              ],
            ),
            Text(
              err,
              style: GoogleFonts.alegreya(
                fontSize: 19,
                fontWeight: FontWeight.bold,
                color: Colors.red,
              ),
            )
          ],
        ),
      ),
    ),
  );
 }
}
```