

lib\model.dart

```
1 import 'package:flutter/material.dart';
2 import 'package:http/http.dart' as http;
3 import 'dart:convert';
4 import 'package:google_fonts/google_fonts.dart';
5
6 class FNNModelApp extends StatefulWidget {
7   const FNNModelApp({super.key});
8
9   @override
10  FNNModelState createState() => FNNModelState();
11 }
12
13 class FNNModelState extends State<FNNModelApp> {
14   TextEditingController textController = TextEditingController();
15   List<int> numericValues = [];
16   List<double> predictionResult = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0];
17   String err = "";
18
19   // Map for DNA sequence to numeric conversion
20   Map<String, int> genoMap = {
21     'AA': 1,
22     'AT': 2,
23     'AG': 3,
24     'AC': 4,
25     'TT': 5,
26     'TG': 6,
27     'TC': 7,
28     'GG': 8,
29     'CG': 9,
30     'CC': 10
31   };
32
33   void mapDnaToNumeric(String dnaSequence) {
34     // Convert the input sequence to uppercase to handle both cases
35
36     dnaSequence = dnaSequence.toUpperCase();
37
38     // Iterate through the DNA sequence in pairs and map to numeric values
```

```

39 List<int> values = [];
40 print(dnaSequence.length);
41 if (dnaSequence.length == 48) {
42     for (int i = 0; i < dnaSequence.length - 1; i += 2) {
43         String pair = dnaSequence.substring(i, i + 2);
44         if (genoMap.containsKey(pair)) {
45             values.add(genoMap[pair]!);
46         } else {
47             setState(() {
48                 err = "Error: Invalid DNA sequence pair: $pair";
49                 numericValues.clear();
50             });
51             return;
52         }
53     }
54
55     setState(() {
56         numericValues = values;
57         err = "";
58     });
59 } else {
60     setState(() {
61         err = "Error: Invalid DNA sequence pair 48 chars required";
62         numericValues.clear();
63     });
64 }
65 }
66
67 Future<void> sendInputToServer(List<int> input) async {
68     print(input);
69     // final url = Uri.parse('http://10.0.2.2:5000/pred?ip=' '$input');
70     final url = Uri.parse('http://10.0.2.2:10000/predict');
71     // Replace with your Flask server URL
72     final headers = {"Content-Type": "application/json"};
73     String jsonString = jsonEncode({'ip': input});
74
75     print(jsonString);
76
77     // Split the input string and trim any leading/trailing whitespace from each element.
78
79     // Validate that inputList contains valid doubles before proceeding.

```

```

80     try {
81         final response = await http.post(url, headers: headers, body: jsonString);
82
83         // final response = await http.get(url);
84
85         if (response.statusCode == 200) {
86             final data = json.decode(response.body);
87             print(data);
88             final predictions = data['preds'];
89             setState(() {
90                 predictionResult = [for (var prediction in predictions) prediction];
91             });
92         } else {
93             setState(() {
94                 err = "Error: Unable to make predictions";
95             });
96         }
97     } catch (e) {
98         setState(() {
99             err = "Error: Invalid input please try again and $e";
100             print(err);
101         });
102     }
103 }
104
105 @override
106 Widget build(BuildContext context) {
107     return Scaffold(
108         backgroundColor: Colors.grey,
109         appBar: AppBar(
110             backgroundColor: Colors.black,
111             title: Text(
112                 "Genomic Prediction of Wheat 🌾",
113                 style: GoogleFonts.alegreya(
114                     fontSize: 19,
115                     fontWeight: FontWeight.bold,
116                     color: Colors.white,
117                     wordSpacing: 3,
118                 ),
119             ),
120         ),

```

```

121 body: SingleChildScrollView(
122   child: Container(
123     padding: const EdgeInsets.all(16.0),
124     alignment: Alignment.center,
125     child: Column(
126       children: [
127         Text(
128           "INPUT RULES !!\n",
129           style: GoogleFonts.alegreya(
130             fontSize: 19,
131             fontWeight: FontWeight.bold,
132             color: Colors.black,
133             wordSpacing: 3,
134           ),
135         ),
136         Text(
137           "Basic meaning of ATGC:\n A - Adenine\n T - Thymine\n G - Guanine \n C - Cytosine",
138           style: GoogleFonts.alegreya(
139             fontSize: 18,
140             fontWeight: FontWeight.bold,
141             color: Colors.black,
142             wordSpacing: 3,
143           ),
144         ),
145         Text(
146           "1) Enter only meaningful alphabets {A,T,G,C}\n2) Maintain the relative Order",
147           style: GoogleFonts.alegreya(
148             fontSize: 19,
149             fontWeight: FontWeight.bold,
150             color: Colors.black,
151             wordSpacing: 3,
152           ),
153         ),
154         const SizedBox(height: 16),
155         TextField(
156           controller: textController,
157           decoration: const InputDecoration(
158             labelText: 'Enter Input (A,T,G,C)',
159             labelStyle: TextStyle(
160               color: Colors.black,
161               fontSize: 19,

```

```

162     ),
163     contentPadding:
164         EdgeInsets.symmetric(horizontal: 16.0, vertical: 10.0),
165     ),
166 ),
167 const SizedBox(height: 16),
168 ElevatedButton(
169     onPressed: () {
170         mapDnaToNumeric(textController.text);
171         sendInputToServer(numericValues);
172     },
173     style: ElevatedButton.styleFrom(
174         backgroundColor: Color.fromARGB(255, 213, 226, 235),
175     ),
176     child: Text(
177         "GET PREDICTION 🏁",
178         style: GoogleFonts.alegreya(
179             fontSize: 13,
180             fontWeight: FontWeight.bold,
181             color: Colors.black,
182         ),
183     ),
184 ),
185 const SizedBox(height: 16),
186 Text(
187     "Your Prediction:",
188     style: GoogleFonts.alegreya(
189         fontSize: 18,
190         fontWeight: FontWeight.bold,
191         color: const Color.fromARGB(255, 177, 4, 4),
192     ),
193 ),
194 // Table to display predictions
195 DataTable(
196     columns: const [
197         DataColumn(
198             label: Text('Trait',
199                 style: TextStyle(
200                     fontWeight: FontWeight.bold,
201                     color: Colors.black))),
202         DataColumn(

```

```

203         label: Text('Prediction',
204                     style: TextStyle(
205                         fontWeight: FontWeight.bold,
206                         color: Colors.black))),
207     ],
208     rows: [
209         DataRow(cells: [
210             const DataCell(Text(
211                 'Days to Heading (DH)',
212                 style: TextStyle(
213                     fontWeight: FontWeight.bold,
214                     color: Colors.black,
215                 ),
216             )),
217             DataCell(Text(
218                 predictionResult.isNotEmpty
219                     ? predictionResult[0].toStringAsFixed(2)
220                     : '',
221                 style: const TextStyle(
222                     fontWeight: FontWeight.bold,
223                     color: Color.fromARGB(255, 255, 255, 255),
224                 ),
225             )),
226         ]),
227         DataRow(cells: [
228             const DataCell(Text(
229                 'Grain Filling Duration (GFD)',
230                 style: TextStyle(
231                     fontWeight: FontWeight.bold,
232                     color: Colors.black,
233                 ),
234             )),
235             DataCell(Text(
236                 predictionResult[1].toStringAsFixed(2),
237                 style: const TextStyle(
238                     fontWeight: FontWeight.bold,
239                     color: Color.fromARGB(255, 255, 255, 255),
240                 ),
241             )),
242         ]),
243         DataRow(cells: [

```

```

244     const DataCell(Text(
245         'Grain Number per Spike (GNPS)',
246         style: TextStyle(
247             fontWeight: FontWeight.bold,
248             color: Colors.black,
249         ),
250     )),
251     DataCell(Text(
252         predictionResult[2].toStringAsFixed(2),
253         style: const TextStyle(
254             fontWeight: FontWeight.bold,
255             color: Color.fromARGB(255, 255, 255, 255),
256         ),
257     )),
258 ],
259 DataRow(cells: [
260     const DataCell(Text(
261         'Grain Weight per Spike (GWPS)',
262         style: TextStyle(
263             fontWeight: FontWeight.bold,
264             color: Colors.black,
265         ),
266     )),
267     DataCell(Text(
268         predictionResult[3].toStringAsFixed(2),
269         style: const TextStyle(
270             fontWeight: FontWeight.bold,
271             color: Color.fromARGB(255, 255, 255, 255),
272         ),
273     )),
274 ],
275 DataRow(cells: [
276     const DataCell(Text(
277         'Plant Height (PH)',
278         style: TextStyle(
279             fontWeight: FontWeight.bold,
280             color: Colors.black,
281         ),
282     )),
283     DataCell(Text(
284         predictionResult[4].toStringAsFixed(2),

```

```

285         style: const TextStyle(
286             fontWeight: FontWeight.bold,
287             color: Color.fromARGB(255, 255, 255, 255),
288         ),
289     )),
290 ],
291 DataRow(cells: [
292     const DataCell(Text(
293         'Grain Yield (GY)',
294         style: TextStyle(
295             fontWeight: FontWeight.bold,
296             color: Colors.black,
297         ),
298     )),
299     DataCell(Text(
300         predictionResult[5].toStringAsFixed(2),
301         style: const TextStyle(
302             fontWeight: FontWeight.bold,
303             color: Color.fromARGB(255, 255, 255, 255),
304         ),
305     )),
306 ],
307 ],
308 ),
309 Text(
310     err,
311     style: GoogleFonts.alegreya(
312         fontSize: 19,
313         fontWeight: FontWeight.bold,
314         color: Colors.red,
315     ),
316 ),
317 ],
318 ),
319 ),
320 ),
321 );
322 }
323 }
324

```