TECNOLOGIE CLOUD E MOBILE

SWEETEDx Dreams

ALESSIA LAZZARI 1078863

TRELLO, GITHUB,

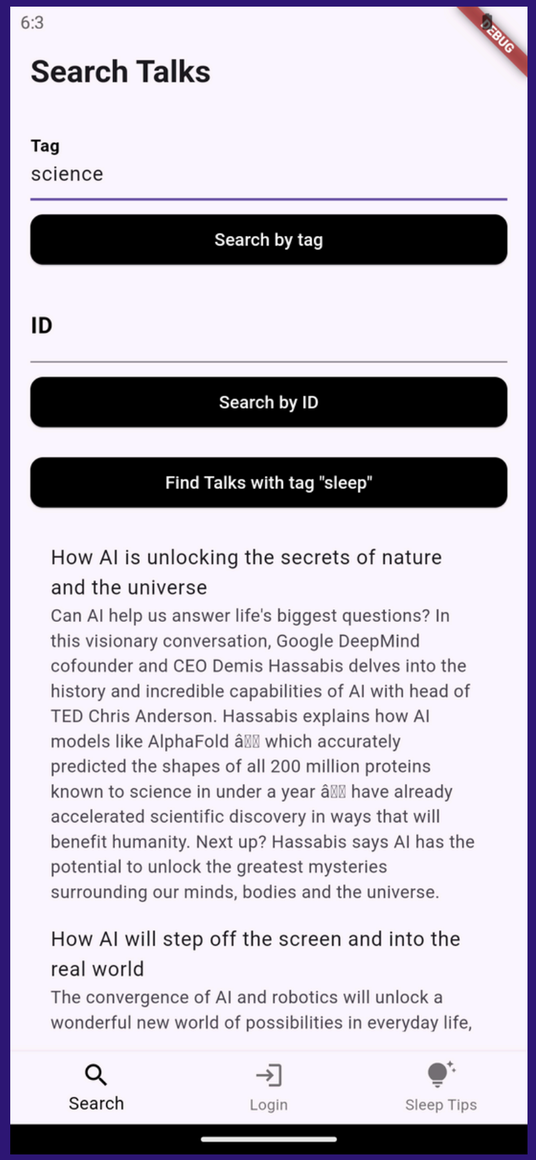
SEARCH TALK - BY TAG, BY ID

La prima funzione implementata ha lo scopo di trovare i talk associati a un determinato "tag" nel primo caso e a un "ID" nel secondo caso, facendo uso dell'API creata nel compito precedente. Se la ricerca ha successo, i talk corrispondenti vengono visualizzati a schermo. In caso contrario, viene sollevata un'eccezione e viene stampato un messaggio di errore.

```dart
Future<List<Talk>> getTalksByTag(String tag, int page) async {
  var url = Uri.parse('https://ecp0cy67m2.execute-api.us-east-1.amazonaws.com/default/Get_Talks_By_Tag');

  final http.Response response = await http.post(
    url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, Object>{
      'tag': tag,
      'page': page,
      'doc_per_page': 6
    }),
  );

  if (response.statusCode == 200) {
    Iterable list = json.decode(response.body);
    var talks = list.map((model) => Talk.fromJSON(model)).toList();
    if (talks.isEmpty) {
      throw Exception('No talks found with tag: $tag');
    }
    return talks;
  } else {
    throw Exception('Failed to load talks');
  }
}
```

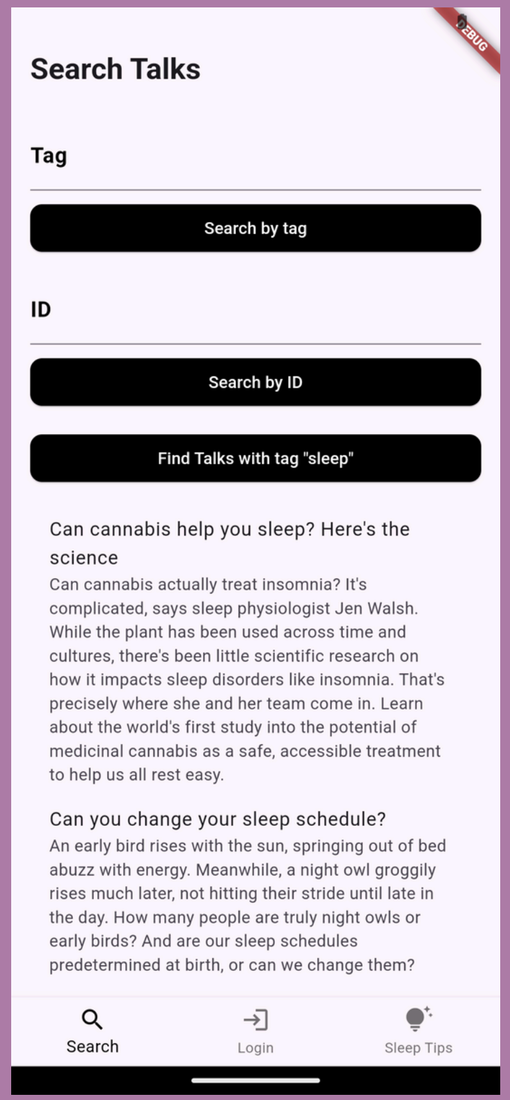
```dart
class _MySearchPage extends State<MySearchPage> {
  final TextEditingController _tagController = TextEditingController();
  final TextEditingController _idController = TextEditingController();

  Future<List<Talk>>? _tagTalks;
  Future<List<Talk2>>? _idTalks;
  bool initTag = true;
  bool initId = true;

  @override
  void initState() {
    super.initState();
    _tagTalks = initEmptyList();
    _idTalks = initEmptyList2();
  }

  void _getTalksByTag() {
    setState(() {
      _tagTalks = getTalksByTag(_tagController.text, 1);
      _idTalks = initEmptyList2();
      _tagController.clear();
    });
  }

  void _getTalksById() {
    setState(() {
      _idTalks = getTalksById(_idController.text, 1);
      _tagTalks = initEmptyList();
      _idController.clear();
    });
  }
```

RISULTATO



Dal punto di vista grafico, ho creato due textfield dove l'utente può specificare il tag e/o l'Id del talk di interesse.

Se la ricerca va a buon fine vengono stampati i talk corrispondenti e la relativa descrizione.



Nella stessa schermata ho creato un terzo bottone che stampa a schermo solo i talk relativi al sonno.

Ho utilizzato la funzione getTalksByTag(), specificando in questo caso il tag "sleep".

```
void _getTalksBySleepTag() {
  setState(() {
    _tagTalks = getTalksByTag('sleep', 1);
  });
}
```

FUNZIONALITÀ DEL SERVIZIO

Tra le seguenti funzionalità, ne sono state implementate due:entate

1. **Raccomandazioni Personalizzate di Contenuti**
2. **Modalità Notturna**
3. **Timer di Spegnimento Automatico**
4. **Raccolta di Feedback e Valutazioni (implica login utente)**
5. **Creazione di Playlist Personalizzate**
6. **Integrazione con Dispositivi e App di Monitoraggio del Sonno**
7. **Consigli e Risorse per il Benessere Notturno**

Funzionalità scelta

04 **Login utente**

Funzionalità scelta

03 **Timer di Spegnimento Automatico**

ALESSIA LAZZARI | CLOUD E MOBILE

LOGIN.DART

Utilizzo l'API LOGINSD per implementare il servizio di autenticazione dell'app.

```dart
Future<String> login(String mail, String password) async {
  var url = Uri.parse(
      'https://a98p6r9e0e.execute-api.us-east-1.amazonaws.com/default/LoginSD');
  var body = jsonEncode(<String, String>{
    'mail': mail,
    'password': password,
  });

  print('Sending POST request to $url');
  print('Request body: $body');
```

Nella classe LoginPage sono stati creati alcuni campi per controllare la validità dei dati inseriti dall'utente. Se questi dati non corrispondono, il login fallisce e non è possibile accedere.

```dart
class LoginPage extends StatefulWidget {
  final int page;
  final String initTag;

  const LoginPage({super.key,
    this.page = 0,
    this.initTag = ''
  });

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

class _LoginPageState extends State<LoginPage> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  String _message = '';

  Future<void> _login() async {
    final response = await login(_emailController.text, _passwordController.text);

    setState(() {
      _message = response;
    });

    if (response == 'login successful') {
      // Redirect to MainNavigation after successful login
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const MySearchPage()),
      );
    }
  }
```

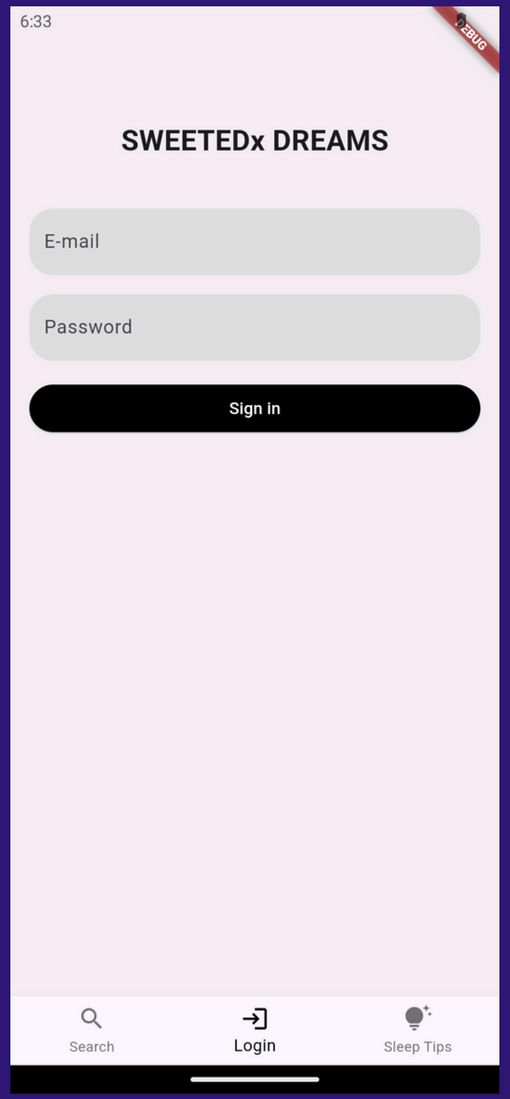
TIMERSCREEN.DART

Utilizzo l'API TimerAutoSpegnimento per implementare la modalità di timer di spegnimento automatico.

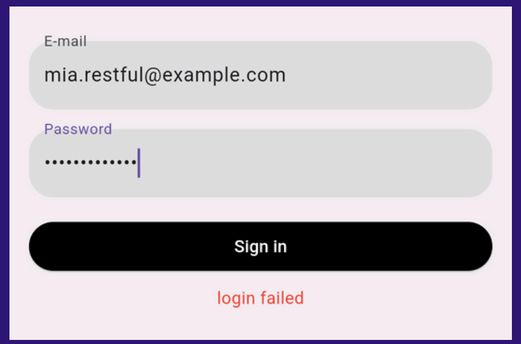
Il timer implementato permette di selezionare un talk, selezionare un intervallo di tempo predefinito al termine del quale il talk termina la sua riproduzione. Se non viene scelto un valore, viene settato un valore prestabilito di 30 minuti.

```dart
Future<String> setAutoShutdownTimer(String url) async {
  var url = Uri.parse('https://hr9iua7818.execute-api.us-east-1.amazonaws.com/default/TimerAutoSpegnimento');

  var minutes;
  var body = jsonEncode(<String, dynamic>{
    'minutes': minutes,
  });

  print('Sending POST request to $url');
  print('Request body: $body');

  final response = await http.post(
    url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: body,
  );

  print('Response status: ${response.statusCode}');
  print('Response body: ${response.body}');

  if (response.statusCode == 200) {
    var responseBody = jsonDecode(response.body);
    if (responseBody['success']) {
      return 'Timer set successfully'; // Timer impostato con successo
    } else {
      return responseBody['message'] ?? 'Failed to set timer'; // Errore impostazione timer
    }
  } else {
    return 'Server error: ${response.statusCode}'; // Errore del server
  }
}
```

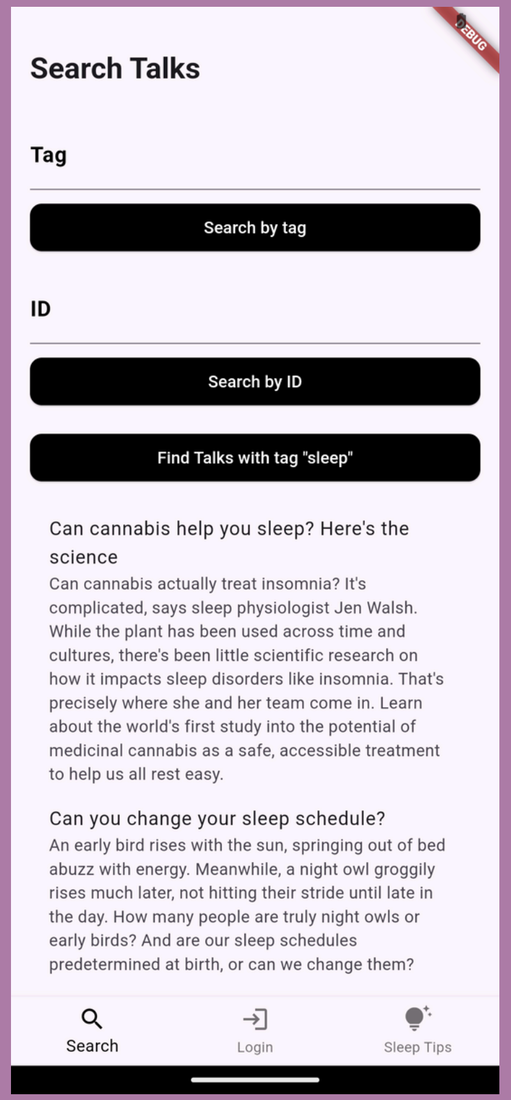
LOGIN.DART



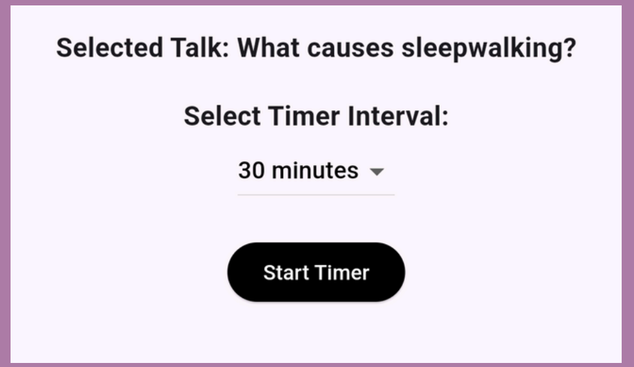
Se i dati inseriti dall'utente sono errati, viene stampato a schermo *login failed* ed è possibile ricompilare i campi.



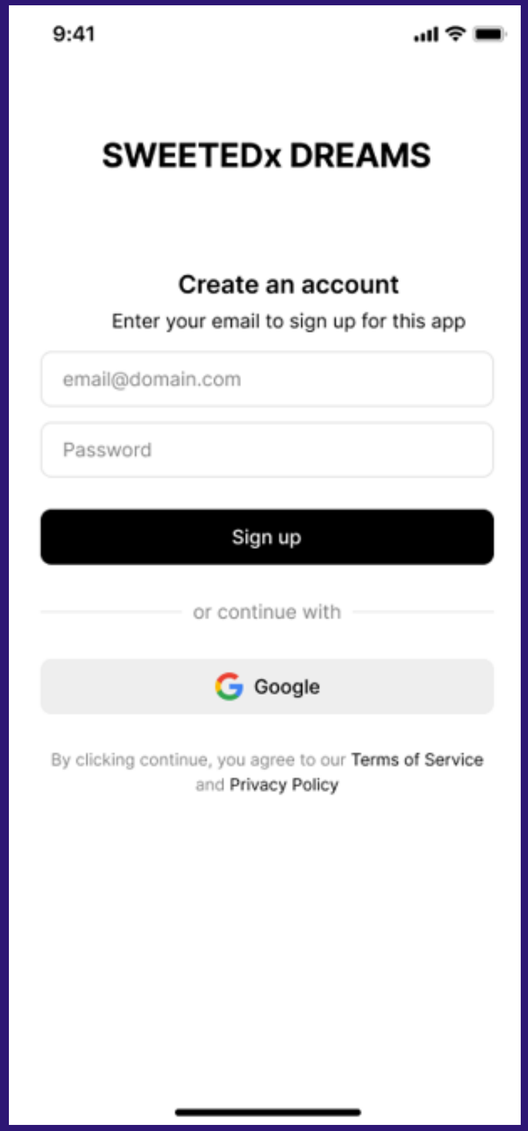
TIMERSCREEN.DART



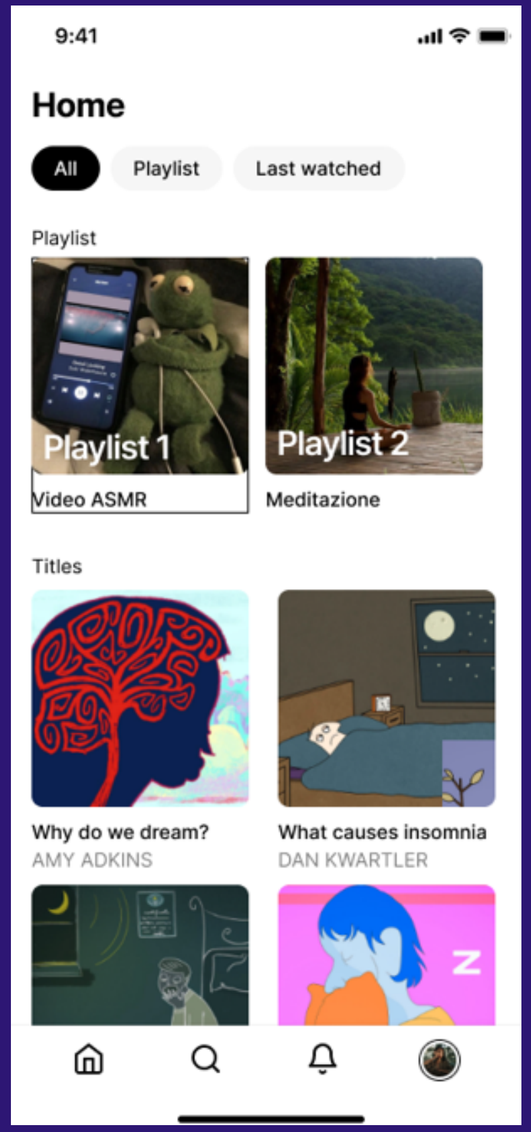
Dalla stessa schermata di getTalksByTag() e getTalksById() è possibile selezionare un talk e si viene reindirizzati a questa pagina dove è possibile selezionare un timer di autospegnimento.



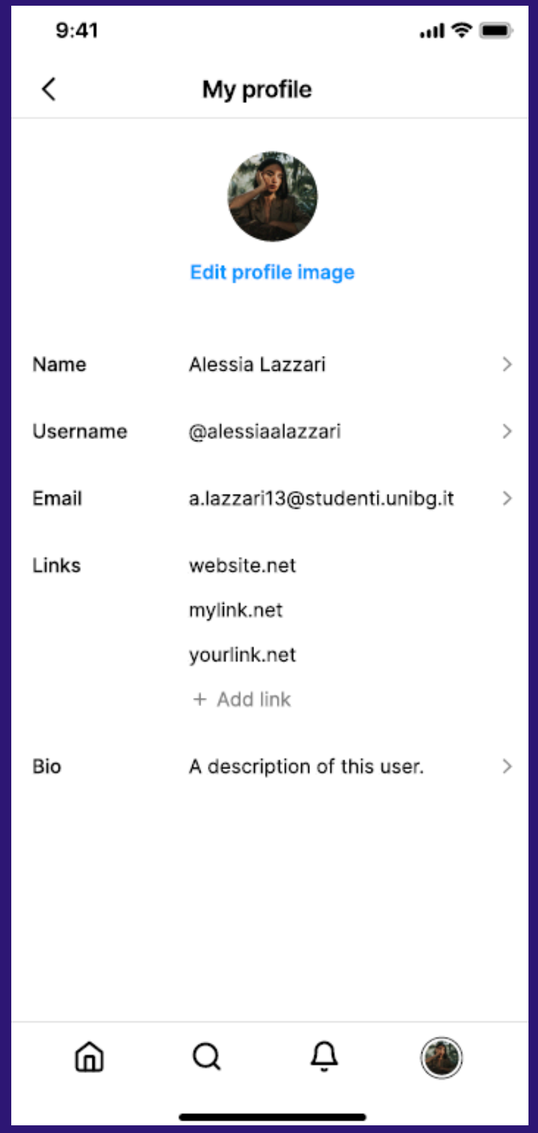
schermata login



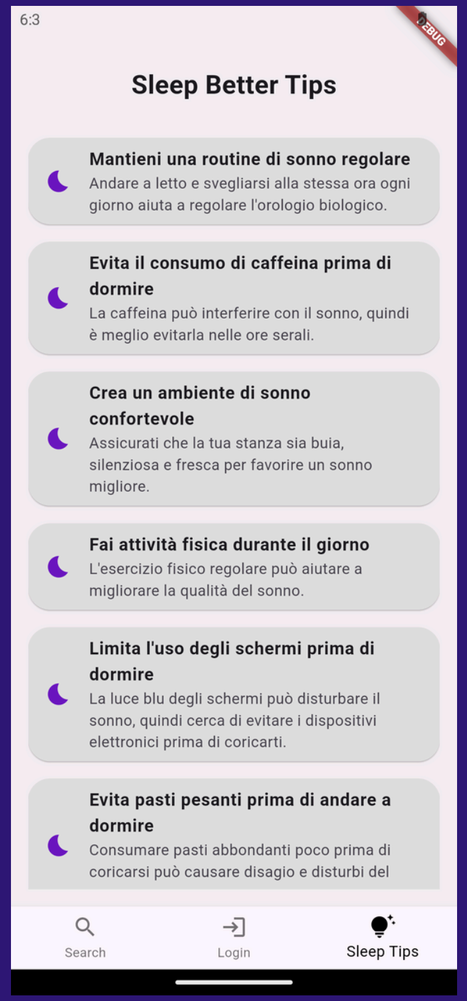
schermata home

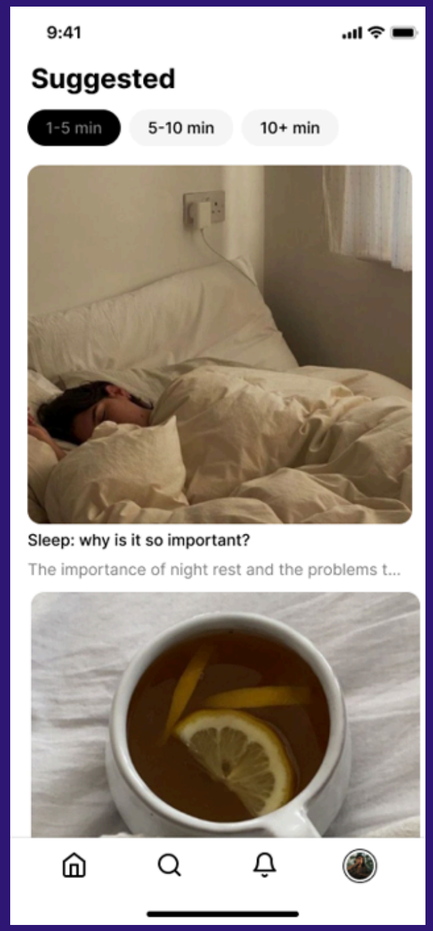


schermata account



schermata articoli





ALESSIA LAZZARI | CLOUD E MOBILE

CRITICITÀ ED EVOLUZIONI

LOGIN SEMPLICE E LIMITATO

L'implementazione del servizio di autenticazione è semplice e limitata, ma rappresenta un primo passo verso il futuro dell'applicazione.

PRINCIPALI NOVITÀ

Oltre ad integrare il servizio di login attraverso l'account google, l'app vuole offrire agli utenti la possibilità di valutare e recensire i video

TIMER DI SPEGNIMENTO

Il timer di spegnimento rappresenta la parte più complessa del servizio ed è difatti semplificata rispetto all'idea originale.

INTEGRAZIONE DI FUNZIONALITÀ

Il servizio si pone l'obiettivo di implementare altre funzionalità tra quelle dichiarate e, dove possibile, integrarle a quella del timer di autospegnimento.