



vervint.

We're Back



For our time
together:

Agenda



What We've Been Up To



What We've Learned



A New Resource?



Post Grad Life

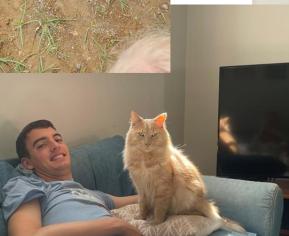


I'm a Dad...*

Of Cats!



MARRIAGE AND MORE



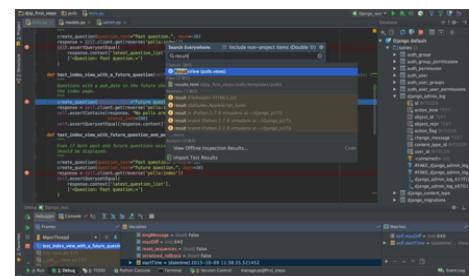
STARTING AT STOW

- Great experience so far
- Onboarding = drink from the firehose
 - I also got a Django tutorial from my homey Mosh
- Foosball every morning
- Learning and asking questions
- Don't be afraid to look dumb - you'll be smarter later
- Have contributed in major ways to the design tool and the easyclosets.com website redesign project



DJANGO FRAMEWORK

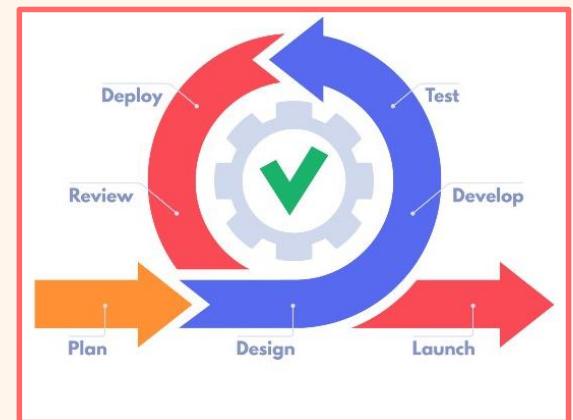
Django is a high-level Python web framework designed for rapid development, clean design, and scalability. It follows the Model-View-Template (MVT) architecture and emphasizes the "Don't Repeat Yourself" (DRY) principle, allowing developers to build robust, maintainable applications efficiently. Django includes a powerful ORM for database interactions, an admin interface for content management, built-in authentication, and security features like protection against SQL injection and CSRF attacks. With its modular and extensible nature, Django is widely used for web applications ranging from simple blogs to complex, enterprise-level systems. - ChatGPT



What We've Learned



Agile



What I Love

Honest Feedback

Open Communication

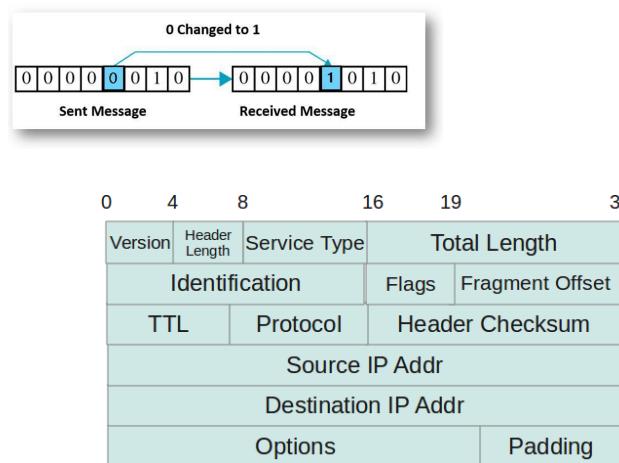
Devotion to Quality



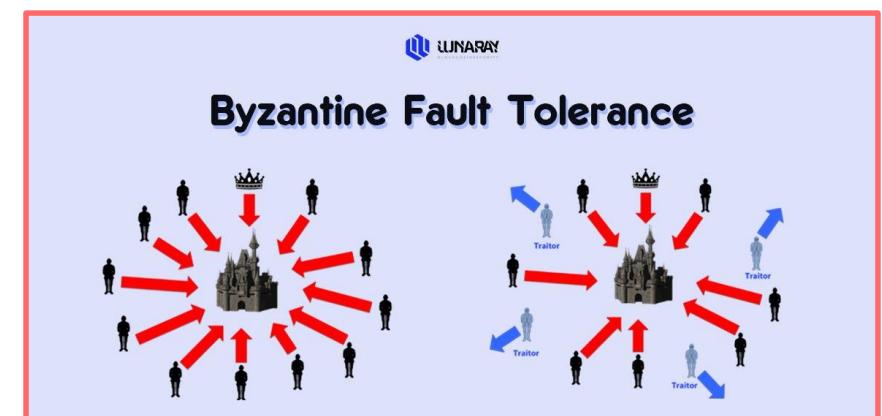
Things I wish I knew

© 2024 → vervint.com

Building Reliable
Things on an
Unreliable
Foundation



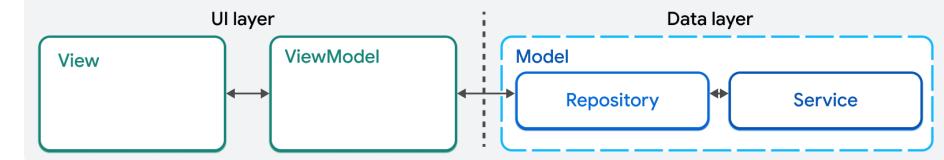
Software Problems from a Real Life POV



© 2024 → vervint.com

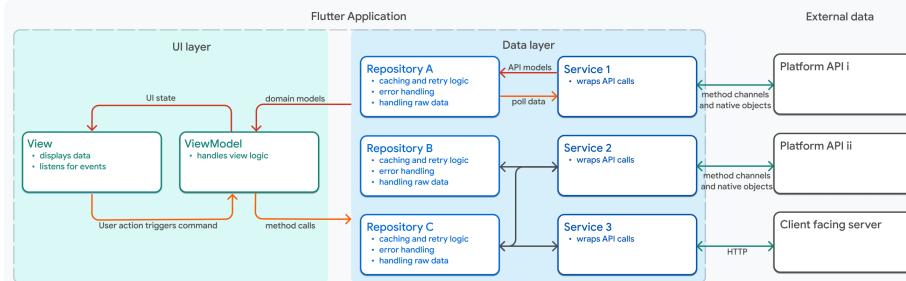
MVVM in Flutter

What is MVVM *Guide to app architecture*



© 2024 → vervint.com

What is MvvM *Guide to app architecture*



© 2024 → vervint.com

In ImHere!

```
1 class _ProfessorDashState extends State<ProfessorDash> {
2     List<Class> classes = [];
3     List<Class> enrolments = [];
4     bool startingAttendance = false;
5
6     @override
7     void initState() {
8         ...
9     }
10
11    void onClassTap(BuildContext context, Class course) {
12        ...
13    }
14
15    Future<bool> _refresh(Person person) async {
16        ...
17    }
18
19    void _refreshWithSnack(BuildContext context) async {
20        ...
21    }
}
```

```
1 Future<bool> deleteClass(Class course) async {
2     ...
3 }
4
5 @override
6 Widget build(BuildContext context) {
7     ...
8 }
9
10 Route _createRoute(Class course) {
11     ...
12 }
13
```

© 2024 → vervint.com

In ImHere!

```
1 Route _createRoute(Class course) {
2   return PageRouteBuilder(
3     pageBuilder: (context, animation, secondaryAnimation) =>
4       TakingAttendance(course: course),
5     transitionsBuilder: (context, animation, secondaryAnimation, child) {
6       const begin = Offset(0.0, 1.0);
7       const end = Offset.zero;
8       const curve = Curves.ease;
9
10      var tween = Tween(begin: begin, end: end).chain(CurveTween(curve: curve));
11
12      return SlideTransition(
13        position: animation.drive(tween),
14        child: child,
15      );
16    },
17  );
}
```

© 2024 → vervint.com



Code Organization

Architecture case study

```
lib
|____ui
| |____core
| | |____ui
| | | |____<shared widgets>
| | |____themes
| |____<FEATURE NAME>
| | |____view_model
| | | |____<view_model class>.dart
| | |____widgets
| | | |____<feature name>_screen.dart
| | |____<other widgets>
```

Architecture case study
<https://docs.flutter.dev/app-architecture/case-study>

```
ui/core/animations/routes
route_animations.dart
```

```
navigator.push(RouteAnimations.slideToTakingAttendanceRoute(course));
```

© 2024 → vervint.com



```
1 Future<bool> deleteClass(Class course) async {
2   var scaffold = ScaffoldMessenger.of(context);
3   try {
4     var server = GetIt.instance<ServerMethods>();
5     bool response = await server.classMethods
6       .deleteClass(course.id, course.professorId, course.className!);
7     if (!response) {
8       scaffold.showSnackBar(const Snackbar(
9         content: Text("Failed to delete class"),
10        ));
11    }
12    // remove class from list without refreshing
13    setState(() {
14      classes.remove(course);
15    });
16    return response == true; // returns false if response is null
17  } catch (e) {
18    scaffold.showSnackBar(const Snackbar(
19      content: Text("Failed to delete class"),
20    ));
21    return false;
22  }
23 }
```

24 → vervint.com

```
1 Future deleteClass(BuildContext context, Class course) async {
2   var scaffold = ScaffoldMessenger.of(context);
3
4   var response = await courseRepository.deleteCourse(
5     course.id, course.professorId, course.className!
6   );
7
7   if (!response) {
8     scaffold.showSnackBar(const Snackbar(
9       content: Text("Failed to delete class"),
10      ));
11    return;
12  }
13
14   classes.remove(course);
15   notifyListeners();
16 }
```

© 2024 → vervint.com

```
1 abstract class ICourseRepository {
2   Future<bool> deleteCourse(int courseId, int professorId, String courseName);
3 }
4
5 class CourseRepository implements ICourseRepository {
6   final courseApi = GetIt.instance<ServerMethods>().classMethods;
7
8   @override
9   Future<bool> deleteCourse(
10     int courseId, int professorId, String courseName) async {
11   var success =
12     await courseApi.deleteClass(courseId, professorId, courseName);
13
14   if (!success) {
15     safePrint("Failed to delete course, trying again");
16     success = await courseApi.deleteClass(courseId, professorId, courseName);
17   }
18   return success;
19 }
20
21 class CourseRepositoryMock implements ICourseRepository {
22   @override
23   Future<bool> deleteCourse(int courseId, int professorId, String courseName) {
24     throw UnimplementedError();
25   }
26 }
27 }
```

24 → vervint.com



```
1 class ProfessorDashboardViewModel extends ChangeNotifier {
2   List<Class> classes = [];
3   List<Class> enrollments = [];
4   bool startingAttendance = false;
5
6   late ICourseRepository courseRepository;
7   late IPersonRepository personRepository;
8
9   ProfessorDashboardViewModel() {
10   var getItInstance = GetIt.instance;
11   courseRepository = getItInstance<ICourseRepository>();
12   personRepository = getItInstance<IPersonRepository>();
13 }
```

© 2024 → vervint.com

```
1 class ProfessorDash extends StatefulWidget {
2   const ProfessorDash({super.key});
3
4   @override
5   State<ProfessorDash> createState() => _ProfessorDashState();
6 }
7
8 class _ProfessorDashState extends State<ProfessorDash> {
9   List<Class> classes = [];
10  List<Class> enrollments = [];
11  bool startingAttendance = false;
12
13  @override
14  void initState() {
15    ...
16  }
17  void onClassTap(BuildContext context, Class course) {
18    ...
19  }
20  Future<bool> _refresh(Person person) async {
21    ...
22  }
23  void refreshWithSnack(BuildContext context) async {
24    ...
25  }
26  Future<bool> deleteClass(Class course) async {
27    ...
28  }
29  void quickTakeAttendance(BuildContext context, Class course) async {
30    ...
31  }
32  void _quickTakeAttendance(BuildContext context, Class course) async {
33    ...
34  }
35  @override
36  Widget build(BuildContext context) {
37    ...
38  }
39  Route _createRoute(Class course) {
40    ...
41  }
42  void _refreshWithSnack(Person person) async {
43    ...
44  }
45 }
```

© 2024 → vervint.com

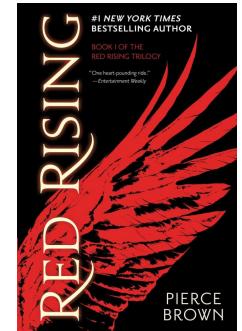
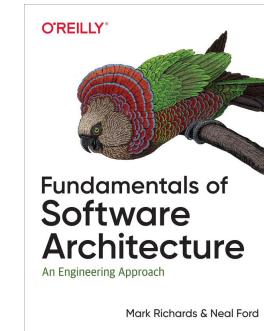


```
1 class ProfessorDash extends StatelessWidget {
2   ProfessorDash({super.key});
3
4   final ProfessorDashboardViewModel viewModel = ProfessorDashboardViewModel();
5
6   @override
7   Widget build(BuildContext context) {
8     ...
9   }
10
11  void onTakeAttendanceTap(BuildContext context) {
12    ...
13  }
14
15  void onClassTap(BuildContext context, Class course) {
16    ...
17  }
18
19  Future deleteClass(BuildContext context, Class course) async {
20    ...
21  }
22
23  Future refresh(BuildContext context, Person person) async {
24    ...
25  }
26
27  void refreshWithSnack(BuildContext context, Person person) async {
28    ...
29  }
30
31  void quickTakeAttendance(BuildContext context, Class course) async {
32    ...
33  }
34
35  void _refreshWithSnack(Person person) async {
36    ...
37  }
38 }
```

```
1 class ProfessorDashboardViewModel extends ChangeNotifier {
2   List<Class> classes = [];
3   List<Class> enrollments = [];
4   bool startingAttendance = false;
5
6   late ICourseRepository courseRepository;
7   late IPersonRepository personRepository;
8
9   ProfessorDashboardViewModel() {
10   var getItInstance = GetIt.instance;
11   courseRepository = getItInstance<ICourseRepository>();
12   personRepository = getItInstance<IPersonRepository>();
13 }
14
15 void onTakeAttendanceTap(BuildContext context) {
16   ...
17 }
18
19 void onClassTap(BuildContext context, Class course) {
20   ...
21 }
22
23 Future deleteClass(BuildContext context, Class course) async {
24   ...
25 }
26
27 Future refresh(BuildContext context, Person person) async {
28   ...
29 }
30
31 void refreshWithSnack(BuildContext context, Person person) async {
32   ...
33 }
34
35 void quickTakeAttendance(BuildContext context, Class course) async {
36   ...
37 }
38 }
```

What has Helped Me

Book Recommendations



© 2024 → vervint.com

Freakonomics Podcast

A screenshot of the Freakonomics Radio website. At the top, there's a navigation bar with links for PODCASTS, SUBSCRIPTION, LIVE SHOWS, ON THE RADIO, BOOKS, SHOP, ABOUT, BLOG, CONTACT, and a search icon. Below the navigation is a section titled 'Freakonomics Radio' featuring a thumbnail for an episode and a 'FOLLOW THIS SHOW' button. A paragraph of text describes the show, mentioning Stephen J. Dubner. At the bottom, a yellow box highlights episode 263, 'In Praise of Maintenance (Update)', with a timestamp of 9/15/24 and 42:37.

Documentation

Resource	POST	GET	PUT	DELETE
/customers	Create a new customer	Retrieve all customers	Bulk update of customers	Remove all customers
/customers/1	Error	Retrieve the details for customer 1	Update the details of customer 1 if it exists	Remove customer 1
/customers/1/orders	Create a new order for customer 1	Retrieve all orders for customer 1	Bulk update of orders for customer 1	Remove all orders for customer 1

RESTful web API design
<https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design>

© 2024 → vervint.com

Endless POC (Sandbox Approach)



© 2024 → vervint.com



Developer Roadmap

JustinFay01 / README.ed
Justin's 2025 Developer Roadmap

Description

The goal of this document is to begin plotting skills and proficiencies within each area. This will include sections such as #Cloud or #DevOps. This document will most certainly change as the year goes on, goals that weren't originally planned may be included and marked off on the same day, or they may never be checked off due to time constraints or other factors.

When an object is checked off the list I hope to include a brief synopsis of where the achievement came from. This could be as detailed as an overview of how I can find resources to review what I have previously learned or something as simple as a one line summary.

Some sections may also include a reasoning, outlining why I wanted to do something or what I was hoping to gain. They may even have a link to there own document for more in depth goals.

But, I want to know what I am striving for, what I have thrived in, and how I was able to succeed.

Cloud

Azure

AZ-900 (Azure Fundamentals Certification)
• Completed 1/12/2023
AZ-204 (Microsoft Certified: Azure Developer Associate)

Profile: Justin Fay (JustinFay01) - 3 followers, 3 following

Followers: Vervint, Michigan, InfJustin-fay-hopecollege

Organizations: InfJustin-fay-hopecollege

© 2024 → vervint.com



A New Resource

Questions — Feedback — Ideas

