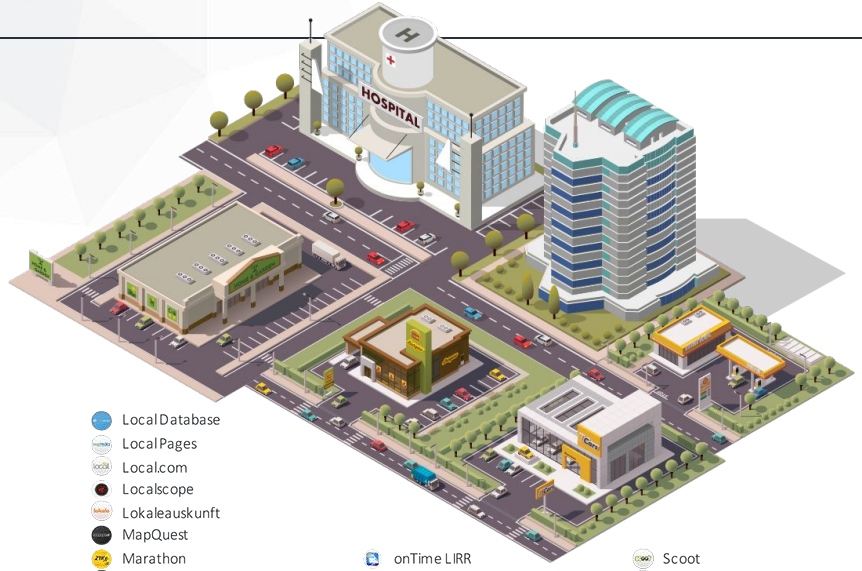
yext

Writing a Hackathon App In Golang

#1 Location Management Platform & Ecosystem



- Apple
- Facebook
- Google
- 1&1
- 5k
- 8coupons
- 10k
- 411.Com
- 192.Com
- 1888
- abcNotes
- AirYell
- AroundMe
- Avantar
- Better Fonts
- Bing
- bizwiki.co.uk
- Branchenbuch Deutschland
- Brownbook
- BundesTelefonbuch
- Busqueda-local.es
- Buurtlink.nl
- ChamberofCommerce.com
- CityMaps
- CitySquares
- CitySearch
- CoPilot
- Countdown
- Craigslist
- Credibility Review
- Credibility.com
- Custom Keyboard
- Cycling
- Cylex
- Daily Record
- Das kleine Badische
- Detelefoongids.nl
- DexKnows
- Dialo
- Elocal
- EZLocal
- Factual
- Find Near Me
- Focus
- Font Candy
- Forecast Now
- Foursquare
- Freieauskunft
- GasBuddy
- Gelbe Seiten
- GetFave
- GMX
- Goldenpages.ie
- Here
- Herold.at
- iBegin
- iGlobal
- The Independent
- Infobel
- Insider Pages
- Local Database
- Local Pages
- Local.com
- Localscope
- Lokaleauskunft
- MapQuest
- Marathon
- Meinestadt
- Merchant Circle
- The Mirror
- My Local Services
- MyRadar
- MyTown.ie
- MyTransit NYC
- MyTuner
- N49
- Najisto.cz
- Navmii
- NOAA Now
- Oeffnungszeiten
- onTime LIRR
- Opendi
- Openingstijden
- Pages24
- Pagesdor.be
- Panoramika.cz
- Perfect365: Portrait
- Photofy
- Piclay
- Pocketly
- Poynt
- Quakefeed
- Ricarce Imprese
- Roadtrippers
- Scoot
- The Scotsman
- Shoply
- Show Me Local
- SkyView
- SloPro
- Smart Ride
- SplitPic
- Stadtbranchenbuch.de
- Start Running!
- Stick Texting Lite
- Studio Design
- The Sun
- Superpages
- Switchboard
- Tapatalk
- Telenav
- TomTom
- Topix
- TouchLocal
- Transit Tracker
- Twitter
- VebidooBiz
- Viber
- Vinden.nl
- Voradius
- White & Yellow Pages
- WhitePages
- Weather Cast HD
- Weather Warning
- Web.de
- Where To?
- Wisepilot
- YP
- YaSabe
- Yahoo
- Yandex
- Yalwa
- YellowMoxie
- YellowPageCity
- YellowMap
- Yellowise
- Yell
- Yelp
- Zlate Sranky
- Zoeken.nl

yext

Building A Winning Hackathon Project

- Find partners
- Grab all the free swag
- Come up with a cool idea
- (attend this tech talk)
- Code all night
- Eat free food
- Profit???

yext

Talk Objective

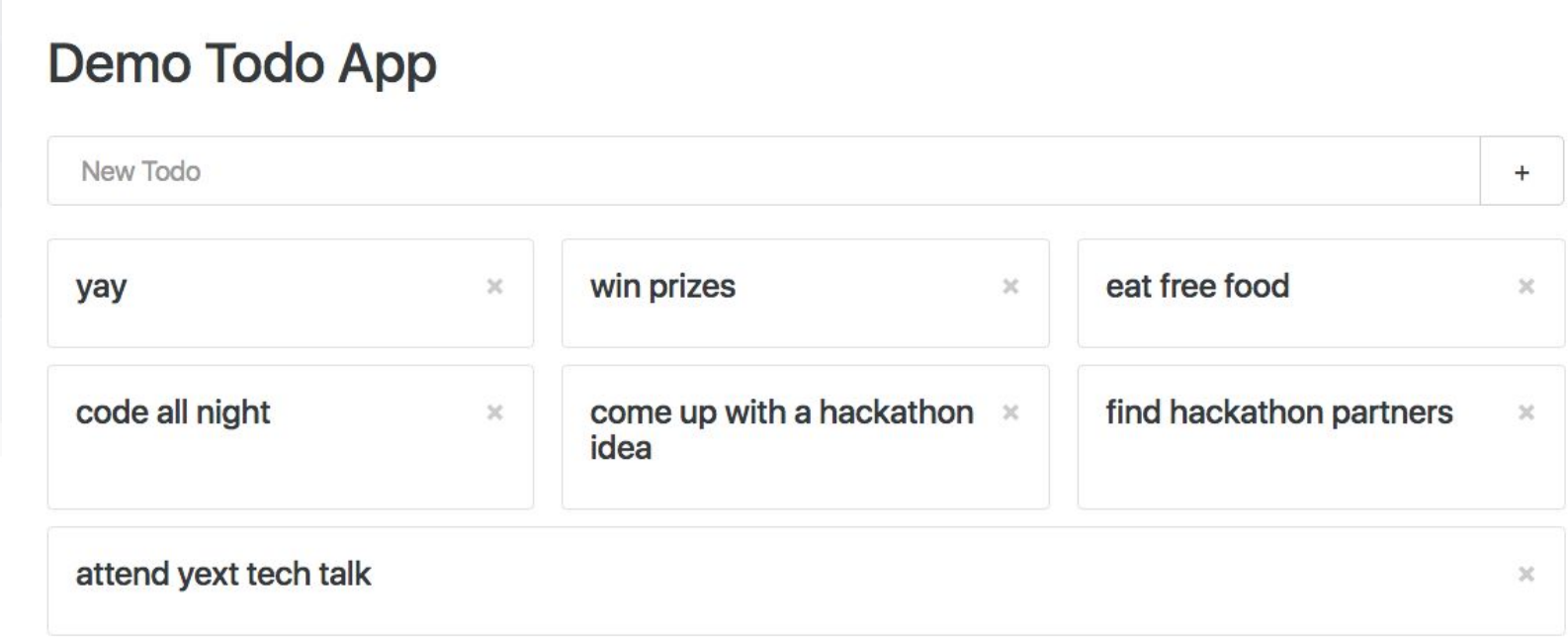
Create a simple Todo app with Golang and

Create, Read, Update, Delete Todo cards

- Introduce Golang
- Structure of the project
- Creating a RESTful API in Golang (optimized for hackathons)
- Bonus: go over frontend code

yext

Preview



yext

Assumed Knowledge

To get the most out of the talk, best to know:

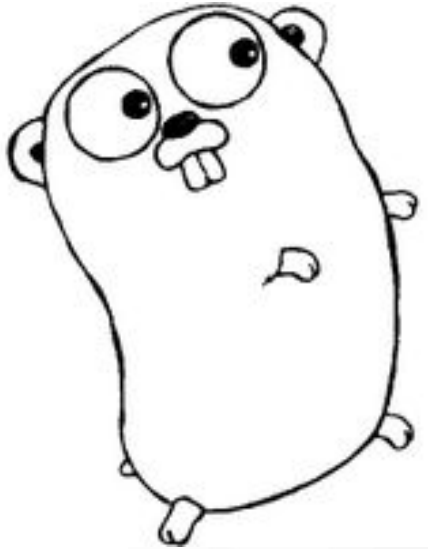
- At least one programming language
- Familiar with pointers
- Preferably familiar with C-style languages or Python
- Familiar with APIs

yext

What is Golang?

Also known as Go, Golang is

- A free open source programming language created at Google
- Compiled
- Statically typed
- Garbage collected



yext

Why Use Go?

- Productive and readable
- Fast compilation
- Fast execution
- Great built in tooling
- Native support for networking and multiprocessing

yext

Go At Yext

At Yext, we use Go for

- Webservices
- Microservices
- Continuous Integration
- Error alerting
- Managing local microservice instances

Hello World Webservice

Server.go

```
package main

import (
	"fmt"
	"log"
	"net/http"
)

func main() {
	http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
		fmt.Fprintf(w, "hello world")
	})
	log.Fatal(http.ListenAndServe(":8000", nil))
}
```

yext

Running Hello World Webservice

After you have installed Go (golang.org):

- See golang.org/doc/code.html to set up workspace
- Create server.go
- In command line: go run server.go
- Go to localhost:8000 in your browser

Handling Multiple Routes

Handling multiple routes with net/http is annoying

- Solution: use gorilla/mux library (github.com/gorilla/mux)
- Installing: go get -u gorilla/mux
- Objective: Create webservice with 2 routes
  - /index1 - displays “hello world 1”
  - /index2 - displays “hello world 2”

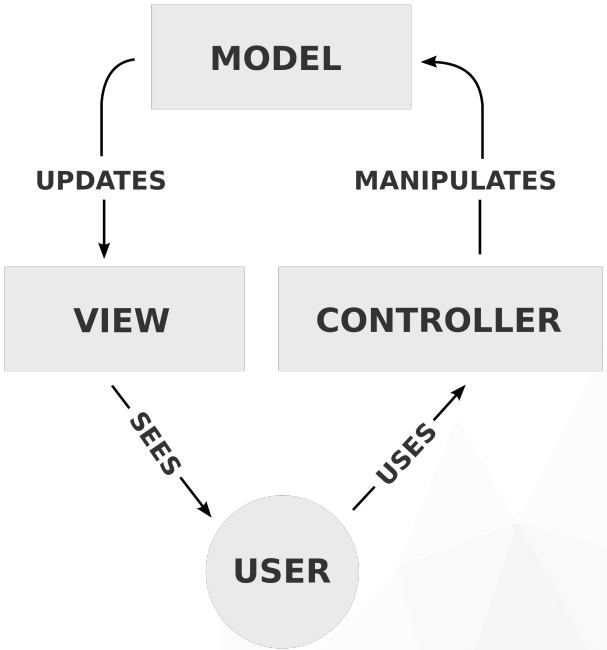
Hello World Webservice With gorilla/mux Library

Server.go

```
package main
import (
        // other imports
        "github.com/gorilla/mux"
)
func Index1(w http.ResponseWriter, r *http.Request) { fmt.Fprintf(w, "hello world 1") }
func Index2(w http.ResponseWriter, r *http.Request) { fmt.Fprintf(w, "hello world 2") }
func main() {
        router := mux.NewRouter().StrictSlash(true)
        router.HandleFunc("/index1", Index1)
        router.HandleFunc("/index2", Index2)
        log.Fatal(http.ListenAndServe(":8000", router))
}
```

Structuring Our Application

- Use MVC (Model-View-Controller) design pattern
- Application structure
  - controller/
    - todo.go
  - model/
    - todo.go
  - server.go
  - todo.json



Structuring Our Application Model

Model

- Will represent the Todo items
- Usually backed by a database (annoying to set up)
- For hackathons, use a JSON file
- Maintain a global variable of all the Todo items
- Model code will go in model/todo.go

yext

Model Code

```
package model

type Todo struct {
	Id		int		`json:"id"`
	Name		string		`json:"name"`
}


type Todos struct {
	NextId		int		`json:"nextId"`
	// Data is of type slice of pointers to Todo, keep Todos inside Data sorted by Id
	Data		[]*Todo		`json:"data"`
}
// global variable of Todos, will be serialized to and from JSON
var todos Todos
```

yext

What Is a Slice?

- An array has a fixed size and length
- Example array in Go:

```
primes := [6]int{2, 3, 5, 7, 11, 13}

// note: ":=" is a declaration and assignment
```

- Slice: dynamically sized flexible view into the elements of an array
- Declaring and assigning a slice

```
var s []int = primes[1:4] // printing s gives [3, 5, 7]
```

Model JSON Serialization And Deserialization

```
// todo.json
{
  "nextId": 0,
  "data": []
}
// model/todo.go
func LoadTodos() error {
        file, err := ioutil.ReadFile(todosFile)
        if err != nil {
                return err
        }
        return json.Unmarshal(file, &todos)
}
```

```
// model/todo.go
const (
        todosFile = "./todos.json"
        wrrPerm   = 0644
)

func saveTodos() error {
        b, err := json.Marshal(todos)
        if err != nil {
                return err
        }
        return ioutil.WriteFile(todosFile, b, wrrPerm)
}
```

Create and List Operations For Model

```
// Multiple return values!
func CreateTodo(name string) (*Todo, error) {
        todo := &Todo{
                Id:   todos.NextId,
                Name: name,
        }
        todos.NextId++
        todos.Data = append(todos.Data, todo)
        if err := saveTodos(); err != nil {
                return nil, err
        }
        return todo, nil
}
```

```
func ListTodos() []*Todo {
        return todos.Data
}
```

Update and Delete Operations For Model

```
func UpdateTodo(todo *Todo) error {
        // findTodoIndex is a user defined function
        // finds the Todo with the given id
        // and returns the index it is at in the Data slice
        idx := findTodoIndex(todo.Id)
        if idx == -1 {
                return errors.New(fmt.Sprintf(
                "Could not find todo with id: %d", todo.Id))
        }
        todos.Data[idx] = todo
        return saveTodos()
}
```

```
func DeleteTodo(id int) error {
        idx := findTodoIndex(id)
        if idx == -1 {
                return errors.New(fmt.Sprintf(
                "Could not find todo with id: %d", id))
        }
        // removes a single Todo from the slice
        todos.Data = append(
                todos.Data[:idx], todos.Data[idx+1:]...)
        return saveTodos()
}
```

yext

Structuring Our Application Controller

Controller

- Will be used to perform operations on the Todo items
- CRUD (Create, Read, Update, Delete)
- Create API endpoints for CRUD operations
- Accept JSON as input and output JSON
- Use the model CRUD operations
- Controller code will go in controller/todo.go

Create and List Operations For Controller

```
func TodoCreate(w http.ResponseWriter, r *http.Request) {
        var data struct {
                Name string `json:"name"`
        }
        err := json.NewDecoder(r.Body).Decode(&data)

        // check error is a user defined function
        if checkError(w, err) { return }
        todo, err := model.CreateTodo(data.Name)
        if checkError(w, err) { return }
        jsonResponse(w, todo)
}
```

```
func TodosList(w http.ResponseWriter, r *http.Request) {
        // anonymous structs
        data := struct {
                Data []*model.Todo `json:"data"`
        }{
                Data: model.ListTodos(),
        }
        // jsonResponse is a user defined function
        // that serializes data to json
        jsonResponse(w, data)
}
```

Update and Delete Operations For Controller

```
func TodoUpdate(w http.ResponseWriter, r *http.Request) {
        var todo model.Todo
        err := json.NewDecoder(r.Body).Decode(&todo)
        if checkError(w, err) {
                return
        }
        err = model.UpdateTodo(&todo)
        checkError(w, err)
}
```

```
func TodoDelete(w http.ResponseWriter, r *http.Request) {
        vars := mux.Vars(r)
        idString := vars["id"]
        id, err := strconv.Atoi(idString)
        if checkError(w, err) {
                return
        }
        if checkError(w, model.DeleteTodo(id)) {
                return
        }
        w.WriteHeader(http.StatusOK)
}
```

Setting Up Api Endpoints In server.go

```
func main() {
	if err := model.LoadTodos(); err != nil {
		log.Fatal("Could not load todos from file ", err)
	}
	r := mux.NewRouter().StrictSlash(true)
	r.HandleFunc("/todos", controller.TodoCreate).Methods("POST")
	r.HandleFunc("/todos", controller.TodosList).Methods("GET")
	r.HandleFunc("/todos", controller.TodoUpdate).Methods("PUT")
	r.HandleFunc("/todos/{id}", controller.TodoDelete).Methods("DELETE")
	http.Handle("/", r)
	log.Fatal(http.ListenAndServe(":8000", nil))
	// CRUD operations should now work!
}
```

Bonus: Add Mutex Locking For Concurrency Safety

```
type Todos struct {
        sync.RWMutex
        NextId int     `json:"nextId"`
        Data   []*Todo `json:"data"`
}
func CreateTodo(name string) (*Todo, error) {
        todos.Lock()
        defer todos.Unlock()
        … rest of create code
}
```

```
func ListTodos() []*Todo {
        todos.RLock()
        defer todos.RUnlock()
        return todos.Data
}
func UpdateTodo(todo *Todo) error {
        todos.Lock()
        defer todos.Unlock()
        // … rest of update code
}
func DeleteTodo(id int) error {
        todos.Lock()
        defer todos.Unlock()
        // … rest of delete code
}
```

yext

Bonus: Frontend

Use Bootstrap 4 and jQuery

- New directory, layout:
  - model/
  - controller/
  - static/
  - server.go
  - todo.json

static directory layout:

- static/
  - css/
    - main.css
  - js/
    - todo.js
  - index.html

yext

Updated For Frontend server.go

```
func main() {
        if err := model.LoadTodos(); err != nil { log.Fatal("Could not load todos from file ", err) }
        r := mux.NewRouter().StrictSlash(true)
        r.HandleFunc("/todos", controller.TodoCreate).Methods("POST")
        r.HandleFunc("/todos", controller.TodosList).Methods("GET")
        r.HandleFunc("/todos", controller.TodoUpdate).Methods("PUT")
        r.HandleFunc("/todos/{id}", controller.TodoDelete).Methods("DELETE")
        r.HandleFunc("/", controller.Index).Methods("GET")                  // serves index.html
        r.PathPrefix("/static").Handler(http.FileServer(http.Dir("./")))    // serves all the static assets
        http.Handle("/", r)
        log.Fatal(http.ListenAndServe(":8000", nil))
}
```

yext

Frontend Javascript Controller

Controller calls server.go API endpoints using AJAX, updates view

- var createCard = function(todo) { … }
- var listCards = function() { … }
- var updateCard = function(todo) { … }
- var deleteCard = function(todoId) { … }

yext

Frontend Javascript Update View Functions

Controller calls server.go API endpoints using AJAX, updates view

- var getCardHtml = function(todo) { … } // single card
- var getCardDeckHtml = function(cardsHtml) { … } // row of cards
- var renderCards = function() { … } // renders all the cards

yext

Get The Code!

github.com/harrisonzhao/simple-golang-webapp

(you might need to fix some of the import paths)

yext

Thank You!